GI-Edition

Lecture Notes in Informatics

Holger Meyer, Norbert Ritter, Andreas Thor, Daniela Nicklas, Andreas Heuer, Meike Klettke (Hrsg.)

Datenbanksysteme für Business, Technologie und Web (BTW 2019)

Workshopband

4.–8. März 2019
Rostock

Holger Meyer, Norbert Ritter, Andreas Thor, Daniela Nicklas, Andreas Heuer, Meike Klettke (Hrsg.): BTW 2019 – Workshopband

290

Proceedings

GESELLSCHAFT
FÜR INFORMATIK

Holger Meyer, Norbert Ritter,
Andreas Thor, Daniela Nicklas,
Andreas Heuer, Meike Klettke (Hrsg.)

Datenbanksysteme für Business, Technologie und Web (BTW 2019)

Workshopband

4.–8. März 2019
in Rostock, Deutschland

Gesellschaft für Informatik e. V. (GI)

Lecture Notes in Informatics (LNI) — Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-290

ISBN 978-3-88579-684-8
ISSN ISSN 1617-5468

Volume Editors

Holger Meyer
Universität Rostock
Lehrstuhl für Datenbank- und Informationssysteme
18055 Rostock, Germany
Email: hme@informatik.uni-rostock.de

Norbert Ritter
Universität Hamburg
Lehrstuhl für Datenbank- und Informationssysteme
20148 Hamburg, Germany
Email: ritter@informatik.uni-hamburg.de

Andreas Thor
Hochschule für Telekommunikation Leipzig
Datenbankmanagementsysteme
04277 Leipzig, Germany
Email: thor@hft-leipzig.de

Daniela Nicklas
Universität Bamberg
Lehrstuhl für Mobile Systeme
96047 Bamberg, Germany
Email: daniela.nicklas@uni-bamberg.de

Andreas Heuer
Universität Rostock
Lehrstuhl für Datenbank- und Informationssysteme
18055 Rostock, Germany
Email: ah@informatik.uni-rostock.de

Meike Klettke
Universität Rostock
Institut für Informatik
18055 Rostock, Germany
Email: meike.klettke@uni-rostock.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Die 18. Fachtagung "Datenbanksysteme für Business, Technologie und Web" (BTW) des Fachbereichs "Datenbanken und Informationssysteme" (DBIS) der Gesellschaft für Informatik (GI) findet vom 4. bis 8. März 2019 an der Universität Rostock statt. Pünktlich zu einem Multi-Jubiläum besucht damit **die** deutsche Datenbanktagung zum ersten Mal in ihrer Geschichte die Ostseeküste: Stadt und Universität Rostock feiern ein Doppeljubiläum (800 Jahre Stadt Rostock in 2018, 600 Jahre Universität Rostock in 2019). Daneben feiert auch die Rostocker Informatik einige Jubiläen in 2019: eine Computergrafik gibt es seit 50 Jahren an der Universität, eine Informatik seit 35 Jahren, den Lehrstuhl Datenbank- und Informationssysteme seit 25 Jahren.

Auf der BTW trifft sich nun auch schon seit fast 35 Jahren im zweijährigen Rhythmus die deutschsprachige Datenbankgemeinde, um neue Fragestellungen zu erörtern und aktuelle Forschungsergebnisse zu präsentieren und zu diskutieren. Nicht nur Wissenschaftler, sondern auch Praktiker und Anwender finden sich hier zum Wissens- und Erfahrungsaustausch zusammen. Die BTW 2019 bietet ein wissenschaftliches Programm, ein Industrieprogramm, ein Demonstrationsprogramm und ein Studierendenprogramm, dazu verschiedene Workshops und Tutorien. Zum zweiten Mal wird auf der BTW ein Wettbewerb veranstaltet, die sogenannte Data Science Challenge — in diesem Jahr zum aktuellen Thema Feinstaubbelastung in Städten.

Das Workshopprogramm umfasst diesmal folgende drei Veranstaltungen:

- "Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)"
- "Digitale Lehre im Fach Datenbanken"
- "Big (and Small) Data in Science and Humanities"

Der Workshop "Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)" wird von David Broneske, Universität Magdeburg und Dirk Habich, TU Dresden organisiert und findet zum ersten Mal statt. Das interessante Programm mit dem Schwerpunkt der Anwendung von neuen Hardware-Konzepten im Bereich der Anfrageverarbeitung, dem Data Mining, Machine Learning oder etwa dem Graph Processing umfasst neben einem Vorwort der Organisatoren eine Keynote von Erich Focht, NEC, einen eingeladenen Vortrag von Carsten Binnig sowie sechs reguläre, begutachtete Beiträge.

Thomas C. Rakow und Heide Faeskorn-Woyke organisieren den Workshop "Digitale Lehre im Fach Datenbanken". Dieser startet mit fünf Impulsbeiträgen, um dann in einer offenen Diskussion der ursprünglichen Idee von Workshops zu folgen. Die Teilnehmer wollen erarbeiten, wie die Lehre aktuell im Jahre 2019 im Fach Datenbanken aussieht und welche Erfahrungen damit gemacht wurden. Es wird auch dem Erarbeiten (gemeinsamer) Best Practices Raum gegeben werden, die im Nachgang des Workshops an geeigneter Stelle publiziert werden.

Der Workshop "Big (and Small) Data in Science and Humanities" findet zum wiederholten Male statt und wird von Friederike Klan, Birgitta König-Ries, Peter Reimann, Bernhard Seeger sowie Anika Groß organisiert. Eingeleitet wird der Workshop mit einer Keynote von Andreas Henrich zu aktuellen Anwendungen, Methoden und Herausforderungen der *Digital Humanities*. Die sechs begutachteten Beiträge reichen von temporaler Graphanalyse bis zu anwendungsspezifischer Entity-Extraktion.

Das Studierendenprogramm wurde von Andreas Thor organisiert. Im zweiten Teil dieses Bandes werden neun positiv begutachtete Beiträge vorgestellt. Die Themenvielfalt reicht von Big Data und Machine Learning bis hin zu Anfrageverarbeitung auf Graphen und Datenströmen.

Im Rahmen des Tutorienprogramms, das von Daniela Nicklas organisiert wurde, werden drei universitäre Angebote im Mittelpunkt stehen:

- *Data Analytics with Graph Algorithms — A Hands-on Tutorial with Neo4J* von Lena Wiese legt den Schwerpunkt auf Graph-Algorithmen und dem Einsatz von Graphdatenbanken.
- *StaRAI or StaRDB? A Tutorial on Statistical Relational AI* wird von Tanya Braun angeboten und fokussiert auf die Verbindung von Datenbanken mit statistisch-relationaler KI (StaRAI).
- Das Tutorial *NoSQL & Real-Time Data Management in Research & Practice* von Wolfram Wingerath und Kollegen widmet sich dem Einsatz von NoSQL in der Echtzeitdatenhaltung und bewertet diese hinsichtlich Skalierbarkeit, Verfügbarkeit, Konsistenz und weiterer Charakteristika, die der Anwender aus klassischen Datenbanksystemen gewohnt ist.

Eine Kurzdarstellung dieser Tutorien findet sich im dritten Teil dieses Bandes. Zwei Hands-on-Tutorien aus der Industrie runden das Programm der Tagung ab.

Weiterhin stellt Kai-Uwe Sattler das DFG-Schwerpunktprogramm 2037: "Scalable Data Management for Future Hardware" vor. Auf der Tagung gibt es dazu einen Vortrag in der Sitzung zu "Challenges in Data Processing" und eine anschließende Poster-Präsentation.

Bereits zum zweiten Mal findet auf der BTW die Data Science Challenge statt, auf der in diesem Jahr fünf Forschergruppen aus Deutschland gegeneinander antreten. In der am 4. Februar 2019 gestarteten Endrunde müssen die Teilnehmer innerhalb eines Monats Analysen hinsichtlich der Feinstaubbelastung unter Einbezug heterogener Datenquellen erstellen. Die fünf Lösungsansätze aus der Vorrunde und eine kurze Beschreibung der diesjährigen Data Science Challenge finden sich hier im Workshop-Band.

Die Informationen und Materialien zur BTW 2019 stehen über die Web-Seiten der Tagung unter `https://www.btw2019.de` zur Verfügung. Die Organisation der BTW-Tagung nebst allen angeschlossenen Veranstaltungen ist nicht ohne die Unterstützung vieler Partner

möglich. Diese sind auf den folgenden Seiten aufgeführt. Zu ihnen zählen insbesondere alle Sponsoren, als Ko-Veranstalter die Universität Rostock und als Unterstützer das Steinbeis-Transferzentrum DBIS an der Universität Rostock. Organisiert wurde die BTW 2019 vom Lehrstuhl Datenbank- und Informationssysteme der Universität Rostock. Insbesondere aber gilt ein Dank der GI-Geschäftsstelle für die finanzielle Abwicklung der Tagung.

Vielen Dank an alle Beteiligten!

Rostock, im Februar 2019

Holger Meyer und Norbert Ritter, Leitung Workshopkomitee

Andreas Thor, Leitung Studierendenprogramm

Daniela Nicklas, Leitung Tutorienprogramm

Andreas Heuer und Meike Klettke, Tagungsleitung

Tagungsleitung

Andreas Heuer, Universität Rostock
Meike Klettke, Universität Rostock

Organisationskomitee

Vorsitz: Holger Meyer, Universität Rostock

Tanja Auge, Universität Rostock
Hannes Grunert, Universität Rostock
Andreas Heuer, Universität Rostock
Sigrun Hoffmann, Universität Rostock
Meike Klettke, Universität Rostock
Dennis Marten, Universität Rostock
Mark Lukas Möller, Universität Rostock
Donald Reebs, Universität Rostock

Koordination Workshops

Hoger Meyer, Universität Rostock
Norbert Ritter, Universität Hamburg

Studierendenprogramm

Vorsitz: Andreas Thor, Hochschule für Telekommunikation Leipzig

Felix Gessert, Universität Hamburg
Anika Groß, Hochschule Anhalt
Harald Kosch, Universität Passau
Thomas Rakow, Hochschule Düsseldorf
Eike Schallehn. Universität Magdeburg
Uta Störl, Hochschule Darmstadt

Tutorialprogramm

Daniela Nicklas, Universität Bamberg

Data Science Challange — Preiskomitee

Vorsitz: Holger Meyer, Universität Rostock

Stefan Goers, TÜV Nord (Umweltservices)
Daniela Nicklas, Universität Bamberg
Kai-Uwe Sattler, TU Ilmenau
Holger Schwarz, Universität Stuttgart
Tim Waizenegger, IBM Böblingen
Rajko Zschiegner, OKLab Stuttgart

Workshop on Big (and Small) Data in Science and Humanities

Vorsitz: Anika Groß, Hochschule Anhalt
Friederike Klan, DLR Institut für Datenwissenschaften
Birgitta König-Ries, Friedrich-Schiller-Universität Jena
Peter Reimann, Universität Stuttgart
Bernhard Seeger, Philipps-Universität Marburg

Alsayed Algergawy, Universität Jena
Peter Baumann, Universität Bremen
Matthias Bräger, CERN
Thomas Brinkhoff, FH Oldenburg
Michael Diepenbroek, Universität Bremen
Jana Diesner, University of Illinois at Urbana-Champaign
Johann-Christoph Freytag, Humboldt-Universität zu Berlin
Michael Gertz, Universität Heidelberg
Thomas Heinis, Imperial College London
Andreas Henrich, Universität Bamberg
Jens Kattge, Max-Planck-Institut für Biogeochemie
Alfons Kemper, TU München
Bertram Ludaescher, University of Illinois at Urbana-Champaign
Alexander Markowetz, Universität Bonn
Jens Nieschulze, Universität Göttingen
Eric Peukert, Universität Leipzig
Norbert Ritter, Universität Hamburg
Kai-Uwe Sattler, TU Ilmenau
Holger Schwarz, Universität Stuttgart
Uta Störl, Hochschule Darmstadt
Andreas Thor, HfT Leipzig

Workshop Digitale Lehre im Fach Datenbanken

Thomas C. Rakow, Hochschule Düsseldorf
Heide Faeskorn-Woyke, Technische Hochschule Köln

1st Workshop on Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)

Vorsitz: David Broneske, Universität Magdeburg
Dirk Habich, TU Dresden

Steering Committee

Wolfgang Lehner, TU Dresden
Gunter Saake, Universität Magdeburg
Kai-Uwe Sattler, TU Ilmenau

Program Committee

Carsten Binnig, TU Darmstadt
Sebastian Breß, DFKI Berlin
Matthias Böhm, IBM Almaden, Uni Graz
David Broneske, Universität Magdeburg
Dirk Habich, TU Dresden
Constantin Pohl, TU Ilmenau
Hannes Rauhe, SAP SE
Knut Stolze, IBM Germany
Jens Teubner, TU Dortmund

Inhaltsverzeichnis

1st Workshop on Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)

Workshop Digitale Lehre im Fach Datenbanken

Workshop on Big (and Small) Data in Science and Humanities (BigDS 2019)

Preface

Workshop Papers

Studierendenprogramm

Tutorienprogramm

Vorstellung DFG-Schwerpunktprogramm 2037

Data Science Challenge 2019

Vorwort

Teilnehmer der Challenge

Autorenverzeichnis

1st Workshop on Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)

Preface

1st Workshop on Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)

David Broneske[1], Dirk Habich[2]

The key objective of database systems is to reliably manage data, where high query throughput and low query latency are core requirements. To satisfy these requirements, database system constantly adapt to novel hardware features. Following that trend, the focus of this one-day workshop is to explore challenges and opportunities of data processing on existing and future heterogeneous hardware architectures. In detail, today's processors are no longer mainly bound by the density and frequency of transistors, but by their power and heat budgets. The so-called "power wall"forces hardware suppliers to rely more on the design of specialized devices optimized for certain types of calculations, which results in an increasingly heterogeneous hardware landscape. Therefore, to meet the above mentioned requirements in our data-driven world, tomorrow's database systems will have to exploit and embrace this increased heterogeneity.

The purpose of this workshop is to assist in the training and growth of a community of researchers and industry practitioners working on data (co-)processing issues on heterogeneous systems. To this end, we want to provide a forum to discuss challenges, progress and directions, while creating an environment for networking with people working on related topics and fostering future collaborations. Especially in the presence of the SPP 2037 on *Scalable Data Management for Future Hardware*, we want to strengthen collaborations beyond single SPP projects by bringing them into contact with other researchers. Moreover, the workshop is co-organized by the GI-Arbeitskreis *Data Management on Modern Hardware.*

The scope of the workshop includes, but is not limited to:

- Applications of modern hardware in
  - data mining
  - data-intensive machine learning
  - query processing
  - non-traditional applications (e.g. graph processing)

[1] University of Magdeburg, Germany, david.broneske@ovgu.de
[2] TU Dresden, Germany, dirk.habich@tu-dresden.de

doi:10.18420/btw2019-ws-01

- Algorithms and data structures for efficient data processing on and across different (co-)processors (e.g., GPUs, APUs, Accelerator cards, FPGAs)
- Exploitation of specialized ASICs
- Efficient memory management, data placement and data transfer strategies in heterogeneous systems
- Energy efficiency in heterogeneous (co-)processor environments
- Programming models and hardware abstraction mechanisms for writing data-intensive algorithms on heterogeneous hardware
- Query optimization, cost estimation and operator placement strategies for heterogeneous hardware
- Transaction processing in heterogeneous systems

With the given scope of the workshop, we are happy to announce a great program. The workshop starts with a keynote by Eric Focht from NEC Deutschland GmbH. He gives an insight into their new vector engine NEC Aurora TSUBASA and how to exploit the design for data management tasks. From the submissions, we were able to accept three technical papers (presented in Session 2) as well as four extended abstracts (presented in Session 3). In Session 2, the first talk by Pietrzyk et al. investigate the SX-Aurora TSUBASA processor for data intensive operations. Furthermore, Becher et al. present a query processing platform for a heterogeneous CPU/FPGA hardware system with a special focus on query placement strategies in such a heterogeneous system. The third technical paper by Götze et al. focuses on problems and possible design decisions for a transactional stream processing system on modern heterogeneous hardware. They especially consider many-core CPUs and non-volatile memory (NVM) and include some considerations on high bandwidth memory (HBM) and co-processors.

In Session 3, four extended abstracts are presented. The first (invited) extended abstract is by Carsten Binnig, who presents his CIDR 2019 paper about a new Data Processing Interface (DPI) for easy usage of RDMA in data intensive applications. Afterwards, Ziegler et al. present a scalable approach for query execution in distributed systems using RDMA under skewed workloads. The key idea is a clever data partitioning between storage and compute nodes and to enable work stealing between compute nodes. The third extended abstract is by Breßet al. presenting their VLDB Journal article. The main focus is on compiling queries for heterogeneous (co-)processors using processor-specific code optimizations to maximize the performance of these queries. The workshop program closes with the presentation by Schmidt et al. proposing an adapted data-placement algorithm for heterogeneous systems.

Last but not least, we like to thank everyone who contributed to this workshop, in particular, the authors, the reviewers, the BTW team, and all participants.

PC Chairs

- David Broneske (University of Magdeburg)
- Dirk Habich (TU Dresden)

Steering Committee

- Wolfgang Lehner (TU Dresden)
- Gunter Saake (University of Magdeburg)
- Kai-Uwe Sattler (TU Ilmenau)

Program Committee

- Carsten Binnig (TU Darmstadt)
- Sebastian Breß (DFKI Berlin)
- Matthias Böhm (IBM Almaden, Uni Graz)
- David Broneske (University of Magdeburg)
- Dirk Habich (TU Dresden)
- Constantin Pohl (TU Ilmenau)
- Hannes Rauhe (SAP SE)
- Knut Stolze (IBM Germany)
- Jens Teubner (TU Dortmund)

Invited Talk

DPI: The Data Processing Interface for Modern Networks

Carsten Binnig[1]

Extended Abstract

The computer networks available in data centers and clusters are evolving rapidly, increasingly providing sophisticated capabilities such as RDMA (Remote Direct Memory Access), in-network processing, and customizable communication protocols. Once the province of specialized, expensive networks, the new functionality is becoming available in off-the-shelf networks as well. An example of how these advances can help with data intensive applications is RDMA, the ability to directly read or write the memory of remote machines without involving the remote CPU. RDMA makes data transfer more efficient, and it frees up computing capacity, which can lead to substantial performance gains [Ka16, Dr15, Dr14, Lo15, Za17, Ou11, Mi13, Ka14, Yo18, De05, Co17]. Unfortunately, using RDMA is complicated because it lacks higher-level abstractions [Dr17]. Recent work on using RDMA in relational databases has shown that the design involves many low-level, yet significant, decisions around connection management, memory allocation, and the choice of which RDMA operations to use [Bi16, Ba15].

This fragile dependency on low-level design aspects and lack of portability across networks is not unique to RDMA; it affects other technologies like smart NICs (Network Interface Cards) and programmable switches as well [Fi18]. This is concerning because modern networks are increasingly software-defined, and there is a growing need to tailor them to data processing, e.g., through load balancing and skew detection at the switch level, data partitioning on the NIC, and content based routing. Although recent results [Bl18, Sa17] have shown that smart NICs and programmable switches can improve the performance of distributed data processing systems, the hand-tuning of low level details remains a problem. Not only is the programming of the devices complex, it also creates resource management problems such as deciding when to offload computation into the network.

In this talk, I present the Data Processing Interface (DPI) as a way to address these problems. DPI's goal is to make it easier for applications to exploit these emerging capabilities of modern networks. Accordingly, DPI defines abstractions and interfaces suited to a broad class of data-intensive applications, yet simple enough for practical implementation with predictable performance and low overhead relative to "hand-tuned", ad hoc alternatives. In designing an interface tailored to data processing, we adopt the approach taken by other high-level interfaces, such as MPI (Message Passing Interface) [Gr14], which have been designed for other application domains and which, consequently, have seen only limited adoption for data processing [Ba17]. A detailed paper about DPI has recently been presented at the CIDR'19 conference [Al19].

[1] TU Darmstadt, Data Management Lab - Informatik, Germany, carsten.binnig@cs.tu-darmstadt.de

doi:10.18420/btw2019-ws-02

References

[Al19] Alonso, Gustavo et al.: DPI: The Data Processing Interface for Modern Networks. In: CIDR 2019. 2019.

[Ba15] Barthels, Claude et al.: Rack-Scale In-Memory Join Processing using RDMA. In: ACM SIGMOD. pp. 1463–1475, 2015.

[Ba17] Barthels, Claude et al.: Distributed Join Algorithms on Thousands of Cores. PVLDB, 10(5):517–528, 2017.

[Bi16] Binnig, Carsten et al.: The end of slow networks: It's time for a redesign. PVLDB, 9(7):528–539, 2016.

[Bl18] Blöcher, Marcel et al.: Boosting scalable data analytics with modern programmable networks. In: ACM DaMoN@SIGMOD. ACM, pp. 1:1–1:3, 2018.

[Co17] Chen, Haibo; other: Fast In-Memory Transaction Processing Using RDMA and HTM. ACM Trans. Comput. Syst., 35(1):3:1–3:37, 2017.

[De05] Devulapalli, Ananth et al.: Distributed Queue-Based Locking Using Advanced Network Features. In: ICPP. pp. 408–415, 2005.

[Dr14] Dragojević, Aleksandar et al.: FaRM: Fast remote memory. In: NSDI. pp. 401–414, 2014.

[Dr15] Dragojević, Aleksandar et al.: No compromises: distributed transactions with consistency, availability, and performance. In: OSDI. pp. 54–70, 2015.

[Dr17] Dragojevic, Aleksandar et al.: RDMA Reads: To Use or Not to Use? IEEE Data Eng. Bull., 40(1):3–14, 2017.

[Fi18] Firestone, Daniel et al.: Azure Accelerated Networking: SmartNICs in the Public Cloud. In: NSDI. pp. 51–66, 2018.

[Gr14] Gropp, William et al.: Using Advanced MPI: Modern Features of the Message-Passing Interface. The MIT Press, 2014.

[Ka14] Kalia, Anuj et al.: Using RDMA efficiently for key-value services. In: Proc. of ACM SIGCOMM. pp. 295–306, 2014.

[Ka16] Kalia, Anuj et al.: FaSST: fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs. In: Proc. of OSDI. pp. 185–201, 2016.

[Lo15] Loesing, Simon et al.: On the Design and Scalability of Distributed Shared-Data Databases. In: ACM SIGMOD. pp. 663–676, 2015.

[Mi13] Mitchell, Christopher et al.: Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In: Proc. of USENIX ATC. pp. 103–114, 2013.

[Ou11] Ousterhout, John et al.: The case for RAMCloud. Communications of the ACM, 54(7):121–130, 2011.

[Sa17] Sapio, Amedeo et al.: DAIET: a system for data aggregation inside the network. In: SoCC. ACM, p. 626, 2017.

[Yo18] Yoon, Dong Young et al.: Distributed Lock Management with RDMA: Decentralization without Starvation. In: ACM SIGMOD. pp. 1571–1586, 2018.

[Za17] Zamanian, Erfan et al.: The End of a Myth: Distributed Transaction Can Scale. PVLDB, 10(6):685–696, 2017.

Workshop Papers

First Investigations of the Vector Supercomputer SX-Aurora TSUBASA as a Co-Processor for Database Systems

Johannes Pietrzyk[1], Dirk Habich[1], Patrick Damme[1], Wolfgang Lehner[1]

Abstract: The hardware landscape is currently changing from homogeneous multi-core systems towards heterogeneous systems with many different computing units, each with their own characteristics. This trend is a great opportunity for database systems to increase the overall performance if the heterogeneous resources can be utilized efficiently. Following that trend, NEC cooperation has recently introduced a novel heterogeneous hardware system called SX-Aurora TSUBASA. This novel heterogeneous system features a strong vector engine as a (co-)processor providing world's highest memory bandwidth of 1.2TB/s per vector processor. From a database system perspective, where many operations are memory bound, this bandwidth is very interesting. Thus, we describe the unique architecture and properties of this novel heterogeneous system in this paper. Moreover, we present first database-specific evaluation results to show the benefit of this system to increase the query performance. We conclude the paper with an outlook on our ongoing research activities in this direction.

Keywords: Database Systems; Heterogeneous Hardware; Vector Processor

1 Introduction

In our digital world, efficient query processing is still an important aspect due to the ever-growing amount of data. To satisfy query response times and query throughput demands, the architecture of database systems is constantly evolving [BKM08, HZH14, KHL17, Li16, Ou17]. For instance, the database architecture shifted from a disk-oriented to a main memory-oriented architecture to efficiently exploit the ever-increasing capacities of main memory [Ab13, Id12, Ki13, St05]. This in-memory database architecture is now state-of-the-art and characterized by the fact, that all relevant data is completely stored and processed in main memory. Additionally, relational tables are organized by column rather than by row [Ab13, BKM08, CK85, Id12, St05] and the traditional tuple-at-a-time query processing model was replaced by newer and adapted processing models like column-at-a-time or vector-at-a-time [Ab13, BKM08, Id12, St05, ZvdWB12].

To further increase the performance of queries, in particular for analytical queries in these in-memory column stores, two key aspects play an important role. On the one hand, data

[1] Technische Universität Dresden, Institut for Systems Architecture, Dresden Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, firstname.lastname@tu-dresden.de

doi:10.18420/btw2019-ws-03

compression is used to tackle the continuously increasing gap between computing power of CPUs and memory bandwidth (also known as memory wall [BKM08]) [AMF06, BHF09, Da17, Hi16, Zu06]. Aside from reducing the amount of data, compressed data offers several advantages such as less time spent on load and store instructions, a better utilization of the cache hierarchy, and less misses in the translation lookaside buffer. On the other hand, in-memory column stores constantly adapt to novel hardware features like vectorization using Single-Instruction Multiple Data (SIMD) extensions [PRR15, ZvdWB12], GPUs [HZH14, KML15] or non-volatile main memory [Ou17].

From a hardware perspective, we currently observe a shift from homogeneous CPU systems towards hybrid systems with different computing units mainly to overcome physical limits of homogeneous systems [Es12, LUH18]. Following that hardware trend, NEC cooperation recently released a novel heterogeneous hardware system called SX-Aurora TSUBASA [Ko18]. The main advantage of this novel hardware system is its strong vector engine which provides world's highest memory bandwidth of up to 1.2TB/s per vector processor [Ko18]. From that point, this novel hardware could be very interesting for database systems. In particular, from the following aspects:

1. Vectorization is a hot-topic in database systems to improve query processing performance by parallelizing computations over vector registers [PRR15, ZvdWB12]. Intel's latest vector extension is AVX-512 with vector registers of size 512 bits. In contrast to that, the vector engine of SX-Aurora TSUBASA features a vector length of 216KB (16.384 bits).
2. SX-Aurora TSUBASA offers enough memory bandwidth to fill these large vectors with data for efficient processing.

Therefore, we describe the unique architecture and properties of this novel heterogeneous system in this paper (Section 2). Moreover, we present first database-specific evaluation results to show the benefit of this system to increase the query performance in Section 3. Based on that, we briefly introduce our ongoing research activities in this direction in Section 4. We conclude the paper in Section 5 with a short summary.

2 Vector Supercomputer SX-Aurora TSUBASA

NEC Cooperation has a long tradition in vector supercomputers with a series of NEC SX models starting 1983. The current model is NEC SX-Aurora TSUBASA. In the following sections, we will describe the overall architecture, the vector processing and the programming approach of this novel SX-Aurora TSUBASA model.

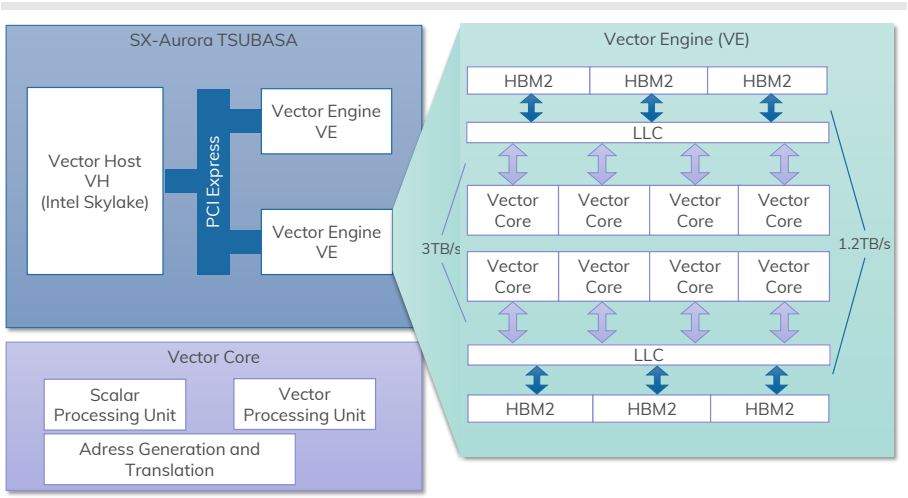


Fig. 1: SX-Aurora TSUBASA Architecture.

2.1 Overall Architecture

The overall architecture of SX-Aurora TSUBASA completely differs from its predecessors in the SX series. The new system model is a heterogeneous system consisting of a vector host (VH) and one or more vector engines (VE). As illustrated in Figure 1, the VH is a regular Intel Xeon Skylake CPU featuring a standard x86 Linux server that provides standard operating systems functions. Moreover, the VH also includes a special operating system for the VE called *VEOS* which runs in the user mode of the VH. *VEOS* controls the VE, whereby each VE is implemented as a PCI Express (PCIe) card equipped with a newly developed vector processor [Ko18].

As illustrated in Figure 1 on the right side, each vector processor consists of 8 vector cores, 6 banks of HBM2[2] high speed memory, and only one shared last-level cache (LLC) of size 16MB between memory and the vector cores. The LLC is one both sides of the vector cores, and it is connected to each vector core through a 2D mesh network-on-chip with a total cache bandwidth of 3TB/s [Ko18]. Moreover, this vector processor design provides the world highest memory bandwidth of up to 1.2TB/s per vector processor [Ko18]. Each vector core consists of three core units: (i) a scalar processing unit (SPU), a vector processing unit (VPU), and (iii) a memory addressing vector control and processor network unit (AVP). The SPU has almost the same functionality as modern processors such as fetch, decode, branch, add, and exception handling. However, the main task of the SPU is to control the status of the vector core.

[2] High Bandwidth Memory Version 2

Type	Frequency	DP Performance of a core	DP Performance of a Processor	Memory Bandwidth	Memory Capacity
VE 10A	1.6 GHz	307.2Gflop/s	2457.6Gflop/s	1228.8GB/s	48 GB
VE 10B	1.4 GHz	268.8.2Gflop/s	2150.4Gflop/s	1228.8GB/s	48 GB
VE 10C	1.4 GHz	268.8.2Gflop/s	2150.4Gflop/s	750.0GB/s	24 GB
VH Intel Xeon Gold 6126	2.5GHz	83.2Gflops/s	998.4Gflops/s	128GB/s	96GB

Tab. 1: Specifications for SX-Aurora TSUBASA.

2.2 Vector Processing and Specific Systems

Besides the high bandwidth, the architecture of the VPU of the vector core is a further advantage of this processor. The VPU has three vector fused multiply add units, which can be independently executed by different vector instructions, whereby each unit has 32 vector pipelines consisting of 8 stages [Ko18]. Generally, the vector length of the VPU is 256 elements[3], each of which is 8 Byte [Ko18]. One vector instruction executes 256 arithmetic operations within eight cycles [Ko18]. The major advantage, compared to wider SIMD functionalities e.g., in Intel processors like AVX-512, is that the operations are not only executed spatially parallel, but also temporally parallel which better hides memory latency [Ko18]. Each VPU has 64 vector registers and each vector register is 2Kb in size (32 pipeline elements with 8 Byte per element). Thus, the total size of the vector registers is 128Kb per vector core, which is larger than a L1 cache in modern regular processors. To fill these large vector registers with data, the LLC is directly connected to the vector registers and the connection has roughly 400GB/s bandwidth per vector core [Ko18].

Generally, NEC offers three types of these VE called 10A, 10B, and 10C as shown in Table 1, which only differs in frequency, memory bandwidth and memory capacity. In every case, the VH is an Intel Xeon Gold 6126 with 12 cores. Table 1 also compares VE and VH with respect to double precision (DP) performance per core and per processor as well as memory bandwidth and memory capacity. As we can see, memory bandwidth of each VE is many times higher than that of the VH, but maximum memory capacity of the VE is 48GB.

The SX-Aurora TSUBASA approach has a high level configuration flexibility and the series includes three product types:

- A100 is a workstation model with one VH and one VE.
- A300 is standard rack-mount model with up to eight VE with one VH. In this case, the maximum size of the vector main memory is 384GB.

[3] In comparison, the vector length of Intel's latest vector extension AVX-512 is limited to 8 elements with 8 Byte per element.

- A500 is designed as large-scale supercomputer with up to eight A300 models connected which results in maximum vector main memory capacity of 3.072GB.

With these memory capacities and bandwidths, this heterogeneous system approach is very interesting for memory-intensive applications such as database management systems.

2.3 Execution Model and Programming Approach

Unlike other accelerators, SX-Aurora TSUBASA is pursuing a different execution model. In general, the VE is entirely responsible for executing applications, while the VH provides basic OS functionalities such as process scheduling and handling of system calls invoked by the applications on the VE [Ko18]. Applications for the VE are written in standard programming languages such as C, C++ or Fortran without having to use special programming models. For this, a C library compliant with standards is ported to VE [Ko18]. Therefore, existing (non-vectorized) applications can be ported to VE just by recompiling using the NEC compiler.

In summary, the high bandwidth of the SX-Aurora TSUBASA VE and big vector registers are a promising combination for improving query processing performance in database systems.

3 Database-oriented Evaluation

Since many in-memory database operations are memory bound, we basically have focused our evaluation on examining the specified memory throughput of the introduced hardware. Therefore, we measured plain sequential memory access in a first step, followed by a column-scan which can be considered as a fundamental physical query operator.

3.1 Investigated Operators

Fundamentally, memory access can be distinguished between reading from memory, writing to memory and copying data. As a first step in our evaluation, we focused on these core primitives. While reading from memory without any further operations can be considered to be optimized out by the compiler, an aggregation is performed over the read memory using the bitwise *OR* operator. Thus, only one cache-resident value has to be updated per actual read. Given a relatively fast aggregation operation with regard to the memory access, it can be assumed that the measured throughput is not distorted by computation effort.

To measure the behaviour of write intense sequential memory access, we filled a given array with a constant value, like a *memset*. As a combination of both classes of memory access

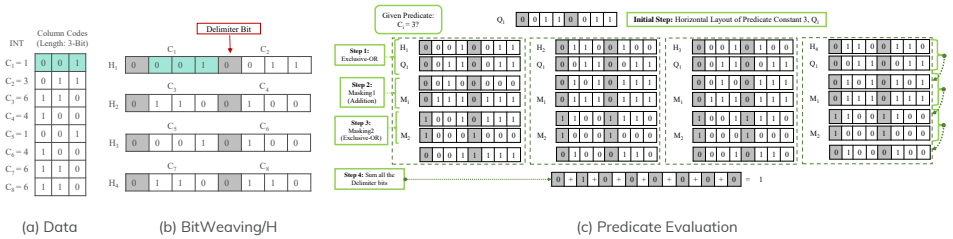


Fig. 2: Illustration of BitWeaving/H (taken from [Li18]).

patterns, we measured the throughput of copying the values from a given array into another array.

As a second step and as a more database relevant I/O bound operation, we evaluated a column scan operation using a state-of-the-art approach called BitWeaving/H [LP13]. Fundamentally, BitWeaving assumes a fixed-length order preserving compression scheme, so that all compressed column codes of a column have the same bit length [LP13]. Then, the bits of the column codes are aligned in main memory in a way that enables the exploitation of intra-cycle parallelism using ordinary processor words. An example is shown in Figure 2(a), where eight 32-bit integer values C_i are represented using 3-bit compressed column codes. As illustrated in Figure 2(b), the column codes are contiguously stored in processor word H_i in BitWeaving/H, where the most significant bit of every code is used as a delimiter bit between adjacent column codes. In our example, we use 8-bit processor words, so that two 3-bit column codes fit into one processor word including one delimiter bit per code. The delimiter bits are used later to store the result of a predicate evaluation.

Now, the task of a column scan is to compare each column code with a constant C and to output a bit vector indicating whether or not the corresponding code satisfies the comparison condition. To efficiently perform such a column scan using the BitWeaving/H, Li et al. [LP13] proposed an arithmetic framework to directly execute predicate evaluations on the compressed column codes. There are two main advantages: (i) predicate evaluation is done without decompression and (ii) multiple column codes are simultaneously processed within a single processor word using full-word instructions (intra-cycle parallelism) [LP13]. The supported predicate evaluations include equality, inequality, and range checks, whereby for each evaluation a function consisting of arithmetical and logical operations is defined [LP13].

Figure 2(c) highlights the *equality* check in an exemplary way, whereby the other predicate evaluations work in a similar way. The input from Figure 2(b) is tested against the condition $C_i = 3$. Then, the predicate evaluation steps are as follows:

Initially: All given column codes and the query constant number *3* are converted into the BitWeaving/H storage layout (H_1, H_2, H_3, H_4) and Q_1, respectively.

Type	Max. vector size	Cache accessible from vector reg.	OS	Compiler
VE 10 B/C	16384 Bit (256 · 64 Bit)	16 MB LLC	VEOS 1.3.2	nc++ 1.6.0
VH Intel Xeon Gold 6126	512 Bit (8 · 64 Bit)	32 KB L1 1 MB L2 19.25 MB L3	CentOs 7.5	g++ 7.3.1

Tab. 2: Specification for hardware, operating system and compiler used for experiments.

Step 1: An *Exclusive-OR* operation between the words (H_1, H_2, H_3, H_4) and Q_1 is performed.

Step 2: *Masking1* operation (*Addition*) between the intermediate results of Step 1 and the M_1 mask register (where each bit of M_1 is set to one, except the delimiter bits) is performed.

Step 3: *Masking2* operation (Exclusive-OR) between the intermediate results of Step 2 and the M_2 mask register (where only delimiter bits of M_2 are set to one and the rest of all bits is set to zero) is performed.

Step 4 (optional): Add delimiter bits to achieve the total count (final result).

The output is a result bit vector, with one bit per input code that indicates if the code matches the predicate on the column. In our example in Figure 2, only the second column code (C_2) satisfies the predicate which is visible in the resulting bit vector.

3.2 Experimental Setup

All operations were measured on two different versions of the SX-Aurora TSUBASA co-processor. General specifications are denoted in Table 2. To compile the implemented operators for the VH system, a gcc 7.3.1 was used with the optimization flags *-O3 -flto* and disabled auto-vectorization (*-fno-tree-vectorize*). For the VE, the proprietary NEC compiler nc++ 1.6.0 was used with the optimization flag *-O3 -fipa* and *-mvector*, enabling the auto-vectorization. A distinction between single-thread performance and multi-thread performance were made by linking the binary with and without OpenMP. To minimize the runtime overhead through dynamic linking, all files were linked statically.

3.3 Experimental Methodology

The time measurements were performed using a c++ wall-time clock on the VH and inline assembly for retrieving user clock cycles on the VE. While an experiment consists of a measurement of all specified task, every experiment was repeated 10 times and and the reported runtimes are averaged. To avoid distortion by the actual time measurement, all

(a) C++ Code for Aggregation

```
1  /*...*/
2  #pragma omp parallel
3  {
4  #pragma _NEC vreg(aRes)
5  #pragma omp for
6  #pragma _NEC nooouterloop_unroll
7   for(;i<nBuf;i+=256){
8  #pragma _NEC shortloop
9     for(j=0;j<256;++j){
10     aRes[j]|=aSrc[i+j];
11    }
12   }
13   for(k=0;k<256;++k){
14    nRes|=aRes[k];
15   }
16  }
17  /*...*/
```

(b) nc++ Listing after compilation

```
1  LINE DIAGNOSTIC MESSAGE
2     1: Vector reg. assigned.: aRes
3     2: Parallel routine generated.
4     7: Parallelized by "for".
5     9: Vectorized loop.
6    13: Vectorized loop.
7    14: Idiom detected.: Bit-op
8  /*...*/
9  LINE LOOP STATEMENT
10 /*...*/
11    7- P---> for(;i<nBuf;i+=256){
12    8- |       #pragma _NEC shortloop
13    9- |V-->    for(j=0;j<256;++j){
14   10- ||  V      aRes[j]|=aSrc[i+j];
15   11- |V--     }
16   12- P---   }
17   13- V---> for(k=0;k<256;++k){
18   14- |    V   nRes|=aRes[k];
19   15- V---> }
20 /*...*/
```

Fig. 3: Excerpt of C++-Code for read intense task and corresponding diagnostic listing, produced by nc++ while compilation with optimization indications.

tasks were repeated multiple times and the accumulated time was divided by the amount of repetitions.

Thus the focus was on evaluating the computing performance of vectorized code alongside the memory bandwidth, all tasks were implemented using vectorization. This was done using intrinsics for the VH. To examine the best performance of the different SIMD extensions offered by the VH, all tasks were implemented and tested using either SSE, AVX2 or AVX512.

As shown in Figure 3(a), a combination of compiler specific preprocessor pragma directives, strip mining and local buffers were used for the VE to facilitate the auto-vectorization feature of the NEC compiler. At first, the main loop which iterates over the buffer as a whole is fragmented into strips according to the size of a vector register (see Line 7). To prevent the compiler from undo the so called loop strip-mining, an according pragma was used (see Line 6). The most inner loop, containing the operator specific instructions, is marked as a shortloop (see Line 8) giving the compiler a hint that the loop should completely be transformed into vector code. In addition, the specific instructions work on temporal buffers which are forcedly assigned to a vector register using *#pragma _NEC vreg(arrayName)* (see Line 4). First measurements showed that this specific hint can significantly improve the performance of the algorithm. OpenMP were introduced through parallel regions using *#pragma omp parallel* (see Line 2) alongside loop parallelism using *#pragma omp for*

VE VPU	VH	Buffer sizes				
LLC	L1	16 KB	32 KB			
	L2	512 KB	1 MB			
	L3	4 MB	8 MB	16 MB		
HBM2	RAM	32 MB	128 MB	1 GB	4 GB	8 GB

Tab. 3: Measured buffer sizes.

depicted at Line 5. When compiling the annotated C++-Code using the NEC compiler, a diagnostic listing, indicating all applied optimization's, can be generated (see Figure 3(b)).

Within the VE, two different element sizes (32, 64 Bit) were measured. Within the VH, different load and store modes (stream, aligned, unaligned) were measured. To evaluate a possible influence of different memory structures on the performance, every experiment were executed while processing different sizes of data (see Table 3).

3.4 Evaluation

The measured time needed for executing the described tasks were used to calculate the average throughput per task. Except for the copying-task the read and write intense tasks either read one buffer or store one buffer. The column scan reads one buffer. Thus the processed buffer size was used for calculating the throughput. When it comes to copying, actually every element from a source buffer is read and stored into the destination buffer. Taken this into consideration the number of processed elements would be the sum of the source and destination buffer sizes. However, the throughput of a copy-operation is typically deduced directly from the size of the source buffer. Thus only this quantity was used to calculate the throughput.

3.4.1 Single-Thread Evaluation

Figure 4 shows the throughput of plain memory access measured on the VH (a)-(c) as well as on the VE (d)-(f). Using the VH, a maximum throughput of around 100 GB/s were obtained when the processed data fits completely into L1. In general the performance gets lower if the buffer sizes exceed the cache sizes. While read intense tasks can utilize L2 without significant performance penalties, write intense tasks suffer from accessing higher levels of cache. Conversely, the throughput measured on the VE gets better with bigger buffer size obtaining an overall maximum throughput of around 300 GB/s (processing 1MB) for write intense tasks (see Figure 4(e)) and 250 GB/s (processing 2MB) while executing read intense tasks (see Figure 4(f)). Accessing the HBM2 leads to a marginal decrease in measured throughput.

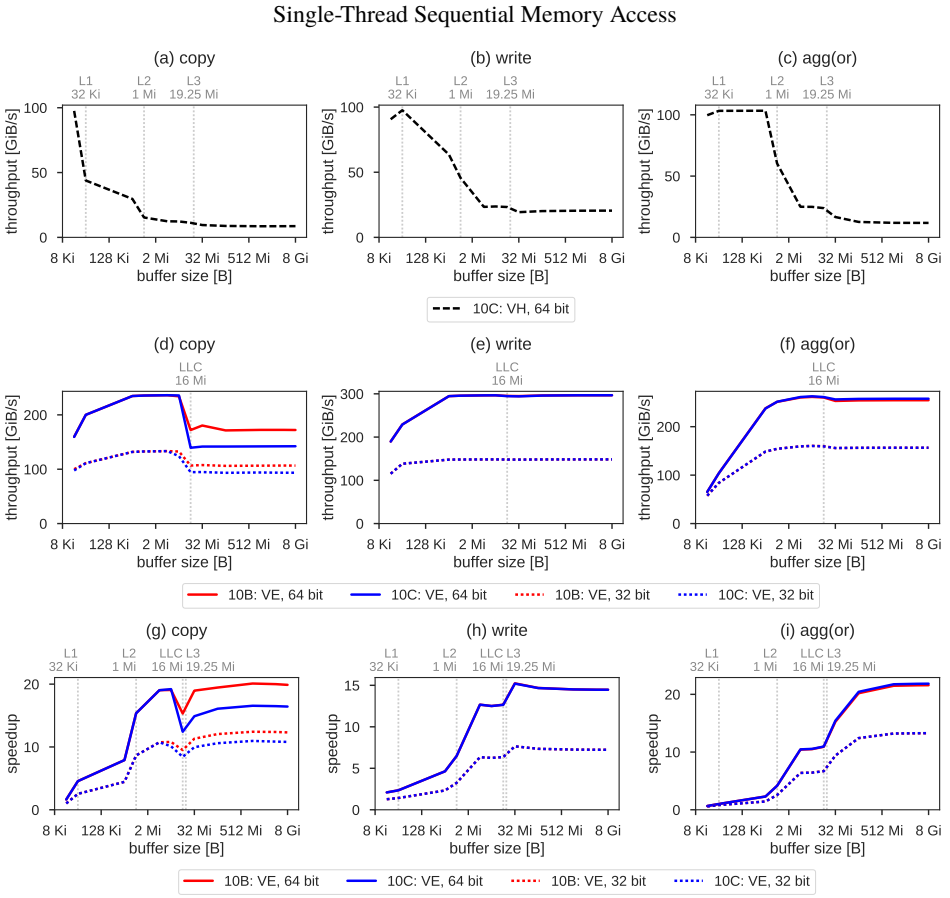


Fig. 4: Measured throughput of VH and engine as well as the speedup obtained by the VE of different IO-operations executed using one thread.

As shown in Figure 4(d), only the performance of a vectorized copy drops dramatically when the processed data sizes exceeds the boundaries of the existing LLC. Since the tasks were executed vectorized and a single vector register can hold up to 2 KB data, small buffers prevent the VE of using the given vector registers in an efficient manner. In general the experiments have shown that processing 64-bit wide elements led to significantly higher throughputs than working on 32-bit sized data. This results from the under-utilization of available vector registers. The vector pipeline processes its elements at a granularity of 64 bit. If only 32 bit wide elements were processed, the remaining bits left unused. An improvement in terms of the memory access could not be achieved on the formally faster TSUBASA 10B neither for vectorized write-intense nor read-intense tasks. Only the performance of the copy task could benefit from the better memory bandwidth of the 10B.

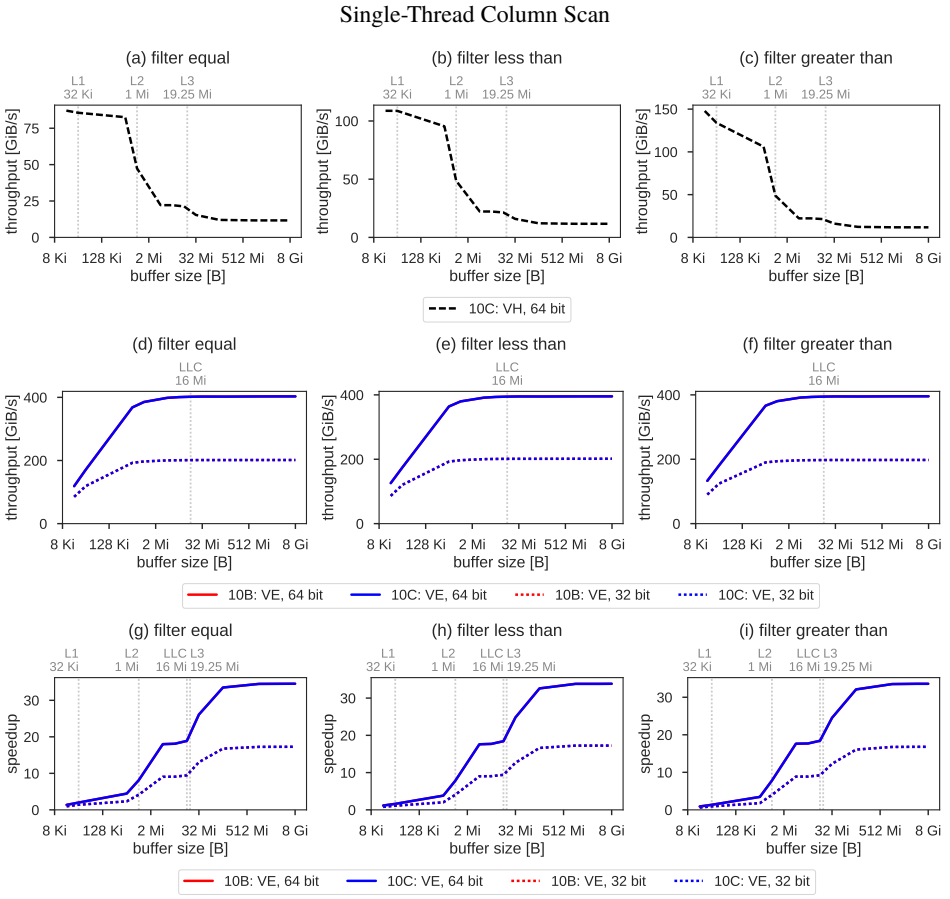


Fig. 5: Measured throughput of VH and engine as well as the speedup obtained by the VE of different Bitweaving-H operations executed using one thread.

As shown in Figure 4(g)-(i) both VE outperform the VH for write intense tasks up to a factor of 15 on the 10C and 20 on the 10B respectively. A maximum speedup of around 21 were obtained for read intense tasks when the processed data exceeds the cache and has to be loaded from DRAM (vh) or HBM2 (ve) respectively.

As mentioned in Section 3.1, the actual scan is executed using arithmetic operations. While a filter for equality needs two bitwise operators (*XOR*, *NOT*) and an addition, a filter for less than needs only one bitwise operator (*XOR*) and an addition. To scan for elements which are greater than the predicate only one addition is executed. These characteristics can be seen in Figure 5(a)-(c) where the discussed column scan operators achieve a maximum throughput in the range of 80 GB/s (Figure 5(a)) up to 150 GB/s (Figure 5(c)) when the processed data

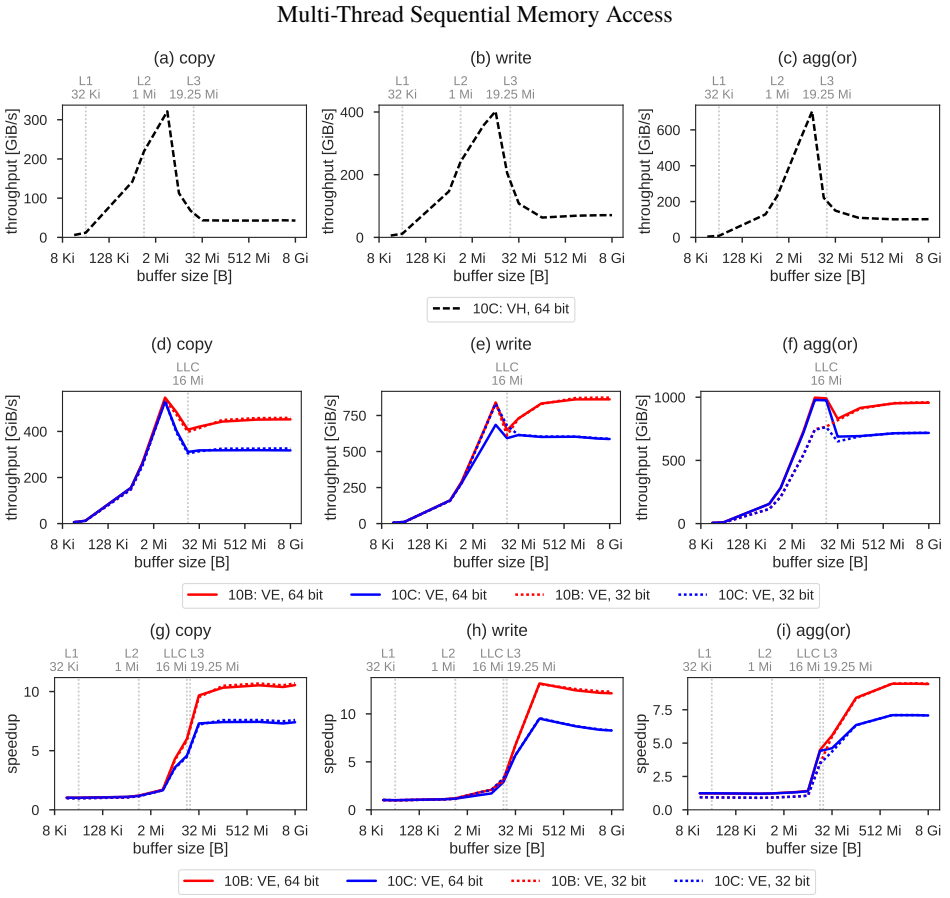


Fig. 6: Measured throughput of different IO-operations executed on the VH using multiple threads.

fits entirely into L1. Running onto the VE the total amount of executed operations does not affect the overall throughput leading to mostly similar behaviour like the write task. A maximum throughput of around 400 GB/s were reached when the processed buffer exceeds the boundaries of the LLC. As shown in Figure 5(g)-(i) both VE outperform the VH up to a factor of 20 when the processed buffer is resident in the caches. When the buffer exceeds the LLC a speedup of factor 33 could be achieved by the VE.

3.4.2 Multi-Thread Evaluation

Using multiple threads introduce additional complexity in terms of thread creation as well as data partitioning. As shown in Figure 6 this overhead leads to significant lower throughputs

for both the VH and the VE when processing small buffers. Nevertheless multithreading pays off on the VH when the processed data exceeds the L2 cache (see Figure 6(a)-(c)) obtaining a maximum throughput of around 320 GB/s for copying, 400 GB/s for writing and 700 GB/s for aggregating respectively. Taken this into account, using multiple threads for plain memory access can speedup the processing up to a factor of 7 compared with single-thread execution.

The same observation holds for the VE in terms of processing small buffers up to 2 MB. While plain memory access using a single thread reaches a maximum throughput when processing 1MB and upwards, multiple threads reach a local maximum with around 4 MB buffer size for copying and 8 MB buffer size for reading and writing respectively. For bigger buffers which still fit into the LLC, the throughput decreases probably caused by effects like cache pollution. As shown in Figure 6(d)-(f) the measured throughput of the 10C remain stable while the throughput at around 300 GB/s for copying and 600 GB/s to 700 GB/s for write and read intense tasks respectively when the processed buffer size exceeds the boundaries of the LLC. The TSUBASA 10B even outperforms the TSUBASA 10C when processing HMB resident buffers through the higher maximum bandwidth resulting in an overall maximum throughput of around 800 GB/s for writing and nearly 1 TB/s for aggregation. Using multiple threads the VE TSUBASA 10B outperforms the VH up to a factor of 10 for copying and aggregating, 13 for writing respectively.

Executing bitparallel column-scans using multiple threads show similar behaviour like the reading task by exceeding the reached throughput of single thread execution by a factor of around 2. Interestingly the bitwidth has an significant influence when running on the VE. As shown in Figure 7(d)-(f) processing 64-bit wide elements led to higher throughputs in general. This impact of processed word size decreases for processing big buffers using less operations.

3.5 Summary

The conducted experiments show that the vector co-processor SX-Aurora TSUBASA can on the one hand improve the performance of computational-bound algorithms through the utilization of wide vector registers alongside an efficient vector processing pipeline. On the other hand memory-bound algorithms can benefit from the integrated high bandwidth memory in combination with the shared LLC which is accessible from the VPU directly.

Furthermore, the existing NEC compiler with its integrated automatic vectorization feature facilitates the reuse of existing scalar C/C++-Code with only minor adjustments.

4 Future Work

As we have clearly shown in the previous section, the VE of SX-Aurora TSUBASA offers outstanding performance characteristics from a database query processing perspective. To

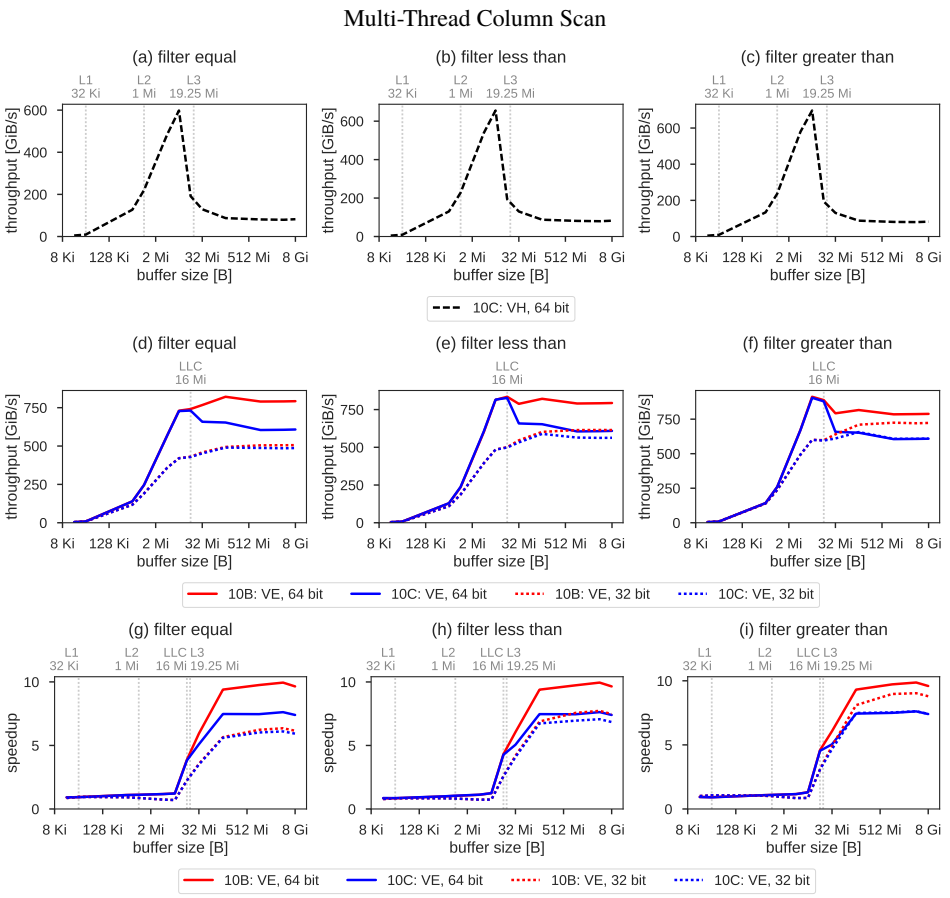


Fig. 7: Measured throughput of different Bitweaving-H operations executed on the VH using multiple threads.

further investigate this hardware system for database systems in detail, our ongoing research goes into two directions as described next.

4.1 Highly Vectorized Query Processing on Compressed Columnar Data

In-memory column store systems are state-of-the-art for the efficient processing of analytical workloads. In such systems, data compression as well as vectorization are extremely relevant aspects. On the one hand, data compression is applied to tackle the continuously increasing gap between computing power of common CPUs and main memory bandwidth (also known as memory wall). Therefore, all in-memory column store systems roughly pursue a similar

approach: (i) each column of a relational table is treated independently, (ii) values of each column are encoded as a sequence of integers usually applying dictionary coding and (iii) different data compression schemes are applied to each sequence of integers resulting in a sequence of compressed data strings. For such lossless integer compression schemes, a large variety of lightweight algorithms has been developed. On the other hand, vectorization is used to improve the processing performance by parallelizing computations over vector registers. This vectorization is usually performed using SIMD (Single Instruction Multiple Data) extensions such as Intel's SSE (Streaming SIMD Extensions) or AVX (Advanced Vector Extensions) and has been available in modern processors for several years. Up to now, Intel's latest SIMD extensions supports 512-bit vector registers. From a database perspective, this vectorization technique is not only very interesting for compression and decompression, but also for all a set of database operator like joins, scans, as well as groupings.

In contrast to Intel's SIMD extensions, the SX-Aurora TSUBASA supports vector registers with size of 16,384-bit (256x64-bit) which significantly outperforms techniques offered by other vendors today. Thus, we want to examine the effects of these large vector registers on the query processing in the context of state-of-the-art in-memory column store systems. Therefore, we plan to investigate two aspects: (i) lightweight data compression as well as decompression and (ii) high-performance and highly vectorized execution of compressed data using scans (point and range queries), joins (sort-merge as well as hash joins), and groupings (sort as well as hash-based).

4.2 Near-Memory Data Processing on Multiple Vector Engines

Modern applications demand data management systems to provide strong analytical and transactional power at the same time while facing a continuously increasing amount of data that needs to be maintained and processed. To challenge those issues, modern data management systems need scale up on today's massively parallel hardware and need to be capable to adapt to the current workload by changing data layout, data partitioning, and hardware resource configurations on-the-fly without negatively affecting the running queries. Since traditional disk-based or in-memory database systems were never designed to scale to the degree of parallelism that is offered by modern hardware, scale-out solutions such as Apache Spark came up to address the issue of analytical power by assuming a read-only data access similar to batch processing setups on HPC systems. With the ERIS data management system [Ki14, KHL18], we already invented a novel near-memory computing-optimized database system architecture that aims at scalability, adaptivity, and a superior absolute performance compared to systems like Spark focusing on scale-up hardware systems.

Currently, our ERIS data management system is optimized for traditional x86 based server hardware. The main objective of this research direction is to port the current C++ implementation to entirely run on multiple VE. In particular, we will address the following research questions:

- Is the SX-Aurora TSUBASA architecture suitable for running sophisticated data management applications and how does this architecture perform compared to a traditional x86-only system. To do this comparison, we plan to employ a rich set of standard transactional and analytical benchmarks for relational and graph workloads.
- Is the database system able to take advantage of the high memory bandwidth and the huge vector register sizes of the VE and which modifications to ERIS are necessary to better utilize the key features offered by this hardware architecture?
- Do the integrated adaptivity features of ERIS offer any advantage on the SX-Aurora TSUBASA architecture in case of a changing workload, for instance, by adapting the physical data layout in case of a changing data access pattern? Are there different kinds of adaptivity features necessary to fully unleash the performance of the hardware architecture?

5 Conclusion

The hardware landscape is currently changing from homogeneous multi-core systems towards heterogeneous systems with many different computing units, each with their own characteristics. This trend is a great opportunity for database systems to increase the overall performance if the heterogeneous resources can be utilized efficiently. Following that trend, NEC cooperation has recently introduced a novel heterogeneous hardware system called SX-Aurora TSUBASA. The core and unique feature of this novel hardware system is its strong vector engine processor providing world's highest memory bandwidth of 1.2TB/s per vector processor. Thus, we introduced this architecture and properties in this paper. Moreover, we presented first insight of database-specific evaluation results to show the benefits of this hardware system to increase the query performance. We concluded the paper with an outlook on our ongoing research activities in this direction and a short summary.

References

[Ab13] Abadi, Daniel; Boncz, Peter A.; Harizopoulos, Stavros; Idreos, Stratos; Madden, Samuel: The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends in Databases, 5(3):197–280, 2013.

[AMF06] Abadi, Daniel J.; Madden, Samuel; Ferreira, Miguel: Integrating compression and execution in column-oriented database systems. In: SIGMOD. pp. 671–682, 2006.

[BHF09] Binnig, Carsten; Hildenbrand, Stefan; Färber, Franz: Dictionary-based order-preserving string compression for main memory column stores. In: SIGMOD. pp. 283–296, 2009.

[BKM08] Boncz, Peter A.; Kersten, Martin L.; Manegold, Stefan: Breaking the memory wall in MonetDB. Commun. ACM, 51(12):77–85, 2008.

[CK85] Copeland, George P.; Khoshafian, Setrag: A Decomposition Storage Model. In: SIGMOD. pp. 268–279, 1985.

[Da17] Damme, Patrick; Habich, Dirk; Hildebrandt, Juliana; Lehner, Wolfgang: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). In: EDBT. pp. 72–83, 2017.

[Es12] Esmaeilzadeh, Hadi; Blem, Emily R.; Amant, Renée St.; Sankaralingam, Karthikeyan; Burger, Doug: Dark Silicon and the End of Multicore Scaling. IEEE Micro, 32(3):122–134, 2012.

[Hi16] Hildebrandt, Juliana; Habich, Dirk; Damme, Patrick; Lehner, Wolfgang: Compression-Aware In-Memory Query Processing: Vision, System Design and Beyond. In: ADMS. pp. 40–56, 2016.

[HZH14] He, Jiong; Zhang, Shuhao; He, Bingsheng: In-Cache Query Co-Processing on Coupled CPU-GPU Architectures. PVLDB, 8(4):329–340, 2014.

[Id12] Idreos, Stratos; Groffen, Fabian; Nes, Niels; Manegold, Stefan; Mullender, K. Sjoerd; Kersten, Martin L.: MonetDB: Two Decades of Research in Column-oriented Database Architectures. IEEE Data Eng. Bull., 35(1):40–45, 2012.

[KHL17] Karnagel, Tomas; Habich, Dirk; Lehner, Wolfgang: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. PVLDB, 10(7):733–744, 2017.

[KHL18] Kissinger, Thomas; Habich, Dirk; Lehner, Wolfgang: Adaptive Energy-Control for In-Memory Database Systems. In: SIGMOD. pp. 351–364, 2018.

[Ki13] Kissinger, Thomas; Schlegel, Benjamin; Habich, Dirk; Lehner, Wolfgang: QPPT: Query Processing on Prefix Trees. In: CIDR. 2013.

[Ki14] Kissinger, Thomas; Kiefer, Tim; Schlegel, Benjamin; Habich, Dirk; Molka, Daniel; Lehner, Wolfgang: ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload. In: ADMS. pp. 74–85, 2014.

[KML15] Karnagel, Tomas; Müller, René; Lohman, Guy M.: Optimizing GPU-accelerated Group-By and Aggregation. In: ADMS. pp. 13–24, 2015.

[Ko18] Komatsu, Kazuhiko; Momose, Shintaro; Isobe, Yoko; Watanabe, Osamu; Musa, Akihiro; Yokokawa, Mitsuo; Aoyama, Toshikazu; Sato, Masayuki; Kobayashi, Hiroaki: Performance evaluation of a vector supercomputer SX-aurora TSUBASA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. pp. 54:1–54:12, 2018.

[Li16] Li, Feng; Das, Sudipto; Syamala, Manoj; Narasayya, Vivek R.: Accelerating Relational Databases by Leveraging Remote Memory and RDMA. In: SIGMOD. pp. 355–370, 2016.

[Li18] Lisa, Nusrat Jahan; Ungethüm, Annett; Habich, Dirk; Lehner, Wolfgang; Nguyen, Tuan D. A.; Kumar, Akash: Column Scan Acceleration in Hybrid CPU-FPGA Systems. In: International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures, ADMS@VLDB 2018, Rio de Janeiro, Brazil, August 27, 2018. pp. 22–33, 2018.

[LP13] Li, Yinan; Patel, Jignesh M.: BitWeaving: Fast Scans for Main Memory Data Processing. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD '13, ACM, New York, NY, USA, pp. 289–300, 2013.

[LUH18] Lehner, Wolfgang; Ungethüm, Annett; Habich, Dirk: Diversity of Processing Units - An Attempt to Classify the Plethora of Modern Processing Units. Datenbank-Spektrum, 18(1):57–62, 2018.

[Ou17] Oukid, Ismail; Booss, Daniel; Lespinasse, Adrien; Lehner, Wolfgang; Willhalm, Thomas; Gomes, Grégoire: Memory Management Techniques for Large-Scale Persistent-Main-Memory Systems. PVLDB, 10(11):1166–1177, 2017.

[PRR15] Polychroniou, Orestis; Raghavan, Arun; Ross, Kenneth A.: Rethinking SIMD Vectorization for In-Memory Databases. In: SIMD. pp. 1493–1508, 2015.

[St05] Stonebraker, Michael; Abadi, Daniel J.; Batkin, Adam; Chen, Xuedong; Cherniack, Mitch; Ferreira, Miguel; Lau, Edmond; Lin, Amerson; Madden, Samuel; O'Neil, Elizabeth J.; O'Neil, Patrick E.; Rasin, Alex; Tran, Nga; Zdonik, Stanley B.: C-Store: A Column-oriented DBMS. In: VLDB. pp. 553–564, 2005.

[Zu06] Zukowski, Marcin; Héman, Sándor; Nes, Niels; Boncz, Peter A.: Super-Scalar RAM-CPU Cache Compression. In: ICDE. p. 59, 2006.

[ZvdWB12] Zukowski, Marcin; van de Wiel, Mark; Boncz, Peter A.: Vectorwise: A Vectorized Analytical DBMS. In: ICDE. pp. 1349–1350, 2012.

ReProVide: Towards Utilizing Heterogeneous Partially Reconfigurable Architectures for Near-Memory Data Processing

Andreas Becher [1], Achim Herrmann [2], Stefan Wildermann [3], Jürgen Teich [4]

Abstract: Reconfigurable hardware such as Field-programmable Gate Arrays (FPGAs) is widely used for data processing in databases. Most of the related work focuses on accelerating one or a small set of specific operations like sort, join, regular expression matching. A drawback of such approaches is often the assumed static accelerator hardware architecture: Rather than adapting the hardware to fit the query, the query plan has to be adapted to fit the hardware. Moreover, operators or data types that are not supported by the accelerator have to be processed in software. As a remedy, approaches for exploiting the dynamic partial reconfigurability of FPGAs have been proposed that are able to adapt the datapath at runtime. However, on modern FPGAs, this introduces new challenges due to the heterogeneity of the available resources. In addition, not only the execution resources may be heterogeneous but also the memory resources. This work focuses on the architectural aspects of database (co-)processing on heterogeneous FPGA-based PSoC (programmable System-on-Chip) architectures including processors, specialized hardware components, multiple memory types and dynamically partially reconfigurable areas. We present an approach to support such (co-)processing called ReProVide. In particular, we introduce a model to formalize the challenging task of operator placement and buffer allocation onto such heterogeneous hardware and describe the difficulties of finding good placements. Furthermore, a detailed insight into different memory types and their peculiarities is given in order to use the strength of heterogeneous memory architectures. Here, we also highlight the implications of heterogeneous memories for the problem of query placement.

Keywords: FPGA; Shared Memory; Query Acceleration; Near-Memory Processing

Acknowledgement: This project is funded by the DFG priority program SPP2037.

1 Introduction

The usage of FPGAs for database operator acceleration has been of interest in the research community for many years [MT09]. Their unique ability to configure logic resources like flip-flops, look-up tables (LUTs), block RAMs (BRAMs), and digital signal processors (DSPs) to implement complex hardware modules inspired a great body of literature on

[1] Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Erlangen, Germany andreas.becher@fau.de
[2] Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Erlangen, Germany achim.herrmann@fau.de
[3] Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Erlangen, Germany stefan.wildermann@fau.de
[4] Friedrich-Alexander Universität Erlangen-Nürnberg (FAU), Erlangen, Germany juergen.teich@fau.de

doi:10.18420/btw2019-ws-04

accelerated database operators and systems. The ever increasing amount of data produced every day also increased the research efforts to utilize the energy efficiency of FPGA-based implementations. Research projects as well as industrial use cases, like Microsoft's Catapult [Pu14] and Baidu's FPGA-based data analysis [Ou16], have demonstrated various benefits of using FPGA-based co-processors for data processing (see also [Be18]):

- I/O rate processing of pipelined, non-blocking operators.
- Energy efficiency and reduction of power consumption and/or . . .
- . . . speedup through parallel and specialized hardware implementations of OLAP operators.
- Resource efficiency by taking workload from processors and providing query-specific hardware acceleration.

However, the majority of related work focuses on accelerating one or a small set of specific operations like sort, join or regular expression matching. A drawback of such approaches is the assumed static hardware-accelerator architecture: It is not possible to adapt the hardware to the query, rather the query plan has to be adapted to fit the hardware. Moreover, operators or data types that are not supported by the accelerator can't be accelerated. Here, dynamic hardware reconfiguration could provide a viable means to instead adapt the hardware to the query by reconfiguration of acceleration modules at runtime. Consequently, it would be possible to provide query-specific acceleration which is optimized for the respective use case while, at the same time, provide a more efficient utilization of the limited FPGA resources. Yet, the challenge of dynamic reconfiguration is to efficiently adapt the FPGA configuration to a query and to time-multiplex query processing under scarce resources [Zi16]. In particular, ensuring that the reconfiguration pays off is a major challenge, as the FPGA reconfiguration itself can take from a few milliseconds (partial reconfiguration) to seconds (complete reconfiguration) [Be16a].

In this paper, we present a query processing platform based on heterogeneous memory and processing resources. We formalize the problem of placing queries onto such highly heterogeneous platforms and discuss how such placements may be optimized. To exemplify the impact a non-optimal usage of the heterogeneous memory may have, a characterization of the available memory is conducted and an example presented.

2 Related Work

FPGA acceleration of single data processing operations has been excessively investigated in recent years. The important *join* operator has been implemented on FPGAs successful and evaluated in various works. Here, FPGA implementations for various algorithms have been evaluated, e. g., hash join [Ha13], sort-merge join [CO14], as well as hardware-software

co-designs [Be15] and reconfigurable implementations [UIO15]. In addition, as sorting is part of many database operators (e. g. sort-merge join) different algorithms have been implemented on FPGAs over the years, see e. g. [CO14; MTA12; Su13]. Also, FPGA implementations for complex operations exist. E. g., István et al. [ISA16] and Becher et al. [BWT18] propose run-time parametrizable operators for *regular-expression matching* on FPGAs.

Whereas the above cases investigate the acceleration of single operators only, the following approaches consider also the FPGA acceleration of multiple operators within a query such as filter, sort, aggregate, join, and group by as implemented on a FPGA by Baidu [Ou16]. A hardware/software co-design including an FPGA accelerator that consists of a feed-forward pipeline of hardware kernels for selection, projection, and sorting is proposed by Sukhwani et al. [Su15]. Sidler et al. [Si17] extended a CPU-based system by an FPGA implementing complex operators. The FPGA can offer acceleration for the Skyline operator, stochastic gradient descent, and regular expression matching.

However, even if all the approaches above are combined, only a very limited number of queries or datasets still can be automatically processed. The *major* drawback of these approaches is that they can only accelerate queries that contain the specific operations provided by the FPGA design. As soon as a query does not contain these operations or requires the operation on different data types (e.g., FLOAT or DATE instead of INTEGER), no hardware acceleration is supported, leading to a poor utilization of available resources. Particularly, implementing every possible operator (with every possible data type) would lead to huge FPGA designs and unnecessary overheads and is therefore not applicable. Making use of the run-time reconfigurability of FPGAs allows to circumvent the need for huge FPGAs while supporting a wide range of operators. This provides query-specific acceleration optimized for the respective query by loading only the operators needed. Saved FPGA resources can therefore be used to improve the specific operators instead of implementing unused logic circuits. Wang et al. [Wa16] show that even for a single query, it makes sense to provide multiple accelerators and reconfigure between these accelerators while processing a query. They propose a methodology for automatically generating multiple execution plans for a given query. At runtime, the execution plan is chosen that has the lowest execution time according to a cost model. While this work validates the benefit of adapting the hardware to the query by means of reconfiguration, it has the major drawback that each accelerator is tailored to a specific query. Synthesis of an accelerator can take hours. Thus, it is considered infeasible to assemble accelerators for dynamically arriving queries at run-time. In addition, the approach is based on full reconfiguration which implies reconfiguration overheads in the range of seconds.

Contrary, Ziener et al. [Zi16] present a methodology based on partial reconfiguration for *on-the-fly data-path generation* of a query-stream pipeline. Query streams can be accelerated by composing and placing pre-synthesized modules (e. g., aggregation and restriction operators) on the FPGA by means of partial reconfiguration. Becher et al. [Be18] propose a reconfigurable architecture for near-data processing called *Reconfigurable Data Provider*

(ReProVide). ReProVide proposes a layout of reconfigurable hardware resources based on multiple partially reconfigurable areas, into which query-/operator-specific accelerators can be dynamically loaded. Also, this approach makes use of a library of pre-synthesized reconfigurable hardware accelerators. The major difference between [Zi16] and [Be18] is that the former makes use of slot-style partial reconfiguration (which is not supported by standard FPGA tools) whereas the latter exploits island-style partial reconfiguration (which is supported by standard FPGA tools), see [KBT09]. In addition, the latter approach also exploits the coupling and co-processing of queries on FPGA-based PSoCs using the available on-chip processor system. These works introduce the design of reconfigurable accelerator modules for different operations and the design of the partially reconfigurable architecture. In addition, [Zi16] introduces cost models for assessing the performance of query processing based on these accelerator modules. Both works, however, do not provide solutions of the problem of automatically mapping a query onto a respective architecture.

3 Heterogeneous Partially Reconfigurable Architectures for Near-Memory Processing

We consider heterogeneous partially reconfigurable architectures for near-memory processing based on the following concepts. The data is stored in non-volatile memory such as SSDs or in volatile memory like DDR-SDRAM directly attached to the platform. It, thus, does not restrict whatsoever the datastructures used on the storage media and the low-level data management. Data access and modification is therefore only permitted by commands to the platform. A heterogeneous processing architecture consisting of CPUs and specialized hardware is assumed in order to combine the flexibility and ease of software implementations with the energy efficiency and performance of specialized hardware implementations. The *programmable logic* of the FPGA-based SoC platform is divided into a *static* part, which contains all components like hardware controllers and interfaces that stay fixed, and one or multiple *partially reconfigurable regions*, into which reconfigurable hardware accelerators can be loaded. The proposed architecture of a ReProVide platform is an example of such a heterogeneous partially reconfigurable architecture for near-memory processing. The target is to pull the (sparsely stored) data of interest out of one or more high-volume data sources. Only transmitting this information-rich subset of data to the host system has the potential of significantly reducing the dominant factor of power consumption in data-center networks: data transport. This means the reduction of data without an increase of execution time is of utmost importance. The platform, therefore, has to utilize the individual strengths of heterogeneous resources in order to accomplish the needed reduction.

3.1 ReProVide Architecture

The design of a ReProVide platform is based on top of existing FPGA technology like Xilinx Zynq All Programmable SoCs that contain programmable (FPGA) logic, multi-core CPUs,

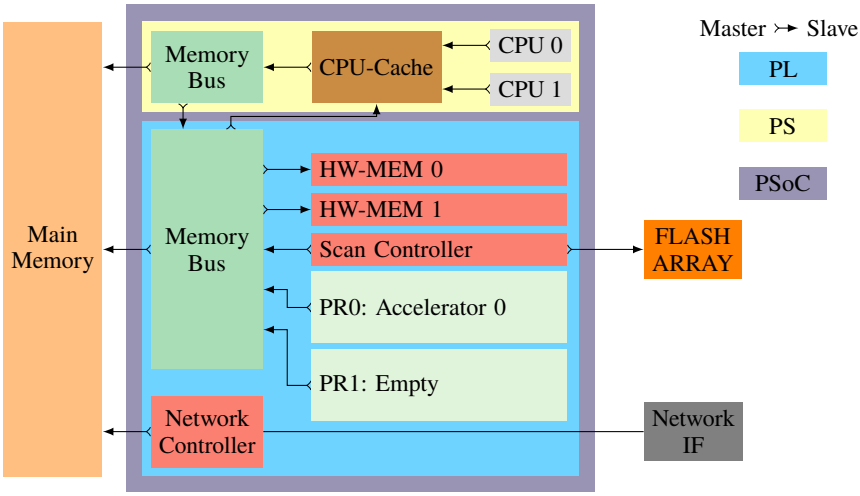


Fig. 1: Architecture of a ReProVide platform. Next to statically implemented IPs such as the *Network Controller*, *Scan Controller*, and heterogeneous (on- and off-chip volatile and non-volatile) memory resources, also partially reconfigurable areas are shown (*PR0, PR1*). These areas are reconfigured during runtime, loading presynthesized hardware modules. Additionally, a Processing System (PS) containing a dual core ARM-based processor is contained on the SoC.

and peripherals. This FPGA-based SoC makes use of hardware reconfiguration to adapt datapaths and accelerators for being able to process different online analytical processing (OLAP) and data mining operators on data from its attached heterogeneous volatile and non-volatile data sources.

Fig. 1 depicts the architecture of a proposed ReProVide platform containing a tightly-coupled processor system (PS) and programmable logic (PL). While the table management is executed on one core of the processing system, handling of the data is mostly dealt with within the programmable logic. To accelerate typical OLAP query operations, multiple partially reconfigurable regions (PRs) within the programmable logic allow offloading of operators. Requests are received via a high-speed network interface and forwarded to the management software for further processing. We will go into details in Section 3.2. To relief the processing system from the task of result transmission, a specialized *Network Controller* implemented in the static system allows sending the resulting data to the requesting host with minimum intervention from the management software. It utilizes a circuit for data reordering which we will introduce in Section 3.1.1.

A host may request data using a specified schema (row-store or column-store layout) while the management might use another schema to store the data on the attached memories. In addition, hardware accelerators are either specialized for a specific schema or are able to cope with different schemes. The first option would, however, increase the development time of the accelerator library and reduce the chances an already configured accelerator

can be reused. For the second option, to evade using additional logic in the accelerator itself or executing software to detangle the various schemes, we introduce a dedicated and parameterizable hardware component called *Scan Controller* to do the job of data loading and schema-on-fly transformations. Therefore, this controller also has direct access to the attached FLASH storage.

3.1.1 Scan Controller

The main task of the *Scan Controller* is to translate the schema which is used to store the data in the available memories and strip not needed attributes. A programmable data reordering engine called ReOrder unit [Be16b] together with scatter-gather DMA-engines are utilized to achieve this task. When combined, the *Scan Controller* can be programmed to gather data from multiple locations (particularly the case in column-oriented table store) to provide compacted tuples for further processing. In the case of row-oriented stored tables, not needed attributes are striped off to relief the interconnect from transferring unnecessary information and various memories from reserving space for this.

This functionality allows to define a single and common schema on which all operators work and may therefore disconnect the operator implementation (e.g., hardware accelerator) from the storage format.

Additional assistance for hardware accelerators is given by the insertion of so-called *placeholders* (intermediate/temporary attributes) into the tuple stream. These *placeholders* can be used to store intermediate results within the tuple, e.g., from arithmetic operations like *additions*, without the need to dynamically change the size of the tuple or the overhead of an additional result stream.

Fig. 2 depicts a simple query from (a) a row-based table and (b) a column-based table.

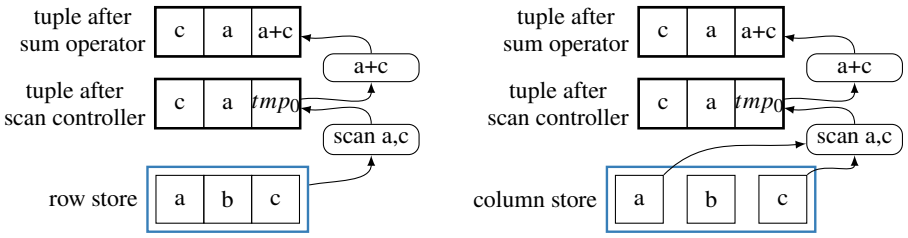


Fig. 2: A tuple (a, b, c) being processed according to the query `SELECT c,a,(a+c);` using a row-oriented storage (a) and a column-oriented storage (b). First, the scan controller produces a new tuple with placeholders and the requested order of attributes. In the next step, the *sum* function is applied to this tuple and the result is written to the previously created placeholder field (tmp_0).

Note that the *sum* operator sees the same input tuple in both cases and can therefore use the same accelerator. As can be seen, the *Scan Controller* can perform simple table transformations transparently to abstract from the storage layout of the tables. Not only does this simplify the operator implementations and increase the overall execution speed as less partial reconfigurations are necessary, it enables optimizations going beyond classical row-/column-oriented storages, e. g. partitioning a table onto different memory types or by related attributes.

A subset of this transformation capabilities is also implemented in the *Network Controller* to remove no longer needed attributes and *placeholders* before transmitting results to the requesting host.

3.2 Query Management

Once a query is received from a requesting host, a local query management is required to actually execute the query on the ReProVide platform. The query manager will be executed on the CPU. It involves three basic steps as illustrated in Figure 3.

The first step is *query placement* of the *query execution plan* (QEP) onto the available resources on the ReProVide platform. This is a *hardware/software partitioning* problem according to [Te12], and consists of the following three steps: (a) *allocation* of resources, (b) *binding* of tasks onto resources, and (c) *scheduling* of tasks on shared resources. For query placement on reconfigurable architectures, this hardware/software partitioning problem has to be adapted as detailed in Section 4.

The task of *query compilation* is then to configure the platform and particularly the selected accelerators with the runtime parameters according to the operator parameters in the query execution plan. This step is operator-specific and beyond the scope of this paper. It is e. g. illustrated in [BWT18] for accelerators for regular expression matching.

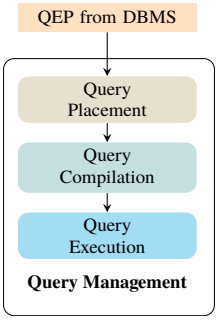


Fig. 3: Overview of runtime management. The lines indicate the flow of how a QEP is processed when obtaining it from a requesting host.

Finally, *query execution* is responsible for orchestrating the query processing according to the schedule generated in the first step. This particularly involves scheduling the reconfiguration of accelerators on shared partial regions, the memory allocation, as well as exception handling which particularly occurs for under- or overflowing buffers.

4 Query Placement Problem

A query execution plan (QEP) describes the order in which operations have to be applied on the data to execute a given query. In order to utilize the full potential, a given query execution plan has to be mapped onto the available resources of a ReProVide platform. Such a mapping can be optimized for query execution time, resource consumption, and energy efficiency or a combination of these objectives. In this subsection, we introduce the formal model of the placement problem for QEPs onto a ReProvide platform.

4.1 Generating the Query-Specific Architecture Configuration

One part of the query placement problem is to decide how to configure the reconfigurable areas to process a given query. Here, the *QEP* is represented as a directed graph $G_Q = (O, D)$ containing the set of operators O (vertices) and their dependencies on other operators $D \subseteq O \times O$ (directed edges), see Fig. 4(a), left for an example.

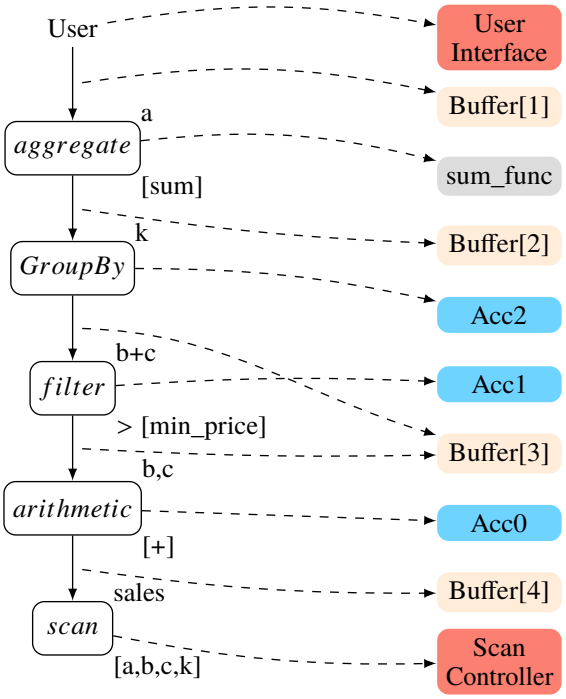
Problem graph. We assume a set H of pre-synthesized hardware modules as well as a set S of software functions for accelerating operators on a ReProVide platform is given. The set of available acceleration functions being available in an *accelerator library* are denoted by $L = H \cup S$. Each function $l \in L$ covers the acceleration of one or multiple operators. Now, for generating the *query-specific configuration*, first a set of accelerators A has to be allocated (instantiated) from this library. Then, each operator node $o \in O$ of the query plan has to be mapped onto an accelerator that supports the respective operator. This operator mapping is given by

$$\beta_O : O \to A. \tag{1}$$

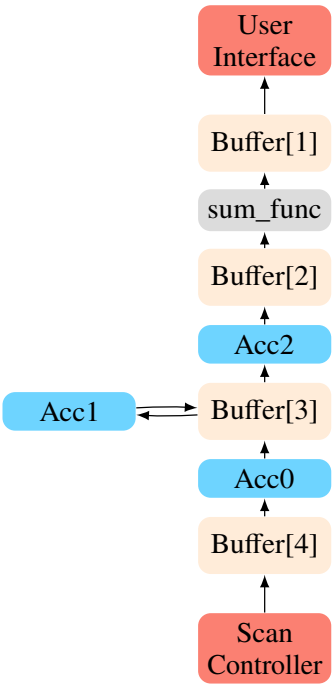
Furthermore, a set of buffers B has to be allocated for storing the data that is exchanged between the accelerators. Each data dependency $d \in D$ of the QEP has to be assigned to one allocated buffer:

$$\beta_D : D \to B. \tag{2}$$

Fig. 4(a) illustrates a mapping of operators onto accelerators and of data dependencies onto buffers. Based on the set $T = A \cup B$ of allocated accelerators and buffers as well as



(a) Example QEP and a query-specific allocation of accelerators and buffers. Operators are mapped onto accelerators, and data dependencies are mapped onto buffers according to the dashed edges. The width of each buffer (i. e. tuple size) is given in brackets and omitted if *zero*.



(b) Problem graph generated from the QEP resource allocation (a). Again, the width of each buffer is given in brackets and omitted if *zero*.

Fig. 4: Problem graph generation from QEP.

the mapping of operators and data dependencies, the *query-specific configuration* can be represented as a query-specific *problem graph* $G_P = (T, F)$, where edges F connect the buffers with the accelerators accessing them. Buffers b connected to an accelerator a by an edge $(b, a) \in F$ or $(a, b) \in F$ are called *input buffers* or *output buffers* of a, respectively. Fig. 4(b) represents the problem graph of the example. Note that the flow of data as shown in a problem graph is opposite to the direction of edges of the QEP.

Buffer allocation. The proper *dimensioning of buffer capacities* is a major optimization control knob. As we will discuss below, it may have a huge influence on achievable performance numbers like throughput. However, on the other hand, it has to adhere to several constraints. The Scan Controller (described in Section 3.1.1) supports to load attributes from various data sources with different layouts. Its purpose is to load all attributes required by the subsequent operators. Moreover, the output buffer of each accelerator has to provide

space for storing all those attributes that are still required by subsequently executed operators. This means that the accelerator (a) copies the respective attributes from its input buffer(s) to its output buffer together with (b) writing the results calculated by the accelerator itself.

Fig. 4(a) illustrates this for an example. The *scan* operator has to load attributes $[a, b, c, k]$. Operator *arithmetic* has to write attributes $[a, b + c, k]$. The filter has to write attributes $[a, k]$, and so on.

Consequently, the *width* of a buffer is fixed for one problem graph and given by the size of one tuple of the respective attributes. However, the *depth* of a buffer $b \in B$ (i.e., the amount of tuples that can be stored in the buffer) is an optimization control knob, which we denote by $n_b \in \mathbb{N}_{\geq 0}$.

Let $width(b)$ be the statistically[5] derivable size of one tuple in buffer b.

Then, the $capacity$ of each allocated buffer $b \in B$ is denoted by

$$capacity(b, n_b) = width(b) \cdot n_b. \tag{3}$$

In Section 5, we empirically study the influence of this optimization variable on throughput. Several operators support stream processing of tuples (e.g., most arithmetics and filters), and thus support arbitrary $n_b \geq 1$. However, so-called *blocking* operators (e.g., *sort*, *unique* and *join*) depend on several or even all tuples from the preceding operators. We therefore include $n_{min} : O \rightarrow \mathbb{N}_{\geq 0}$ which gives the minimum amount of tuples the operator needs to access for calculation. Note that this may rise to the size of a whole table if operators such as *joins* need random access on the intermediate result. Therefore, the minimum buffer depth depends on the operators executed on the accelerators accessing the buffer, and is obtained as:

$$\forall o \in O, b \in B : (b, \beta_O(o)) \in F : n_b \geq n_{min}(o) \tag{4}$$

The problem graph generation may already apply domain knowledge to create optimized buffer allocations. One such optimization is the instantiation of multiple buffers to support streaming execution of QEP operations. Another is to reduce the over-provisioning of buffers: Without accurate statistics, some buffer might be highly over-dimensioned if designed to hold the whole table to guarantee no information is lost.

4.2 Determining a Configuration of the Reconfigurable Architecture

From the problem graph specification, we still have to determine a *query-specific configuration* (QSC) of the platform. The actual reconfigurable architecture (as presented in Section 3)

[5] The size can be determined accurately if a tuple consists of fixed length columns only. If variable length columns are present, a meaningful average is statistically obtained. Query execution has to handle occurring exceptions when buffers are too small to hold a single tuple due to the length of the variable length fields.

provides static resources that have to be programmed (or re-configured) according to such a configuration. Notably, this is also a mapping problem which we formalize in the following.

Architecture Graph. In order to embed a problem graph generated from a QEP onto a concrete ReProVide platform, also a graph representation can be used. A heterogeneous architecture is described by a directed architecture graph $G_A = (R, L)$. This architecture graph models the available resources, i. e., CPU cores, memories, co-processors, and partially reconfigurable areas, as well as the connections between these resources. Fig. 5 (right) visualizes an architecture graph of the ReProVide platform shown in Fig. 1. The vertices represent these resources $R = R_P \cup R_M \cup R_C$, being processing R_P, memory R_M, and communication resources R_C. Memory resources are characterized by a maximum storage capacity $capacity(m), \forall m \in R_M$.

Directed links $l \in L$ connect resources which can communicate with each other: $L \subset R \times R$. Note that the direction of an edge describes which node is the *initiator* (or *Master*) of a data transfer (outgoing edges) and therefore does not describe the direction of the flow of data itself. This is consistent with the *Master Slave principle* that is also illustrated by Fig. 1, and allows to express separated memory domains (e.g., the network controller can only access the main memory in Fig. 5). Let $C_{G_A}(v)$ with $v \in R$ define the set of all resources connected to v in G_A. A pair of resources $(a, b) : a, b \in R$ is connected if a directed path from a to b or b to a exists. For simplicity, we assume all links to allow for bidirectional (duplex) data transfers.

Mapping the Problem Graph. The problem of determining a query-specific configuration (QSC) on the reconfigurable target platform can thus be formalized as mapping the problem graph $G_P = (A \cup B, F)$ onto the architecture graph. This involves *binding* each accelerator $a \in A$ of the problem graph to one processing resource $r \in R_P$ of the architecture graph, given by

$$\beta_A : A \rightarrow R_P. \tag{5}$$

Here, obviously, hardware accelerators can only be bound to partially reconfigurable regions, and software accelerators to software processing resources. Furthermore, each buffer $b \in B$ has to be bound onto one memory resource $r \in R_M$:

$$\beta_B : B \rightarrow R_M. \tag{6}$$

The bindings have to adhere to the following mapping constraints.

Routing contraints: Each accelerator a can access the memories containing its input and output buffers from the processing resource $\beta_A(a)$ on which it is executed, or formally with

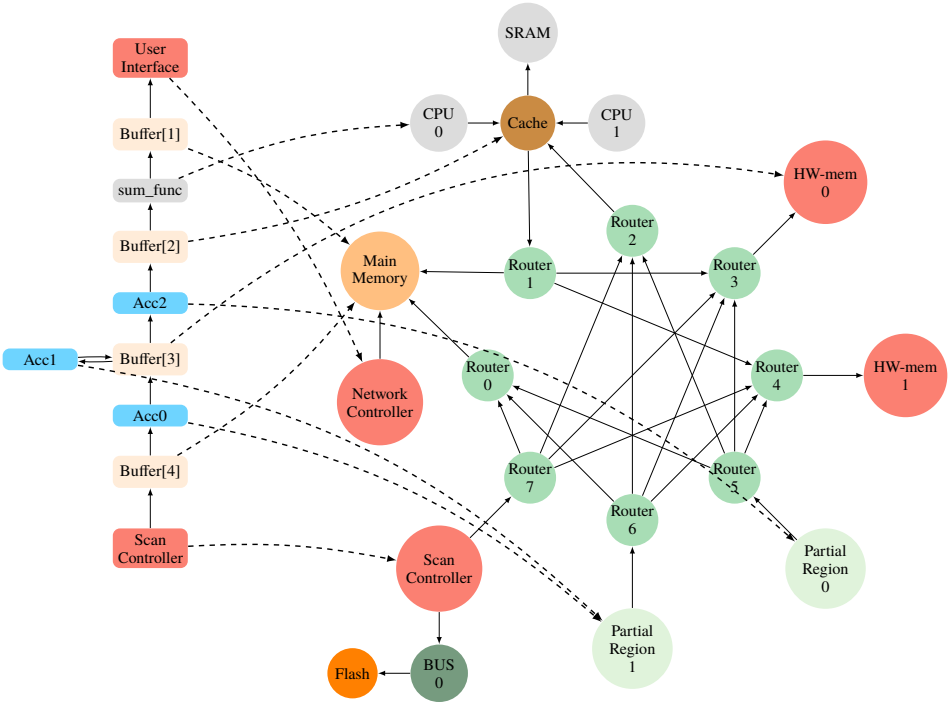


Fig. 5: Problem graph (left) and its binding to the architecture graph of a heterogeneous architecture such as the ReProVide platform depicted in Fig. 1. The communication resource *BUS0* models the half duplex connection between the *Flash*-Memory and the *Scan Controller*.

C_{G_A} representing the set of reachable resources:

$$\forall a \in A, b \in B, (b,a), (a,b) \in F : \beta_B(b) \in C_{G_A}(\beta_A(a)) \tag{7}$$

Memory contraints: We assume that the buffers are statically allocated and then used throughout the execution of the query. As discussed before, the size of buffers depends on decision variables which are subject to the constraint in Eq. (4). When binding buffers to memory resources, it has to be ensured that the memory resources provide sufficient capacities to hold the assigned buffers, i.e.,

$$\forall r \in R_M : \sum_{\substack{b \in B: \\ \beta_B(b)=r}} capacity(b, n_b) \leq capacity(r) \tag{8}$$

Fig. 5 illustrates a valid binding of the exemplary G_P onto the architecture graph G_A of a ReProVide. As illustrated for $Acc0$ and $Acc1$, accelerators may be bound to the

same processing resource, in this case partially reconfigurable region PR1. In such cases, it is necessary to provide *scheduling strategies* to resolve possible conflicts. In case of data-dependent accelerators as in this example ($Acc1$ requires the results from $Acc0$), the scheduling priority is given according to the dependencies. In case of accelerators without dependencies, i.e., accelerators of two different QEP branches before a *join* operation, all accelerators on the branch that is prioritized by the join operator will be scheduled before the accelerators on the other branch.

While we used a single query as a driving example, please note that multiple concurrent queries can be placed. The same constraints apply for the allocation and binding. The scheduler, however, might be adapted to apply prioritization if resources are shared.

4.3 Optimal Query Placement

So far, the steps and constraints for determining a *feasible* query-specific configuration (QSC) were discussed. However, in general, an *optimal placement* is desired which optimizes performance numbers like throughput or latency. Multiple factors influence the performance numbers such as the set of allocated accelerators, the mapping of operators onto these accelerators, and the mapping of accelerators onto the available resources. The big advantage of hardware accelerators is their determinism. Particularly, the performance of streaming operations can be predicted. So, quite accurate cost models exist for such accelerators. Particularly, Ziener et al. [Zi16] propose cost models that allow to estimate latency and throughput of a pipeline of such accelerators. The accelerators in the library of ReProVide are pre-synthesized. This means that optimizing a single accelerator for latency and throughput are offline tasks and thus not part of the query placement problem.

However, reconfiguration between accelerators during processing of a query introduces additional offsets which are not considered in [Zi16]. ReProVide make use of island-style reconfiguration, which means that, when a module is loaded into a reconfigurable area, the complete area is reconfigured. Therefore, the reconfiguration time depends not on the accelerator but on the size of the reconfigurable area. For reasonably sized areas, reconfiguration time can stay within few milliseconds, see [Be16a]. However, determining the size of the reconfigurable areas is an offline task and thus not part of the query placement problem.

For ReProVide platforms, we assume a self-triggered execution, i. e. each accelerator starts working when it receives a ready signal from its succeeding accelerators and the directly preceding accelerator signals that it has filled the input buffer. When an accelerator starts execution, it consumes the tuples from its input buffer and produces the output in its output buffer. Subsequently, it sends a ready signal to its preceding accelerator and triggers the execution of the succeeding accelerator. With this execution scheme, the dimensioning of the allocated buffers and their mapping onto the available memory resources can significantly influence performance numbers as latency and throughput. Particularly, a buffer depth of

$n_b = 1$ does not necessarily lead to the best pipelined schedule, even when all accelerators should have the same latency. In order to motivate the huge impact, we take a look into the characteristics of different memory types in the next subsection.

The optimal query placement depends on the allocation of accelerators and buffers, as well as their binding onto the architecture. The problem formalization of this paper is an extension of the system-level synthesis problem, which is proven to be NP-hard [BTT98][6]. Thus, finding an, at least, optimized placement is a complex task which can only be efficiently tackled by heuristics. Particularly, for query processing, an optimized placement should be available within a few milliseconds. While processing, additional queries may arrive and a new placement has to be found covering all running queries. This means the placement might change every time a new query arrives or completes, making it even more important to be adapted quickly. Whereas the main contribution of this paper is on understanding and formalizing this problem, the investigation of efficient heuristics is future work.

5 Heterogeneous memory system

ReProVide platforms allow to use three different types of memories: (I) local memory like FPGA BlockRAM or CPU caches, (II) DDR-SDRAM and (III) Flash memory. In order to optimize the mapping of the buffers of a problem graph to physical memory, one must know how these different memories behave in terms of throughput, latency, capacity and costs. This section classifies the memories according to these properties. The values used in this section are mostly taken from literature and therefore inaccurate, but this is still sufficient for the purpose of characterization. Additionally, the characteristic values for an example of a concrete ReProVide platform will be exemplified.

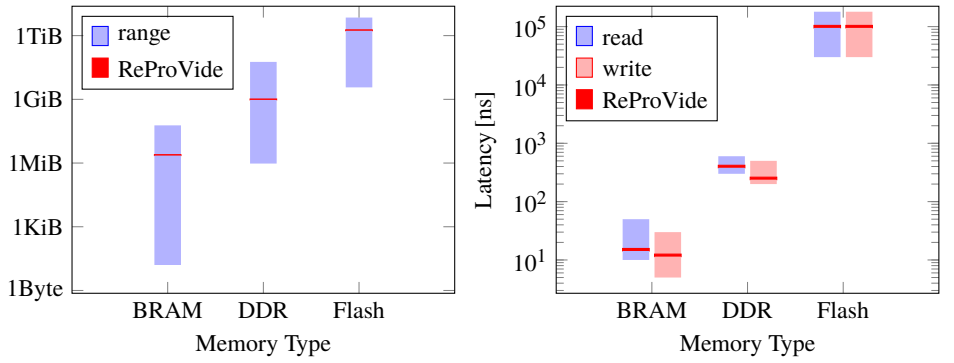
5.1 Memory Characterization

Capacity As ReProVide's purpose is the processing and filtering of big amounts of data, it is necessary to have enough capacity to store the intermediate data. The capacity depends on the particular hardware in use. Typical ranges for the different memories are depicted in Fig. 6(a). Local memory usually has capacities from a few bytes up to a few megabytes, DDR-SDRAM can have a few gigabytes and Flash memory goes up to many terabytes.

Cost Of course, it would be possible to have, for example, a terabyte of DDR-SDRAM, but compared to flash memory, this would be much more expensive in terms of both cost and energy. According to Xilinx Vivado, one Byte of BRAM has a power consumption in the scale of µW. DDR-SDRAM is more efficient with nW per Byte [LC03]. Flash memory

[6] This means that also the query placement problem is NP-hard, as the system-level synthesis problem can trivially be reduced onto it.

has the lowest power consumption per Byte: as it is non-volatile, just holding data requires no energy at all. When in use, its power consumption is in the magnitude of pW/B [LMP09]. Also, the price of 0.2 $/GB for flash memory is much lower than for DDR-SDRAM, which costs about 6 $/GB [Ha17].



(a) Capacity

(b) Latency

Fig. 6: Typical capacity and latency of different memory types and the Xilinx zc706-based ReProVide prototype.

Latency But capacity often comes at the cost of latency. Here, latency is defined as the time it takes from request on until the first byte has been processed, i. e. read or written. It is desirable to keep the latency low as this will improve the data processing performance, especially for a small amount of data. Fig. 6(b) depicts typical latencies for different memory types in nanoseconds: The small BRAMs are usually placed tightly to the processing unit, leading to very low latency down to one cycle. The bigger DDR-SDRAMs require controllers to manage memory banks, increasing the latency to a few dozen of clock cycles. Flash memories often additionally use internal buffering and a serial interface. Both may increase the latency further into mid microseconds range [Gr09].

Throughput Throughput is the main characteristic when it comes to transferring big blocks of data. The throughput of a memory depends on different factors. The most obvious ones are bus-width and frequency. These factors impact the throughput linearly, means when doubling one of them, also the throughput doubles. They are both determined by the hardware implementation, so the query placement has no influence on them. A factor which can be influenced by query placement is the blocksize (i. e. the *capacity acc. to Eq.* (3)), and thus, how many data is transferred within one request. When transferring only small blocks, latency is the biggest component of the transfer time. With the blocksize high enough, the latency becomes negligible as it can be compensated through pipelining, and the overall throughput rises. Fig. 7 depicts the normalized optimum read and write throughput of the memory types, depending on the transferred blocksize. As it has the lowest latency, BRAM

generally has the highest throughput independent of the blocksize. Flash memory on the other side requires huge blocks to reach an acceptable throughput. Note that also the access pattern has a high impact on the throughput: sequential access is faster than random access.

For example, a ReProVide platform based on a Xilinx zc706 can access the DDR-SDRAM either over the Zynq's high performance ports (HP) or over the Accelerator Coherency Port (ACP). Sadri [Sa13] found out that the ACP has equal performance of up to 1,7 GB/s as the HP port, as long as the transfer size does not exceed the cache size of 512 KByte. Afterwards, the throughput drops to 600 MB/s due to invalidating the caches. The ACP performance is also effected by background applications running on the CPU, since they also use the caches. So are the high performance ports effected by each other, since they have to share the internal interconnect.

The examplary ReProVide platform uses a raid of SATA-3 SSDs as flash memory. The SATA-3 protocol has a theoretical throughput of 600 MB/s. However, the access pattern has huge impact on the actual throughput. Sequential-like throughput is only achieved with blocksizes greater than 32 KByte. Furthermore, there are effects like garbage collection and defragmentation running inside the SSD firmware to consider which may reduce the achievable throughput dramatically [Se18]. However, these effects are highly specific to the use case and the manufacturer.

In summary, there is a tradeoff between fast but small and expensive memories and slow but big and inexpensive memories. If one needs fast random access with small blocksizes, BRAMs offer the best performance if the amount of available storage is sufficient. DDR-SDRAM can be used if a higher capacity is needed or in case the blocksizes to access the memory are bigger. Lastly, flash memory is the best choice for storing huge amounts of data which is mostly read using sequential access or huge datablocks. Therefore, the buffer

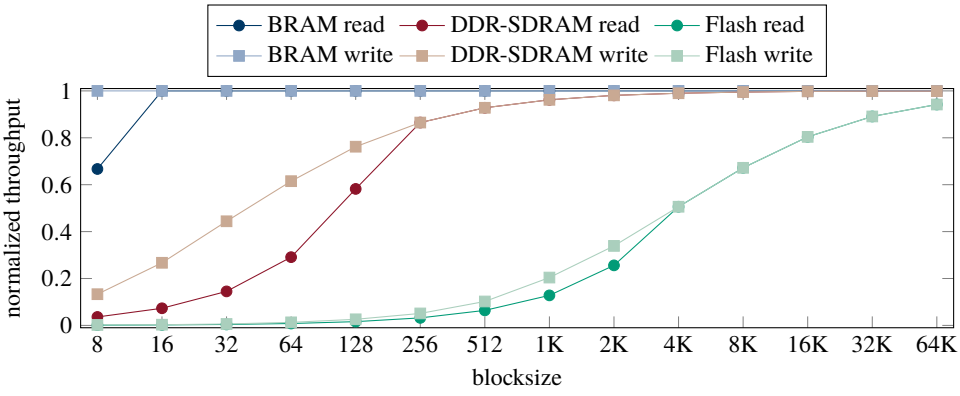
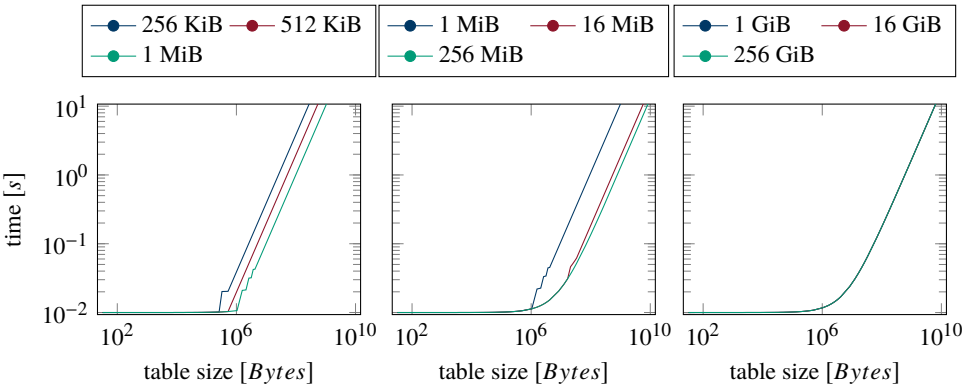


Fig. 7: Random access throughput of different memory types in relation to the size of each transfer (blocksize). The throughput is normalized to the maximum sequential throughput of each memory type.



(a) BRAM (b) DDR-SDRAM (c) Flash

Fig. 8: Execution time for two accelerator scheduled on the same partial reconfigurable area to process tables of different sizes. A buffer is allocated for exchanging data between both accelerators. Reconfiguration between accelerators is performed as soon as all tuples are produced in/consumed from the buffer. Per memory type, the execution time for different sizes allocated for this buffer is depicted.

allocation as part of query placement as presented in Section 4.1 has direct impact on the execution time of the query to process.

5.2 Exemplary Impact of Buffer Allocation and Binding on Query Execution Time

In order to give an example of the impact of different buffer allocations, we consider two accelerators mapped onto one partial region and exchanging tuples via a shared buffer, comparable to operators $Acc0$ and $Acc1$ in Fig. 5[7]. Each time accelerator $Acc0$ has filled its output buffer, a partial reconfiguration is triggered so that accelerator $Acc1$ consumes the data.

Fig. 8 depicts the execution time (y-axis) of these two accelerators processing tables of different sizes (x-axis). The execution time is depicted for all three memory types and for different sizes of the buffer used for exchanging data between the accelerators. We have chosen buffer sizes which are reasonable for the memory type with respect to its specific capacity ranges (see Fig. 6(a)). As one can see, the buffer size has a direct impact on the execution time. Increasing buffer size reduces the overall execution time as accelerators can process more tuples with less reconfiguration overhead.

Furthermore, the binding of buffers to the memory type has a direct impact on execution time. BRAM is the fastest if the table size is low (Fig. 8(a)). However, the throughput

[7] However, we ignore the scheduling of preceding and succeeding accelerators in the following example.

advantage is lost when table sizes grow: As the capacity of BRAM is limited, only small buffer sizes can be chosen (kilobytes to ~one megabyte). This means that for greater table sizes the reconfiguration has to be performed more frequently, and thus, reconfiguration overhead becomes dominant. Due to their higher capacities, the DDR-SDRAM and the Flash memories support bigger buffer sizes and are thus able to decrease the number of required reconfigurations. Particularly, Flash memories allow to choose buffer sizes in the range of gigabytes and thus to reduce the reconfiguration overhead even further. However, DDR-SDRAM has a higher throughput than Flash. Therefore, when choosing a sufficiently high buffer size (over 16 MiB in our example), the DDR-SDRAM is able to compensate the additional reconfigurations. This leads to DDR-SDRAM providing the lowest overall execution time (Fig. 8(b)), even compared to Flash (Fig. 8(c)). For a given table size, a simple heuristic can be derived to obtain a good buffer size.

Please note that, in this example, we assumed that the memories are exclusively accessed by the accelerators such that no congestion occurs. However, during operation of a ReProVide platform, concurrently executed accelerators of the same or even other parallel queries may access the interconnect and memories. This would reduce the actually achievable throughput. As such, also the congestion on interconnect and memories should be considered when solving the query placement optimization problem.

6 Conclusion and Future Work

While heterogeneity promises benefits in terms of reduced execution time, energy efficiency and costs for database processing, it is a burden too. Utilization of such heterogeneous architectures to gain benefits can be challenging. We presented a model to describe heterogeneous architectures such as a ReProVide platform and formalized the task of automatically generating a configuration from a given query execution plan while satisfying a number of physical constraints on computing, communication and memory resources. To allow for the optimization of this process, various factors have to be considered. Therefore, the available memory resources have been characterized and the effects of buffer allocation have been illustrated. With the presented model and characterization, we will investigate algorithms to find good solutions for the placement problem in the future.

References

[Be15] Becher, A.; Ziener, D.; Meyer-Wegener, K.; Teich, J.: A co-design approach for accelerated SQL query processing via FPGA-based data filtering. In: FPT. Pp. 192–195, Dec. 2015.

[Be16a] Becher, A.; Pirkl, J.; Herrmann, A.; Teich, J.; Wildermann, S.: Hybrid energy-aware reconfiguration management on Xilinx Zynq SoCs. In: ReConFig. Pp. 1–7, 2016.

[Be16b] Becher, A.; Wildermann, S.; Mühlenthaler, M.; Teich, J.: ReOrder: Runtime datapath generation for high-throughput multi-stream processing. In: ReConFig. Pp. 1–8, 2016.

[Be18] Becher, A.; B.G., L.; Broneske, D.; Drewes, T.; Gurumurthy, B.; Meyer-Wegener, K.; Pionteck, T.; Saake, G.; Teich, J.; Wildermann, S.: Integration of FPGAs in Database Management Systems: Challenges and Opportunities. Datenbank-Spektrum 18/3, pp. 145–156, Nov. 2018.

[BTT98] Blickle, T.; Teich, J.; Thiele, L.: System-Level Synthesis Using Evolutionary Algorithms. Design Automation for Embedded Systems 3/1, pp. 23–58, Jan. 1998, ISSN: 1572-8080.

[BWT18] Becher, A.; Wildermann, S.; Teich, J.: Optimistic regular expression matching on FPGAs for near-data processing. In: DaMoN. 4:1–4:3, June 2018.

[CO14] Casper, J.; Olukotun, K.: Hardware acceleration of database operations. In: FPGA. Pp. 151–160, 2014.

[Gr09] Grupp, L. M.; Caulfield, A. M.; Coburn, J.; Swanson, S.; Yaakobi, E.; Siegel, P. H.; Wolf, J. K.: Characterizing flash memory: Anomalies, observations, and applications. In: MICRO. Pp. 24–33, Dec. 2009.

[Ha13] Halstead, R. J.; Sukhwani, B.; Min, H.; Thoennes, M.; Dube, P.; Asaad, S. W.; Iyer, B.: Accelerating Join Operation for Relational Databases with FPGAs. In: FCCM. Pp. 17–20, 2013.

[Ha17] Havard: Historical Cost of Computer Memory and Storage, Accessed: November 20, 2018, Dec. 2017, URL: https://hblok.net/blog/storage/.

[ISA16] István, Z.; Sidler, D.; Alonso, G.: Runtime Parameterizable Regular Expression Operators for Databases. In: FCCM. Pp. 204–211, 2016.

[KBT09] Koch, D.; Beckhoff, C.; Teich, J.: Minimizing Internal Fragmentation by Fine-Grained Two-Dimensional Module Placement for Runtime Reconfiguralble Systems. In: FCCM. Pp. 251–254, Apr. 2009.

[LC03] Lee, H. G.; Chang, N.: Energy-aware Memory Allocation in Heterogeneous Non-volatile Memory Systems. In: ISLPED. Pp. 420–423, 2003, ISBN: 1-58113-682-X, URL: http://doi.acm.org/10.1145/871506.871609.

[LMP09] Lee, S.-W.; Moon, B.; Park, C.: Advances in Flash Memory SSD Technology for Enterprise Database Applications. In: SIGMOD. Pp. 863–870, 2009, ISBN: 978-1-60558-551-2, URL: http://doi.acm.org/10.1145/1559845.1559937.

[MT09] Müller, R.; Teubner, J.: FPGA: What's in it for a database? In: SIGMOD. Pp. 999–1004, 2009.

[MTA12] Müller, R.; Teubner, J.; Alonso, G.: Sorting networks on FPGAs. VLDB J. 21/1, pp. 1–23, 2012.

[Ou16] Ouyang, J.; Qi, W.; Wang, Y.; YichenTu; Wang, J.; Jia, B.: SDA: Software-Defined Accelerator for general-purpose big data analysis system. In: HCS. Pp. 1–23, Aug. 2016.

[Pu14] Putnam, A.; Caulfield, A.; Chung, E.; Chiou, D.; Constantinides, K.; Demme, J.; Esmaeilzadeh, H.; Fowers, J.; Gray, J.; Haselman, M.; Hauck, S.; Heil, S.; Hormati, A.; Kim, J.-Y.; Lanka, S.; Peterson, E.; Smith, A.; Thong, J.; Xiao, P. Y.; Burger, D.; Larus, J.; Gopal, G. P.; Pope, S.: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In: ISCA. Pp. 13–24, 2014.

[Sa13] Sadri, M.; Weis, C.; Wehn, N.; Benini, L.: Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ. In: FPGAworld. 5:1–5:8, 2013, ISBN: 978-1-4503-2496-0, URL: http://doi.acm.org/10.1145/2513683.2513688.

[Se18] Seagate Technology, L.: Lies, Damn Lies And SSD Benchmark Test Result, Accessed: November 20, 2018, 2018, URL: https://www.seagate.com/de/de/tech-insights/lies-damn-lies-and-ssd-benchmark-master-ti/.

[Si17] Sidler, D.; István, Z.; Owaida, M.; Alonso, G.: Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures. In: SIGMOD. Pp. 403–415, 2017.

[Su13] Sukhwani, B.; Thoennes, M.; Min, H.; Dube, P.; Brezzo, B.; Asaad, S. W.; Dillenberger, D.: Large Payload Streaming Database Sort and Projection on FPGAs. In: SBAC-PAD. Pp. 25–32, 2013.

[Su15] Sukhwani, B.; Thoennes, M.; Min, H.; Dube, P.; Brezzo, B.; Asaad, S. W.; Dillenberger, D.: A Hardware/Software Approach for Database Query Acceleration with FPGAs. International Journal of Parallel Programming 43/6, pp. 1129–1159, 2015.

[Te12] Teich, J.: Hardware/Software Codesign: The Past, the Present, and Predicting the Future. Proceedings of the IEEE 100/Special Centennial Issue, pp. 1411–1430, May 2012, ISSN: 0018-9219.

[UIO15] Ueda, T.; Ito, M.; Ohara, M.: A dynamically reconfigurable equi-joiner on FPGA, IBM Technical Report RT0969, 2015.

[Wa16] Wang, Z.; Paul, J.; Cheah, H. Y.; He, B.; Zhang, W.: Relational query processing on OpenCL-based FPGAs. In: FPL. Pp. 1–10, 2016.

[Zi16] Ziener, D.; Bauer, F.; Becher, A.; Dennl, C.; Meyer-Wegener, K.; Schürfeld, U.; Teich, J.; Vogt, J.; Weber, H.: FPGA-Based Dynamically Reconfigurable SQL Query Processing. TRETS 9/4, 25:1–25:24, 2016.

Query Planning for Transactional Stream Processing on Heterogeneous Hardware: Opportunities and Limitations

– Novel Ideas & Experience Reports –

Philipp Götze[1] Constantin Pohl[1] Kai-Uwe Sattler[1]

Abstract: In a heterogeneous hardware landscape consisting of various processing units and memory types, it is crucial to decide which device should be used when running a query. There is already a lot of research done for placement decisions on CPUs, coprocessors, GPUs, or FPGAs. However, those decisions can be further extended for the various types of memory within the same layer of the memory hierarchy. For storage, a division between SSDs, HDDs or even NVM is possible, whereas for main memory types like DDR4 and HBM exist. In this paper, we focus on query planning for the transactional stream processing model. We give an overview of several techniques and necessary parameters when optimizing a stateful query for various memory types, outlined with chosen experimental measurements to support our claims.

Keywords: Stream Processing, Transactions, Cost Model, Xeon Phi, NVM, Non-Volatile Memory

1 Introduction

Technological advance has lead to a high degree of specialization in terms of hardware. Instead of a single processing device being capable of dealing with any requirements and applications, specialized variants improve performance more than a general approach would. GPUs and many-core CPUs come with an intense computational power through parallelism, FPGAs allow reconfigurations of functionality, NVM technology could provide persistence on main memory speed, HBM greatly increases available bandwidth for memory-bounded applications, etc. This heterogeneous landscape of hardware increases the possible search space to come to an optimal query plan, though. Therefore, it is important to reduce the complexity by isolating the factors that mostly influence query performance.
In this paper, we look at the possibilities of query planning for the transactional stream processing model, highlighting opportunities but also limitations of different approaches. In addition to the actual execution plan, the physical representation of states must also be considered in this model. Thus, the following aspects must be taken into account when selecting queries: ① the state representations (the underlying data structures), ② the data placement (on which medium the data is stored), and ③ the algorithms (i.e. the appropriate

[1] TU Ilmenau, Databases & Information Systems Group, Ilmenau, Germany, *first.last*@tu-ilmenau.de

doi:10.18420/btw2019-ws-05

implementation of the operators). All three points play closely together and can hardly be considered separately. The questions that arise from this are, on the one hand, which parameters are necessary for a suitable selection and, on the other hand, which of them are actually accessible. Since stream queries usually run on a long-term basis, data rates and characteristics (e.g., skew) may also change over time. This raises the further question whether plans should, therefore, be repeatedly adjusted or rather a plan should be as robust as possible right from the start.

2 Related Work

Query Planning. Cost models and query execution planning is a widely studied topic in the DBMS world. Therefore, we will only discuss the most profound and relevant work for us in the following. The work of Manegold [Ma02] was one of the first proposing a hardware-based cost model for modern CPUs considering sophisticated features like cache hierarchies. It investigated the typical memory access pattern of database operations and distinguished between logical, algorithmic, and physical costs. Sixteen years later, Zeuch [Ze18] has been working extensively on query planning for today's generation of CPUs that employ even more utilization techniques. A counter-based approach was proposed that progressively optimizes the queries at runtime. Krämer [Kr07] have dealt intensively with continuous queries and a corresponding cost-based resource management especially for sliding windows. Here, an adaptive approach was chosen too, where the window size is dynamically adjusted to the available resources within predefined bounds. Karnagel et al. [Ka17] discuss another adaptive approach to optimally utilize query execution on heterogeneous hardware. Their focus is on the optimal placement of work on compute units, particularly for OpenCL-based DBMSs.
It was found that little work has been done on a cost model for data stream processing. At the same time, the question arises which concepts of DBMSs are applicable to transactional stream processing. More details to these and further approaches are discussed in Sect. 4.

Transactional Stream Processing. The STREAM [Mo03] project was one of the first to deal with the combination of relational databases and stream processing. They have considered continuous queries, possibly with a synopsis (≡ state), and provide a way to share subplans and states. The plan selection is based on stream constraints with the sole goal of minimizing the use of resources. Although constraints can be modeled with the help of punctuations, there is no actual transaction management. Instead, a global scheduler coordinates the successive execution of the individual operators by assigning time slots.
A more recent system that supports transactions in data stream processing is S-Store [Me15], which is implemented on top of the main memory OLTP system H-Store [Ka08]. It can guarantee the ACID properties by reapplying the existing transaction concepts of H-Store for time-varying relations on, e.g., windows and streams. Queries over streams are expressed as dataflow graphs, where each node represents a stored procedure and the edges define the

execution order. Each execution of a stored procedure combined with an input batch forms a transaction. Although it is not addressed directly, it can be assumed that query planning was also inherited from H-Store.
Botan et. al. [Bo12] have defined a unified transactional model to combine traditional transaction and data stream processing. This is realized by transforming continuous queries into a sequence of one-time queries and, thus, treating streams and relations uniformly. They implemented this model on top of an existing storage manager and extended it by a transaction management component dealing with concurrency and failures. Continuous query planning, however, was not really addressed here either.

3 Background

3.1 Transactional Stream Processing

The transactional stream processing model considered in this context distinguishes between two types of data occurrence: *tables* to represent states and *streams* for data flowing through the queries. Similar to the relational model, tables are a finite collection of data divided into rows and columns. Streams, on the other hand, can be potentially infinite and are typically defined as a sequence of tuples. While streams are volatile, tables also require a physical representation.
Additionally, operators are needed to link these two concepts. We divide them into three classes: *TO_TABLE* which updates tuples from a stream in a table, *TO_STREAM* producing a stream of tuples based on either events in a table or the whole table, and *FROM* to attach to a stream or to read data from a table. These operators are illustrated in Fig. 1.

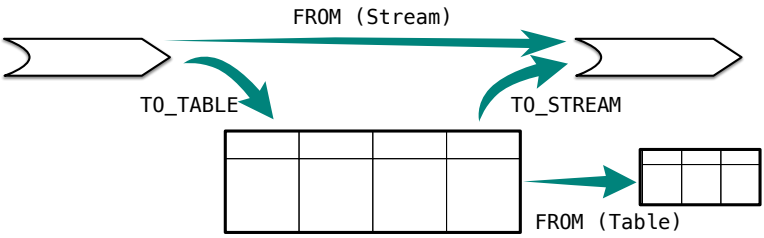


Fig. 1: Overview of the transactional semantics for data stream processing.

Manipulations of states and streams is performed within a transactional context to ensure atomicity and durability during writing. In order to guarantee atomicity, transaction boundaries are required, which could be, for instance, automatically defined per stream element or propagated alongside the actual data via punctuations. *FROM* and *TO_STREAM* operators, on the other hand, provide different isolation levels where the latter requires an additional trigger policy.

3.2 Hardware Considerations

Today's hardware landscape is becoming increasingly heterogeneous and there is a development from general-purpose processors to special hardware for specific applications or operators. This means that the corresponding application scenarios must first be identified in our model. Since GPUs can process massively independent data in parallel, but have comparatively high shipping costs, it is questionable whether they are suitable for stream processing at all or whether SIMD registers are sufficient. Possible scenarios could be ad-hoc queries on very large states or linear algebra operators for matrix calculations. FPGAs, on the other hand, can already be soldered on CPU sockets and, thus, would not have too high shipping costs. These could, e.g., be programmed for special operators or operator pipelines which are highly CPU bound [Mü09]. Another aspect of heterogeneous hardware are modern high-speed networks and technologies such as Infiniband, RoCE, and RDMA respectively, which can be advantageous for distributed data management systems [Bi18]. Here, however, we focus on local stateful operations, which are more likely to be memory or storage bound. Therefore, we aim our attention regarding modern hardware at Non-Volatile Memory (NVM) and many-core processors, which we will describe briefly in the following.

Many-Core Architecture. Since there are physical limits to the clock rate in the development of CPUs, nowadays the number of integrated cores is increased to enable high parallelism. In contrast to multi-core CPUs, the many-core architecture offers even more features for parallelization, such as more cores, each with hyperthreading, and wider SIMD registers. However, the high number of densely packed cores creates a strong heat, which is counteracted by simpler cores and a lower clock speed which in turn leads to poorer singlethreaded performance. The Intel® Xeon Phi™ KNL, for instance, has up to 72 cores (@ 1.5 GHz). In addition, the KNL features multi-channel DRAM (MCDRAM) which is a variant of high-bandwidth memory (HBM), to increase the bandwidth up to 400 GB/s. For query planning, the degree of parallelization and the partitioning of states is of particular interest. It is also important to consider whether it is worth using HBM and in which mode.

Non-Volatile Memory. So that states in our model can survive power loss or system failures, the data must be stored non-volatile. Whereas in the traditional way of persisting data, I/O has accounted for the majority of the cost, this could change significantly with the advent of NVM. This technology promises to combine the byte-addressability and low latency of DRAM with the persistence and density of block-based storage media. However, NVM suffers from a limited cell endurance and read-write asymmetry regarding the latency. These characteristics mean for an optimizer that storage accesses are no longer clearly dominating the costs. Furthermore, when selecting an execution plan and the data representation of the states, the properties of the device must be considered. For NVM, writes should be exchanged for multiple reads if possible to counteract the lower endurance and read-write asymmetry. In addition, byte-addressability can be exploited with regard to atomicity.

4 Continuous Query Planning

As stated earlier, when optimizing continuous queries not only the execution plan but also the physical schema design must be determined. This applies in particular to queries with stateful operators. Depending on the application, there can be various optimization goals, such as low latency, high throughput or resource efficiency. Achieving these goals depends on various factors. The objective is to identify the most influential of them. Therefore, we first look at important parameters for query planning for stateful operators. We then discuss various approaches how an optimizer can get and use these to construct suitable plans.

4.1 Stateful Processing

There are basically three scenarios when a state needs to be accessed with ACID guarantees: (a) persisting modifications to a state, (b) querying a shared state, and (c) recovering a state after a failure. Depending on the ratio and frequency with which these scenarios occur, combined with their access profile, several types of states are sometimes more suitable than others. State representations could be, e.g., a durable log, hash-based structures, tree-based structures, sorted arrays/lists, graphs, or even various combinations of these. In addition, there are available hardware and resource factors with appropriately optimized algorithms and data placement. This basically results in a huge decision space. The task of the optimizer is to select a suitable combination of state representation, data placement, and access algorithms. For this, it receives the requirement profile and the available resources as input and matches them with the state types and algorithms implemented in the system. Next, we will go into more detail about the some possible parameters.

Data-driven Parameters. Regarding the access profile, a distinction can be made primarily between dominant kinds of access (persisting, querying, recovery) and their ratio and frequency as described above. In a transactional system, there are additional very influential factors that affect the guarantee of ACID properties. For a shared state it is important to know the number of concurrent accesses as well as the approximate degree of contention. Based on this, an appropriate concurrency control protocol would have to be selected. The question with data stream processing is where to get this information when creating a query if no data has been seen yet. One possibility would be to rely on statistics and heuristics based on previous queries. A modified form of sampling, if feasible, would also be conceivable. Since a state is always part of an operator, it may be possible to derive typical access patterns from it. It becomes more complicated if the state is defined via a general purpose table. In this case, user annotations would be helpful. In order not to burden the users with the task, a dynamic adaptation based on the actual workload might be a better approach. However, this can lead to a situation in which another data representation or placement is suddenly classified as much more efficient. Therefore, a potentially expensive state conversion or migration may be worthwile. This would throttle the performance for

a certain time, especially during a high demand for the affected state. This is why robust query plans are often used that can withstand a wide range of scenarios.

Hardware-based Parameters. In addition to data-related parameters, there are also hardware-based factors that can be indispensable for determining the optimal state representation, data placement, and access algorithms. The first relevant factor is the general availability of specific processing units (e.g., FPGA, GPU, many-core CPU) or, regarding our focus, memory and storage variants (e.g., HBM, NVM, SSD, HDD). Moreover, the available numbers of capacity, latency, and bandwidth are interesting with regard to storage media. For example, due to limited space or performance reasons, a data structure could be stored across multiple memory layers, as it is the case with the LSM-tree or hybrid structures such as the FPTree [Ou16]. Sticking with LSM-trees, a different buffering strategy could be chosen depending on the memory type [Le17]. Furthermore, depending on the access granularity of the devices, the basic organization (blocks, pages, tuples, etc.) of the data must be taken into account. For PUs, on the other hand, the degree of parallelizability and the clock frequency are important. So it is possible to use different partitioning approaches here to leverage the heterogeneous units. But the transport and merge times of a co-processor must also be included in order to determine the profitability (cf. [Po17]). In practice, it becomes even more complicated, especially with modern processors since sophisticated techniques like caching, prefetching, branch prediction, reordering, etc. are involved (cf. [Ze18]). Below, we outline how different approaches could acquire and use these parameters.

4.2 Optimizing Strategies

Existing Models. In the following, we discuss three classes of optimizing approaches, which we categorize as hardware-oblivious, hardware-conscious, and learning models. Cost models without hardware-related parameters (being *hardware-oblivious*) were relatively common in the early database systems. Such models are usually application-based [LN96], which means that a DBMS is manually tuned to the underlying hardware by identifying and re-implementing major performance bottlenecks [WK90]. This allows to reduce the number of parameters and, thus, the search space of cost models. However, a disadvantage of this approach is that a change of hardware or software requires manual adjustments again.
A cost model that depends on hardware parameters like memory access latencies or cache sizes is more robust against changes in general (being *hardware-conscious*). The usual way to get this hardware information is by running a calibration tool [Ma02] whenever hardware changes. The difficulty of such an approach is mainly to correlate those parameters correctly since they can influence each other. To point an example, a higher clock frequency also leads to a reduced latency of main memory accesses.
Recent research focuses on the combination of database models and machine learning. By feeding parameters into well-trained machine learning models, manual tuning and cost model adaptations are unnecessary. In [Or18], they applied deep reinforcement learning to

incrementally find the optimal query execution plan through subqueries. However, learned query optimization models can become difficult to configure correctly and also represent a black box, which makes it difficult to interpret how a certain result is achieved.

Strategy for Transactional Stream Processing. For the transactional stream processing model, an overall solution could be that the state type is determined by the access pattern (e.g., using operator characteristics) and the optimizations regarding data placement and algorithms are set by the hardware factors. Whether this is determined by a learning or a concrete cost model still has to be shown. As we have seen, there is a plethora of parameters, and only the most influential factors should be considered to avoid making the model too complex. Moreover, not all information is available at all times, which means that certain procedures may be excluded in advance. Therefore, we think an adaptive or progressive optimization might be quite reasonable.
As a concrete proposal, we extend our cost formula for stateful operations from [Po17]. Calibrated hardware parameters are assumed. Since we include NVM in the consideration, we separate read and write accesses. We further differentiate between state ($f_{<s>}$) and operator ($f_{<op>}$) dependent cost factors. The former contains the average state-specific and additional synchronisation (ACID) costs per access and the latter the logical operator-specific read or write accesses. The first factor must also be determined depending on the underlying hardware. This can be done either by another cost formula based on, e.g., the devices latency or by using performance counters (cf. [Ze18]). Using these factors, we provide a formula for every kind of state access (persisting, querying, and recovery) in Eqs. (1) to (3).

$$c_{<op>} = f_{<op>_r} \cdot f_{<s>_r} + f_{<op>_w} \cdot f_{<s>_w} \quad (1)$$

$$c_{<s>\_q} = \sigma \cdot <s>_{size} \cdot f_{<s>_r} \quad (2)$$

$$c_{<s>\_rec} = \Delta \cdot <s>_{size} \cdot (f_{<s>_r} + f_{<s>_w}) \quad (3)$$

Whereas the first depends on the operator, the other two cost formulas are rather state dependent ($<s>$). The persistence costs consist of the read and write costs for the operator-typical access and depend on the underlying state and hardware. State querying is similar to typical cost models and, thus, cardinality and selection (σ) based. For recovery, we can differentiate between three state dependent cases. The first case is when no recovery is necessary for the state (only atomically visible changes), then Δ and the total cost is zero. As typical in DBMS, however, certain checkpoints could also be set, whereby only changes up to the last checkpoint need to be considered. This distance (Δ) could, for instance, be given as a percentage of the total size. In the worst case ($\Delta = 1$), if there are no checkpoints, the entire state may have to be read and parts rewritten or removed. The precision of this model in practice still has to be examined in future work.

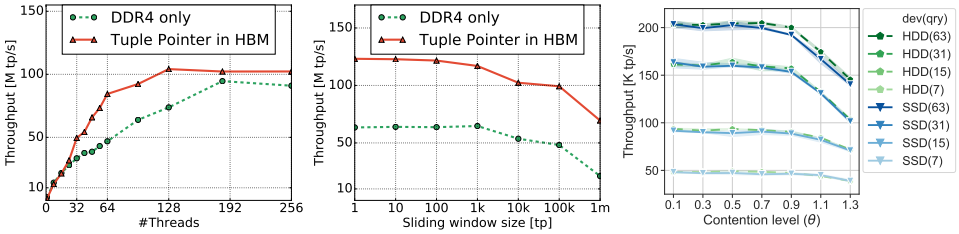
5 Experiments

The question we want to address in this section is whether it is possible to predict performance behavior by a cost model on the memory layer. For reasons of space, we pick two comparisons: DDR4-HBM and HDD-SSD, examining the performance of a sliding window operation and an LSM-Tree. We expect a higher bandwidth to result in improved sliding window performance if the rest of a query has few computations or the degree of inter-query parallelism is high (hence more bandwidth is used). For the LSM-Tree, its structure as well as modern prefetching and caching techniques should mask disk characteristics, almost eliminating performance differences.
The experiments run on the Intel® Xeon Phi™ Knights Landing 7210 with 64 cores à 4 threads @ 1.5 GHz (max), 96 GB DDR4, Linux kernel 3.10, and GCC 7.3. We compare different performance metrics regarding memory (DRAM vs. HBM) and storage technologies (HDD vs. SSD). Since we can only emulate NVM (using DRAM) so far, we have omitted this technology from the experiments for now. The tests run with our data stream processing framework PipeFabric[2] with prototypical transaction support.

5.1 Sliding Window

A sliding window is a common operator for data stream processing. It keeps track of incoming tuples, invalidating older ones to keep only the newest tuples available. The sliding effect allows to continuously update the window tuple by tuple by *sliding* over the input stream. In PipeFabric, a sliding window is a list of tuples where new entries are added at the end of the list. Older tuples are removed by a count- or time-based check afterwards. If the list is small enough to fit in one of the caches, fast access can be guaranteed. In addition, higher memory bandwidth can be useful for pushing down recent changes of the list from cache to main memory (inclusive caching). Furthermore, throughput of a window depends on additional operators that a thread has to run. The more computations per tuple must be done, the less bandwidth can be utilized. Fig. 2 shows the performance of inserts and deletes of a sliding window operator with (a) varying thread numbers (each thread running an own sliding window operator) and (b) varying the window size.
When inter-query parallelism gets higher (with more than 20 threads), the HBM can sustain a higher throughput on average for each sliding window, as expected. Increasing the window state size to keep track of more tuples leads to performance degradation, since the state cannot be kept in the CPU caches (inserts at the end, deletes at the front). The number of TLB and page misses also increases when more than 100k tuples per window are stored. While throughput behavior can be predicted for varying window sizes, it is hard to anticipate the query workload in real systems due to the interference of heterogeneous queries. Memory access can differ fundamentally between operators, not to mention UDFs with unpredictable behavior before execution. We therefore suggest a calibration approach for the available stream operators like in our previous work [Po17] as a possible approximation.

[2] PipeFabric: `https://github.com/dbis-ilm/pipefabric`



(a) Throughput with varying thread count (10k window size).

(b) Throughput with varying window size (64 threads).

(c) LSM-tree with varying devices (dev) and ad-hoc queries (qry).

Fig. 2: Transactional Stream Processing experiments.

5.2 LSM-Tree as State Representation

This experiment aims to show that hardware and data structures are interdependent and that it is essential which and where the hardware parameters are added to the model. Therefore, we run the tests in our transactional stream processing framework having one continuous and multiple ad-hoc queries accessing the same states. As state representation we used an LSM-tree (RocksDB[3]) and varied the number of ad-hoc queries and the contention level. Due to the nature of LSM-trees where write costs are amortized over time, we expect little difference in throughput when storing it on an HDD and an SSD respectively. Theoretically, the SSD is 10 times faster than the HDD. The results are shown in Fig. 2c.
As expected the performance for HDD and SSD is nearly the same. Instead, the concurrency control protocol and the contention have more influence for such a data structure. This also raises the question whether the use of NVM is worthwhile in this case. However, with other data structures or queries that are, e.g., more bandwidth dependent, the selected device could have a severe impact. Thus, there is an interdependence between hardware and data structures which needs to be determined in the state-specific factor ($f_{<s>}$).

6 Conclusion

There have already been a number of studies on query planning in the DBMS and DSMS world. For transactional stream processing, the task is to unify them into one model that processes both streams and tables. Since there is hardly any apriori knowledge about the data in streams, a progressive approach seems to be useful. However, as constant changes can be very expensive, we think that a combination of adaptive and robust query planning is necessary. In this paper, we have briefly discussed which data- and hardware-based parameters can be chosen for this and how they can be used. In doing so, we drew attention to the opportunities and limitations of existing approaches and have underpinned these with first experiments. A fully functional model is part of future work.

[3] RocksDB (version 5.15.10): `https://github.com/facebook/rocksdb`

Acknowledgments

This work was partially funded by the German Research Foundation (DFG) within the SPP2037 under grant no. SA 782/28.

References

[Bi18] Binnig, C.: Scalable Data Management on Modern Networks. Datenbank-Spektrum 18/3, pp. 203–209, 2018.

[Bo12] Botan, I. et al.: Transactional Stream Processing. In: EDBT. Pp. 204–215, 2012.

[Ka08] Kallman, R. et al.: H-Store: A High-Performance, Distributed Main Memory Transaction Processing System. PVLDB 1/2, pp. 1496–1499, 2008.

[Ka17] Karnagel, T. et al.: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. PVLDB 10/7, pp. 733–744, 2017.

[Kr07] Krämer, J.: Continuous Queries over Data Stream - Semantics and Implementation, PhD thesis, University of Marburg, Germany, 2007.

[Le17] Lersch, L. et al.: An analysis of LSM caching in NVRAM. In: DaMoN@SIGMOD. 9:1–9:5, 2017.

[LN96] Listgarten, S.; Neimat, M.: Modelling Costs for a MM-DBMS. In: RTDB. Pp. 72–78, 1996.

[Ma02] Manegold, S.: Understanding, Modeling, and Improving Main-Memory Database Performance, PhD thesis, 2002.

[Me15] Meehan, J. et al.: S-Store: Streaming Meets Transaction Processing. PVLDB 8/13, pp. 2134–2145, 2015.

[Mo03] Motwani, R. et al.: Query Processing, Approximation, and Resource Management in a Data Stream Management System. In: CIDR. 2003.

[Mü09] Müller, R. et al.: Data Processing on FPGAs. PVLDB 2/1, pp. 910–921, 2009.

[Or18] Ortiz, J. et al.: Learning State Representations for Query Optimization with Deep Reinforcement Learning. In: DEEM@SIGMOD. 4:1–4:4, 2018.

[Ou16] Oukid, I. et al.: FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In: SIGMOD. Pp. 371–386, 2016.

[Po17] Pohl, C. et al.: A Cost Model for Data Stream Processing on Modern Hardware. In: ADMS@VLDB. 2017.

[WK90] Whang, K.; Krishnamurthy, R.: Query Optimization in a Memory-Resident Domain Relational Calculus Database System. TODS 15/1, pp. 67–95, 1990.

[Ze18] Zeuch, S.: Query Execution on Modern CPUs, PhD thesis, 2018.

Skew-resilient Query Processing for Fast Networks

(Extended Abstract)

Tobias Ziegler[1] Carsten Binnig[1] Uwe Röhm[2]

1 Introduction

Motivation: Scalable distributed in-memory databases are at the core of data-intensive computation. Although scaling-out solutions help to handle large amounts of data, more nodes do not necessarily lead to improved query performance. In fact, recent papers have shown that performance can even degrade when scaling out due to higher communication overhead (e.g., shuffling data across nodes) and limited bandwidth [Rö15]. Thus, current distributed database systems are built with the assumption that the network is the major bottleneck [BH13] and should be avoided at all costs.

In recent years, high-speed networks (e.g., InfiniBand (IB)) with a bandwidth close to the local memory bus [Bi16] have become economically viable. These network technologies provide Remote Direct Memory Access (RDMA) to allow direct memory access to a remote host and also reduce the latency of data transfer through bypassing the remote's CPU [In17, Gr10]. Therefore, the assumption that the network is the bottleneck no longer holds.

Consequently, recent research has focused on integrating RDMA-enabled high-speed networks into existing database systems designed along a *Shared-Nothing Architecture (SN)* [Rö16, LYB17]. This architecture co-locates computation and data to reduce the communication overhead in a cluster. Although combining a SN with IB's higher network bandwidth enables scalability to a certain extent, this approach fails if the data or workload is skewed and cannot be evenly partitioned. The root cause is that classical query execution schemes assume that each partition is processed by one node. Since nodes with larger partitions must process more data, they may become a bottleneck and hinder the overall scalability. In consequence, only utilizing the higher bandwidth without adapting the database architecture and query execution, does not automatically lead to improved scalability [Bi16].

Contributions: In this paper, we present a new approach to execute distributed queries on fast networks with RDMA. Our main contribution is a novel execution strategy, which enables collaborative query processing by remote work stealing to mitigate skew, as this is a common issues that hinders scalable query execution [WDJ91, Ly88]. Moreover, we implement this execution strategy in our prototype engine I-Store and show that it introduces almost no overhead to handle skew.

[1] TU Darmstadt, Data Management Lab - Informatik, Germany, firstname.lastname@cs.tu-darmstadt.de
[2] University of Sydney, School of Computer Science, Australia, uwe.roehm@sydney.edu.au

doi:10.18420/btw2019-ws-06

2 System Overview

I-Store builds upon an architecture specifically designed along fast networks — called the network-attached-memory (NAM) architecture [Bi16, Sa17]. The NAM architecture logically decouples *compute nodes* from *storage nodes* and uses RDMA for communication between all nodes as shown in Figure 1. The idea is that storage nodes provide a shared distributed memory pool that holds all the data and auxiliary data structures, which can be accessed via one-sided RDMA from all compute servers. In contrast to the traditional SN database architecture which physically co-locates the query execution with the storage location, the NAM architecture separates them. Due to the separation of compute and storage servers, computation can be executed independently of its storage location. Thus the computation is less sensitive to workload skew, since in case of a straggling compute server any other compute server can help. Therefore, in the NAM architecture data locality is not a hard requirement but only a tuning parameter that can be added to speed-up workloads.

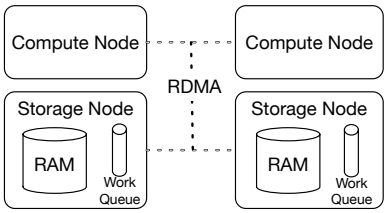


Fig. 1: The NAM Architecture

Our skew-resilient execution engine I-Store relies on fine-grained work elements that are stored inside *work queues* as depicted in Figure 1. A query execution is broken down into multiple work elements, similar to the morsel-driven execution on single node database systems as proposed in [Le14]. The queue-based execution, in combination with the possibility to access every data item via RDMA, allows load balancing by *work stealing*. However, not every data access between a compute and a storage node needs to be via RDMA. If we allow the (logical) compute and storage nodes to be co-located on the same (physical) cluster node, I-Store can use local memory access instead of RDMA for all local available data.

3 Remote Work Stealing

Queue-based Query Execution: I-Store implements a *queue-based query execution* strategy that allows fined-grained execution by organizing work in smaller chunks, namely *work elements*. A work element encodes the operation (e.g., an operator) and on which part of the data the operation is executed. The work elements are then stored in work queues, which are placed on the storage nodes as shown in Figure 1. Each work queue manages work elements which belong to the same partition as indicated in Figure 2. In order to process a query, a compute node is initially assigned to one work queue, i.e., to one partition. A compute node pops the work elements sequentially from its respective queue. Based on the information in the work element, a compute node processes the specified data pages. Once the assigned queue of a compute node is empty (i.e., if all work elements have been processed), this node starts stealing work elements remotely from other straggling compute nodes (i.e., from their work queues).

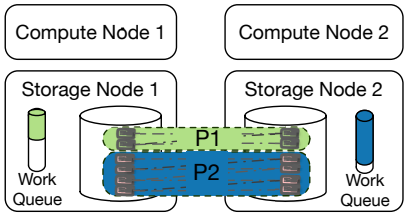


Fig. 2: The NAM-Partitioning

NAM-Partitioning: A problem of remote work stealing is that multiple compute nodes may try to steal data from the same straggling compute node, which would cause the available bandwidth of the storage node to be shared among them. To achieve a more balanced network usage, I-Store implements a novel partitioning scheme, called *NAM partitioning*, which distributes data equally among all storage nodes independent of the data distribution. In NAM partitioning, a partition is split into many small *pages*. These pages are then distributed evenly in a round-robin fashion to all storage nodes. To maintain a logical partition, the pages are linked together via a remote pointer to form a distributed linked list of pages, as indicated in Figure 2. To avoid pointer chasing, I-Store implements a *prefetching* mechanism: Using the remote pointer from an already fetched data page, I-Store can overlay computation with data retrieval by exploiting the RDMA-network card as a co-processor to prefetch the next page.

4 Experimental Evaluation

In the following, we present the results of a small experimental performance evaluation of I-Store to validate the performance benefits of work stealing and NAM-partitioning. The evaluation was conducted on a four-node cluster connected via a single InfiniBand FDR 4X switch using one Mellanox Connect-IB card[3]. Each server had two Intel Xeon E5-2660 v2 processors (20 cores in total) and ran on Ubuntu 14.01 Server Edition (kernel 3.13.0-54-generic). I-Store was compiled using gcc 4.8.5.

To be able to assess workload skew, we generated two synthetical datasets consisting of four relations (A, B, C, D), one where the partition key is following a uniform distribution (i.e., all partitions have the same size), while the second dataset followed a Zipf distribution with $z = 1.25$ (i.e., one partition dominates the others in size). Each record in the dataset consisted of three attributes (PK, $payload$, FK) similar to [Rö16], with a tuple width of 24 Bytes. In total, each table contained 420M records, which yields a total size of about 10 GB per table. The query workload consisted of SQL queries that execute three joins (i.e., A⋈B⋈C⋈D) with an additional selection on each inner relation (A⋈ σ_X(B)⋈ σ_Y(C)⋈ σ_Z(D)). We mainly concentrated on joins since these operations are widely used in many analytical SQL workloads and we thus can show the effects of or work stealing algorithms for a wide class of analytical SQL queries.

We assess the runtime of this workload on two system architectures: The baseline is the *shared-nothing* architecture with co-partitioned (A,B) tables to minimize network transfers, while I-Store implements a *NAM architecture*. We used the same cluster with four physical nodes for this experiment. We configured I-Store to co-locate one compute and one storage

[3] Theoretical bandwidth of 6.8 GB/s per incoming and outgoing link

node on each single physical node. We measured I-Store in four different configurations: Plain I-Store without further optimizations, with work-stealing (WS), with NAM-partitioning (NAM-Part), and with local-access optimization (LocOpt) enabled (i.e., data accesses do not use RDMA but local memory accesses). For this paper, work stealing was done on the granularity of the selection operators (scan and pre-filtering of data pages).

As expected, the uniform dataset is the ideal case for shared-nothing, however I-Store with all optimizations shows a similar performance (1220 ms vs 1251 ms). Interestingly, work stealing can improve query execution even for homogeneous clusters: I-Store WS was 5% faster than plain I-Store. This shows that even if the dataset follows a uniform distribution, it can happen that individual nodes become slower than others due to external factors, for example in a shared experiment cluster like ours. NAM-partitioning (I-Store WS+NAM-Part) further decreases the runtime by another 5% since it balances network access among storage nodes and avoids delays due to network congestion. The last optimization leverages local-access (I-Store with LocOpt) and performs similarly as the baseline.

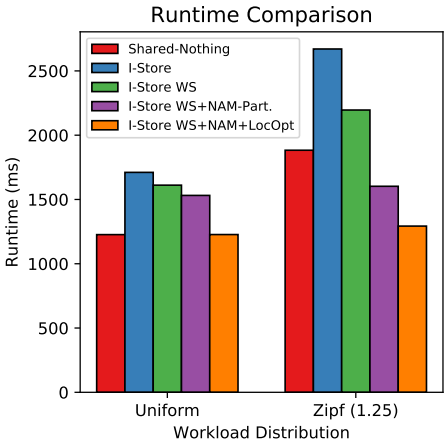


Fig. 3: Performance of Different Execution Strategies on Uniform and Skewed Data

For the skewed distribution, the runtime of shared-nothing (1917 ms) was dominated by the slowest node which needed to process 4.1 GB per partition. I-Store without any optimizations takes the longest to finish (2699 ms), but if work-stealing is enabled the runtime is close to our baseline. With NAM-Partitioning enabled, I-Store outperforms shared-nothing and shows the effect of network congestion. The runtime with NAM-partitioning compared to the vanilla work stealing approach is reduced by a 35%. In both distributions I-Store with all optimizations performs best. Additionally, the overhead induced by the skewed workload is only around 60 ms compared to the uniform execution.

5 Conclusion

This paper explored techniques to better align query execution with direct memory access over high-speed networks. We presented I-Store, a novel queue-based query execution engine that efficiently supports load balancing via NAM-aware data partitioning and work stealing. In a short evaluation we showed that I-Store can handle skew with almost no overhead. As an avenue of future work, we plan to implement different work stealing strategies and show that our work stealing approach is applicable to a variety of operators.

References

[BH13] Babu, Shivnath; Herodotou, Herodotos: Massively Parallel Databases and MapReduce Systems. Found. Trends databases, 5(1):1–104, November 2013.

[Bi16] Binnig, Carsten; Crotty, Andrew; Galakatos, Alex; Kraska, Tim; Zamanian, Erfan: The End of Slow Networks: It's Time for a Redesign. Proc. VLDB Endow., 9(7):528–539, March 2016.

[Gr10] Grun, Paul: Introduction to infiniband for end users. White paper, InfiniBand® Trade Association (IBTA), 2010.

[In17] InfiniBand® Trade Association (IBTA): , Infiniband Roadmap. `http://www.infinibandta.org/content/pages.php?pg=technology_overview`, 2017. Accessed: 2017-10-19.

[Le14] Leis, Viktor et al.: Morsel-driven parallelism: a NUMA-aware query evaluation framework in the many-core age. In: ACM SIGMOD. 2014.

[Ly88] Lynch, Clifford A.: Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distribution of Column Values. In: Proceedings of the 14th VLDB. VLDB '88, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 240–251, 1988.

[LYB17] Liu, Feilong; Yin, Lingyan; Blanas, Spyros: Design and Evaluation of an RDMA-aware Data Shuffling Operator for Parallel Database Systems. Proceedings of the Twelfth European Conference on Computer Systems - EuroSys '17, pp. 48–63, 2017.

[Rö15] Rödiger, Wolf; Mühlbauer, Tobias; Kemper, Alfons; Neumann, Thomas: High-speed Query Processing over High-speed Networks. Proc. VLDB Endow., 9(4):228–239, dec 2015.

[Rö16] Rödiger, Wolf; Idicula, Sam; Kemper, Alfons; Neumann, Thomas: Flow-Join: Adaptive skew handling for distributed joins over high-speed networks. 2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016, pp. 1194–1205, 2016.

[Sa17] Salama, Abdallah; Binnig, Carsten; Kraska, Tim; Scherp, Ansgar; Ziegler, Tobias: Rethinking Distributed Query Execution on High-Speed Networks. IEEE Data Eng. Bull., 40(1):27–37, 2017.

[WDJ91] Walton, Christopher B.; Dale, Alfred G.; Jenevein, Roy M.: A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins. In: Proceedings of the 17th VLD. VLDB '91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 537–548, 1991.

An Overview of Hawk: A Hardware-Tailored Code Generator for the Heterogeneous Many Core Age

Sebastian Breß[1] Henning Funke[2] Steffen Zeuch[1] Tilmann Rabl[1] Volker Markl[1]

Abstract: Processor manufacturers build increasingly specialized processors to mitigate the effects of the power wall in order to deliver improved performance. Currently, database engines have to be manually optimized for each processor which is a costly and error prone process. In this paper, we provide a summary of our recent VLDB Journal publication, where we propose concepts to adapt to performance enhancements of modern processors and to exploit their capabilities automatically. Our key idea is to create processor-specific code variants and to learn a well-performing code variant for each processor. These code variants leverage various parallelization strategies and apply both generic and processor-specific code transformations. We observe that performance of code variants may diverge up to two orders of magnitude. Thus, we need to generate custom code for each processor for peak performance. Hawk automatically finds efficient code variants for CPUs, GPUs, and MICs.

1 Introduction

Over the last decade, the main memory capacity has reached the terabyte scale. Main memory databases exploit this trend in order to satisfy the ever-increasing performance demands. As a result, they store data primarily in main-memory to eliminate disk I/O as the main bottleneck. As a result, memory access and data processing have become the new performance bottlenecks for in-memory data management [Ma00].

Current designs of main-memory database systems assume that processors are homogeneous, i.e., with multiple identical processing cores. However, todays hardware vendors break with this paradigm in order to circumvent the fixed energy budget per chip [BC11]. This so-called *power wall* forces vendors to explore new processor designs to overcome the energy limitations [Es11]. Hardware vendors integrate heterogeneous processor cores on the same chip, e.g., combining CPU and GPU cores as in AMD's Accelerated Processing Units (APUs). Another trend is *specialization*: processors are optimized for certain tasks, which already have become commodity in the form of *Graphics Processing Units* (GPUs), *Multiple Integrated Cores* (MICs), or *Field-Programmable Gate Arrays* (FPGAs). These accelerators promise large performance improvements because of their additional computational power and memory bandwidth. Thus, from a processor design perspective, the *homogeneous many*

[1] TU Berlin and DFKI GmbH, Berlin {sebastian.bress,steffen.zeuch,tilmann.rabl,volker.markl}@dfki.de
[2] TU Dortmund, Dortmund, henning.funke@tu-dortmund.de

doi:10.18420/btw2019-ws-07

core age ends [BC11]. The upcoming *heterogeneous many core age* provides an opportunity for database systems to embrace processor heterogeneity for peak performance.

Our goal is to empower database systems to automatically generate efficient code for any processor without *any a priori* hardware knowledge, thus making database systems fit for the heterogeneous many-core age. To achieve this goal, we proposed Hawk [Br18], a novel hardware-tailored code generator, which produces variants of generated code. By executing code variants of a compiled query, Hawk adapts to a wide range of different processors without any manual tuning. Hawk achieves low compilation times and executes queries on a wide range of processors. In this paper, we provide a summary of our recent publication in the VLDB Journal [Br18]. Hawk's code is available as open source.[3]

2 Overview of Hawk

To provide an architectural overview, we describe Hawk's role in the process of executing an SQL query. The SQL parser translates queries into relational query plans. After that, the query optimizer rewrites the query plan by applying common optimizations to obtain a query execution plan. On the next layer, Hawk provides a code generation back-end to perform query compilation for efficient query execution. To this end, Hawk compiles query execution plans just-in-time into machine code of a target processor.

Hawk's key feature is the generation of efficient code for processors of different architectures. Our approach follows the principles of query compilation [Ne11] as opposed to vector-at-a-time processing [Bo05], because query compilation has the largest potential of applying processor-specific optimizations. Hawk uses a three-step compilation process: 1) query segmentation, 2) variant optimization, and 3) code generation (see Figure 1). In general, Hawk receives a query plan as input and outputs optimized code for the underlying processors. This process centers around *pipelines*, i.e., non-blocking data flows. In particular, all operations in a pipeline are fused into one operator. The individual steps are as follows.

Query Segmentation. Hawk first segments query execution plans into pipelines using the produce/consume model [Ne11] (Step (1) in Figure 1). During this step, Hawk creates a *pipeline program* for each pipeline as the intermediate representation for a pipeline. A pipeline program consists of simple operations such as loop, filter, and hash probe and establishes the start point for optimization and target code generation.

Variant Optimizer. The initial pipeline program represents a hardware-oblivious blueprint as a starting point for processor-specific optimizations. Based on that, Hawk produces hardware-tailored code by applying modifications to the pipeline programs. A *modification* is a change to a pipeline program, which preserves its semantic but changes the

[3] https://github.com/TU-Berlin-DIMA/Hawk-VLDBJ

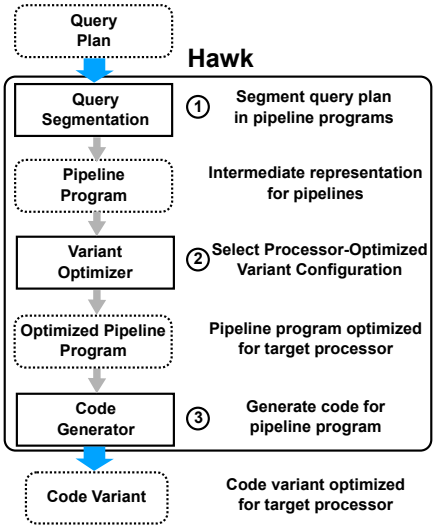


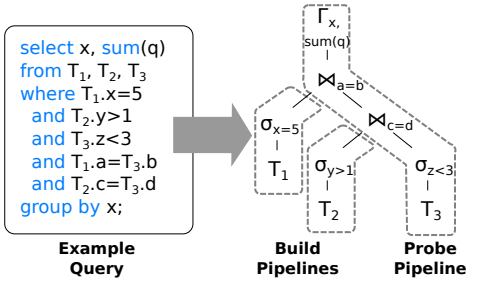
Fig. 1: Hawk's compilation process.

generated code (e.g., memory access pattern). A *variant configuration* captures all modifications of a pipeline program and thus provides a value for each supported modification. The set of all modifications defines the code generated by Hawk. The variant optimizer selects an efficient variant configuration for each pipeline program on a target processor (Step (2) in Figure 1). Note that Hawk automatically determines a variant configuration for each target processor without the need for manual tuning. In sum, Hawk applies the modifications specified in the variant configuration to the input pipeline program and returns an optimized pipeline program.

Code Generator. The code generator takes the optimized pipeline program as an input and produces the target code (Step (3)). We refer to the compilation result as *code variant*.

3 End-To-End Compilation Example

We exemplify the translation process of Hawk with the query illustrated in Figure 2. We also show the pipeline programs that are created during the translation. The query contains two hash joins, leading to a query plan with three pipeline programs. The pipeline programs describe three pipelines. The two build pipelines iterate over their input tables (T_1 and T_2), apply their filters, insert the matching keys into a hash table, and materialize their result. The probe pipeline program iterates over table T_3, applies it's filter, probes the hash tables, and performs the aggregation. Next, the variant optimizer selects a variant configuration, which describes the customized features for the translation. Then, the variant optimizer annotates each pipeline program according to the variant configuration. Hawk determines variant configurations by an offline-training workload of test queries using a structured experiment [Br18]. For simplicity and space restrictions, we continue our example for Build Pipeline 1 only.



Build Pipeline 1	Build Pipeline 2	Probe Pipeline
LOOP(T_1, ..)	LOOP(T_2, ..)	LOOP(T_3, ..)
FILTER(x=5, ..)	FILTER(y>1, ..)	FILTER(z<3, ..)
HASH_PUT(a, ..)	HASH_PUT(b, ..)	HASH_PROBE(a=c, ..)
PROJECT(a, x, ..)	PROJECT(b, ..)	HASH_PROBE(b=d, ..)
		HASH_AGGREGATE(x, sum(q), ..)

Fig. 2: Segmentation of example query into pipeline programs.

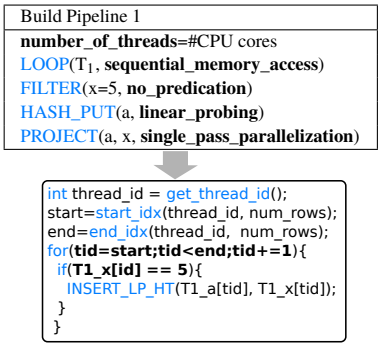


Fig. 3: Compiling an optimized pipeline program to target code.

Hawk supports several code modifications such as the memory access pattern (LOOP), branched predicate evaluation or software predication (FILTER), different hashing schemes (HASH_PUT), and parallelization strategies (PROJECT). We show a CPU-optimized pipeline program in Figure 3. It uses one thread per core, a sequential memory access pattern, a linear probing hash table, and single-pass parallelization. We illustrate the code generated by Hawk in Figure 3. On GPUs, we use a different parallelization approach called multi-pass to avoid high synchronization overhead between threads [Br18]. Finally, Hawk passes the code to the OpenCL compiler and executes the final kernel program.

4 Summary of Key Insights

In this paper, we provided an overview of Hawk, a hardware-tailored code generator that customizes code for a wide range of heterogeneous processors. Through hardware-tailored implementations, Hawk produces fast code *without manual tuning* for a specific processor. Our abstraction of pipeline programs allows us to flexibly produce code variants while keeping a clean interface and a maintainable code base. Code variants optimized for a particular processor can result in performance differences of up to two orders of magnitude on the same processor. Thus, it is crucial to optimize the database system for each processor.

Acknowledgments. This work was funded by EU project E2Data (780245), DFG Priority Program "Scalable Data Management for Future Hardware" (MA4662-5) and Collaborative Research Center SFB 876, project A2, and the German Ministry for Education and Research as BBDC I (01IS14013A) and BBDC II (01IS18025A).

References

[BC11] Borkar, Shekhar; Chien, Andrew: The future of microprocessors. Communications of the ACM, 54(5):67–77, 2011.

[Bo05] Boncz, Peter et al.: MonetDB/X100: Hyper-Pipelining Query Execution. In: CIDR. pp. 225–237, 2005.

[Br18] Breß, Sebastian et al.: Generating Custom Code for Efficient Query Execution on Heterogeneous Processors. The VLDB Journal, Jul 2018.

[Es11] Esmaeilzadeh et al.: Dark Silicon and the End of Multicore Scaling. In: ISCA. ACM, pp. 365–376, 2011.

[Ma00] Manegold, Stefan et al.: Optimizing Database Architecture for the new Bottleneck: Memory Access. The VLDB Journal, 9(3):231–246, 2000.

[Ne11] Neumann, Thomas: Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB, 4(9):539–550, 2011.

Workload-Driven Data Placement for GPU-Accelerated Database Management Systems

Christopher Schmidt[1], Matthias Uflacker[1]

Abstract: An increase in the memory capacity of current Graphics Processing Unit (GPU) generations and advances in multi-GPU systems enables a large unified GPU memory space to be utilized by modern coprocessor-accelerated Database Management System (DBMS). We take this as an opportunity to revisit the idea of using GPU memory as a hot cache for the DBMS. In particular, we focus on the data placement for the hot cache. Based on previous approaches and their shortcomings, we present a new workload-driven data placement for a GPU-accelerated DBMS. Lastly, we outline how we aim to implement and evaluate our proposed approach by comparing it to existing data placement approaches in future work.

Keywords: GPU-Acceleration, Coprocessor-Accelerated Databases, Data Placement

1 Introduction

Advances in hardware lead to an increased specialization of processors. For optimal performance a DBMS is advised to utilize all available hardware resources. GPUs raised an interest as a coprocessor for DBMSs, particularly for main memory column stores, due to their high memory bandwidth and parallel compute capabilities for query processing [Br14; Mo13; YLZ13]. Yet, their memory capacity is limited and data transfer, via PCIe, between on-device and host memory remains a bottleneck [GH11]. Note that with NVLink a faster interconnect is introduced, but it still lacks support by major CPU vendors, which rely on PCIe. Considering these challenges, query processing in coprocessor environments follows one of two major execution models.

The first execution model streams data into the device memory, enabling the execution of one or multiple operators, before results are transferred back into host memory. Despite of recent improvements by Funke et al. [Fu18], where the bandwidth transfer bottleneck is pushed from the interconnect to the GPU memory, initial data transfer accounts for a significant portion of the total execution time.

The second execution model limits query processing to data that is already resident in GPU memory to avoid data movement, in case it is harmful to query execution time [BFT16].

[1] Hasso Plattner Institute, Enterprise Platform and Integration Concepts, August-Bebel-Str. 88, 14482, Potsdam, vorname.nachname@hpi.de

doi:10.18420/btw2019-ws-08

Using the GPU memory as a hot cache requires a periodic data placement, which ensures to provide relevant data for query processing in the hot cache. This approach is limited by the device memory.

Recent GPU generations have seen an increase in memory capacity. Furthermore, combining multiple GPUs in a single system enables an even larger unified GPU memory space. Due to this relaxation of the on-device memory limitations, we see potential to revisit the idea of using GPU memory as a hot cache. Based on limitations in previous data-driven approaches, we propose a novel strategy for the data placement in GPU-accelerated DBMS with the goal to reduce the overall execution time for a given workload.

2 Related Work

GPU-accelerated DBMSs are an active area of research, aiming to fully utilize the hardware capabilities by implementing operators specifically for the GPU [He08] and by tackling challenges when integrating the coprocessor as an additional execution unit into the DBMS, i.e., operator placement [Br14; KHL17] or data placement [BFT16; Mo13].

In CoGaDB [BFT16], data placement is implemented as a background job migrating data into GPU memory. In order to decide for the data to be transferred they implement the strategies least-frequently used (LFU) and least-recently used (LRU). They state that both strategies perform similar in their evaluation. In MapD [Mo13], a strategy based on LFU using the GPU memory as a fast buffer pool is implemented. In contrast to CoGaDB, they partition columns into chunks, allowing a fine-granular data placement. Furthermore, a database administrator (DBA) is able to influence the data placement by pinning chunks into a layer of the memory hierarchy. In this context, we see potential for improvement with both LRU and LFU. While LRU lacks the capability to account for frequently accessed data partitions, LFU tends to keep certain data partitions longer than needed in GPU memory. We believe that even the expertise of a DBA is insufficient to fully eliminate the shortcomings. Additionally both strategies do not account for performance difference of migrated data.

3 Workload-Driven Data Placement

Using the GPU memory as a hot cache for the DBMS, i.e., with a data-driven execution model [BFT16], requires to place columns into GPU memory. Since the placement happens independent of query execution, a separate data placement job is executed. In order to account for changes in a given workload, this job has to occur in periodic intervals or has to be triggered by events, such as a violation of a service level agreement (SLA). The data placement aims to transfer columns following a defined optimization goal, i.e., the reduction of the overall execution time for a given workload. Deciding on the columns to fill the hot cache requires a strategy, i.e., LRU or LFU. In contrast to these, we propose to derive the

decision for data placement from a cost metric based on the profit of placing a particular column in GPU memory. In particular, we describe how this profit is updated during query execution by reusing runtime estimates acquired during operator placement.

In general, we define the profit of placing a column in GPU memory over all previously executed queries, by the sum of the profits of each individual query. The profit for a single query is determined by the difference in runtime of executing the query on the GPU to the runtime of executing the query on the CPU. The runtime for the GPU-based execution includes result data transfer only, as the input data resides in GPU memory. Furthermore, in case of a faster runtime on CPU, we assume a profit of zero. This constraint is based on the assumption that columns in GPU memory present a mirrored subset of all columns present in a higher memory hierarchy, as implemented in MapD [Mo13]. Hence, queries can run on the CPU even if columns required to answer the query are present in GPU memory.

Executing each query on both execution devices in order to obtain the accurate runtimes for the calculation of the proposed cost metric is infeasible in a productive DBMS. Therefore, we utilize estimates of the runtime, which are commonly used in query optimization or operator placement. Since query processing in heterogeneous execution environments requires an operator placement, which is based on runtime estimates for given operators [Ka14], these estimates can be reused for the calculation of the cost metric. The operator placement is either integrated into the query optimizer or executed by a dedicated placement optimizer [KHL17]. Therefore, we extend the according component to update the cost metric for each column during the placement decision. Note, letting the placement component update the cost metric allows to take a more fine-granular unit for calculating the profit into account. Thus, the profit per query is replaced by the profit per operator or sub-operator according to the unit for operator placement.

The data placement job evaluates the above defined cost metric for each column and transfers columns into GPU memory in periodic intervals. At first, a list containing a pointer to the column, its accumulated profit and its memory footprint is created. Next, the list is ordered starting with the column providing the highest profit per memory. Afterwards, the candidates for the hot cache, the GPU memory, are drawn from the list starting at the top until the amount of GPU memory reserved is exhausted. In case the current candidate does not fit into the remaining GPU memory the next fitting one is chosen, until no more columns fit within the given memory budget. Thus, we employ a greedy approach. This could be replaced for example by using linear programming to find an optimal solution. Note, parts of the GPU memory are reserved for intermediate and final results of operators. At last the hot cache is updated according to the list of candidates. This involves checking and evicting the columns resident in GPU memory, as well as, transferring new columns into it.

For the proposed workload-driven data placement it remains open to decide for an appropriate interval to update the data placement or for appropriate events that trigger the data placement. Furthermore, it needs to be evaluated whether the cost metric has to incorporate a temporal factor to account for changes in workloads faster.

4 Next Steps

In our work we propose a novel data placement strategy using a cost metric for a GPU-accelerated DBMS, which uses the GPU memory as a hot cache. In a next step, the workload-driven data placement is implemented in an existing open source GPU-accelerated DBMS. Currently, we investigate CoGaDB [Br14] and MapD [Mo13] for this purpose. Once integrated, our strategy is evaluated against existing approaches based on LFU and LRU. For the measurements we will use workloads, based on the star schema benchmark and a modified TPC-H benchmark, which have been used in [BFT16] to allow for comparability. Furthermore, we aim to focus the evaluation on worst case scenarios for LFU, LRU and our workload-driven data placement in order to investigate the limits of each strategy.

References

[BFT16] Breß, S.; Funke, H.; Teubner, J.: Robust Query Processing in Co-Processor-accelerated Databases. In: Proceedings of the 2016 International Conference on Management of Data. SIGMOD '16, ACM, San Francisco, California, USA, pp. 1891–1906, 2016.

[Br14] Breß, S.: The Design and Implementation of CoGaDB: A Column-oriented GPU-accelerated DBMS. Datenbank-Spektrum 14/3, pp. 199–209, Nov. 2014.

[Fu18] Funke, H.; Breß, S.; Noll, S.; Markl, V.; Teubner, J.: Pipelined Query Processing in Coprocessor Environments. In: Proceedings of the 2018 International Conference on Management of Data. SIGMOD '18, ACM, Houston, TX, USA, pp. 1603–1618, 2018.

[GH11] Gregg, C.; Hazelwood, K.: Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In: ISPASS IEEE. Pp. 134–144, 2011.

[He08] He, B.; Yang, K.; Fang, R.; Lu, M.; Govindaraju, N.; Luo, Q.; Sander, P.: Relational Joins on Graphics Processors. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. SIGMOD '08, ACM, Vancouver, Canada, pp. 511–524, 2008.

[Ka14] Karnagel, T.; Habich, D.; Schlegel, B.; Lehner, W.: Heterogeneity-Aware Operator Placement in Column-Store DBMS. Datenbank-Spektrum 14/3, pp. 211–221, Nov. 2014.

[KHL17] Karnagel, T.; Habich, D.; Lehner, W.: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. Proc. VLDB Endow. 10/7, pp. 733–744, Mar. 2017.

[Mo13] Mostak, T.: An Overview of MapD (Massively Parallel Database), 2013, URL: www.smallake.kr/wp-content/uploads/2014/09/mapd_overview.pdf.

[YLZ13] Yuan, Y.; Lee, R.; Zhang, X.: The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. Proc. VLDB Endow. 6/10, pp. 817–828, Aug. 2013.

Workshop Digitale Lehre im Fach Datenbanken

Digitale Lehre im Fach Datenbanken

Thomas C. Rakow[1], Heide Faeskorn-Woyke[2]

Auf dem Herbsttreffen 2008 in Düsseldorf beschäftigte sich die GI-Fachgruppe Datenbanken mit dem Thema "Quo Vadis: Formen der Datenbankausbildung und -weiterbildung", im Datenbank-Spektrum wurde anschließend eine Ausgabe diesem Thema gewidmet [Rak09]. Jährlich werden auf der Konferenzreihe E-Learning-Fachtagung Informatik (DeLFI) fachübergreifend ïnternet-, medien- und rechnergestützte Lehr- und Lernformen"vorgestellt [Del18]. Letztes Jahr wurde unter dem Motto "Digitalisierungs-(wahn)sinn? - Wege der Bildungstransformation"die Problematik digitaler Lehre aufgegriffen. In dem Workshop werden die Teilnehmer erarbeiten, wie die Lehre aktuell im Jahre 2019 im Fach Datenbanken aussieht und welche Erfahrungen damit gemacht wurden. Themen des Workshops sind:

- Implementierung und Anwendung von Tools für die Entwicklung von Datenbanken
- Erstellung und Nutzung von Lernumgebungen für Datenbanken
- Erstellung von E-Learnings, Animationen und Videos für Datenbanksysteme
- Datenbanken und Dashboards zur Nutzungsanalyse und -befragung
- Digitale Prüfungen im Gebiet Datenbanken

Der Workshop ist als Erfahrungsaustausch Lehrender zu sehen, aber auch Studierende sollen ihre Erfahrungen vorstellen. Hauptsächlich zielt der Workshop auf das kooperative Vorstellen von eigenen gesammelten Erfahrungen ab. In fünf Impulsdarstellungen werden Teilnehmer ihre Erfahrungen austauschen. Dabei werden sowohl die Vorgehensweisen zum Lehren im Fach Datenbanksysteme - digitale Kommunikation, Portale, Blended Learning - präsentiert als auch der Umgang aus der Lernerperspektive erfahrbar gemacht, wie mit einem Tool für die relationale Algebra sowie E-Learnings und Lernvideos. Auch die Beziehung zu ingenieurwissenschaftlichen Anforderungen der Softwaretechnik werden thematisiert. Das Erarbeiten (gemeinsamer) Best-Practices wird sich an die Darstellungen anschließen.

Die folgenden Beiträge werden im Workshop vorgestellt:

[1] Hochschule Düsseldorf, Fachbereich Medien, emailadresse@author1
[2] Technische Hochschule Köln, Fakultät für Informatik und Ingenieurwissenschaften, emailadresse@author2

doi:10.18420/btw2019-ws-09

Name	Hochschule	Titel
Heide Faeskorn-Woyke	Technische Hochschule Köln	Das eLearning Datenbank Portal edb2.0 in neuer Auflage
Sebastian Preetz	Universität Potsdam	Lernen am System am Beispiel Datenbanksysteme
Thomas Rakow, Jens Lambert, Björn Salgert	Hochschule Düsseldorf	Lehr- und Lernmaterialien digital kommunizieren und evaluieren
Günther Specht	Universität Innsbruck	RelaX – ein webbasiertes Tool für die relationale Algebra in der Lehre
Andreas Thor	Hochschule für Telekommunikation Leipzig	Ein Blended-Learning-Kurs für das Fach Datenbanksysteme

Literaturverzeichnis

[Rak09] Rakow, T.C., Faeskorn-Woyke, H., Schiefer, B., Vossen, G., Wäsch, J.: Tools für die Lehre im Fach Datenbanken. Datenbank-Spektrum 9(29): 5-13 (Themenschwerpunkt: Lehre in Datenbanken und Information Retrieval). D-Punkt Verlag, Heidelberg, Mai 2009. ISSN 1618-2162.

[Del18] E-Learning-Fachtagung Informatik 2018 (DeLFI), `https://www.delfi2018.de/` (Zugriff: 09.02.2019)

Workshop on Big (and Small) Data in Science and Humanities (BigDS 2019)

Preface

Workshop Big (and Small) Data in Science and Humanities (BigDS 2019)

Friederike Klan[1] Birgitta König-Ries[2] Peter Reimann[3] Bernhard Seeger[4] Anika Groß[5]

Over the last decade, we have witnessed a still ongoing digital transformation of science, society and economy. Advances in data acquisition and the expansion of the internet to an ubiquitous medium led to the era of Big Data, which is characterized by the availability of a huge and ever increasing volume of complex, interlinked and heterogeneous data. Remote and ground-based sensors in earth observation for example produce petabytes of data with increasing spectral, temporal and spatial resolution. Social media users generate content at a high rate. Information and knowledge encoded in those data have an enormous value potential, that if revealed, could help to better understand the mechanisms underlying complex systems such as the human society or our earth, to generate innovations and to make well-founded decisions.

Thus, the importance of data has dramatically increased not just in economy but also in almost all scientific disciplines, e.g. in meteorology, genomics, complex physics simulations, biological and environmental research, and recently also in humanities. The unprecedented availability of data stimulates a rethinking in scientific disciplines on how to extract useful information and on how to foster research. At the same time researchers face severe challenges in leveraging data, since appropriate data management, integration, analysis and visualization tools have not been available so far. Recent advances in the development of big data technologies and the progress in machine learning, semantic technologies and other areas seem to be not only useful in business, but also offer great opportunities in science and humanities. Scientific workflows need to be realized as flexible end-to-end analytic solutions to allow for complex data processing, integration, analysis and visualization of Big Data in various application domains.

The need to discuss real-world problems in data science as well as recent advances in big data technology with database researchers and scientists from various disciplines led to the first and second edition of the workshop on Big (and Small) Data in Science and Humanities

[1] DLR Institut für Datenwissenschaften
[2] Friedrich-Schiller-Universität Jena
[3] Universität Stuttgart
[4] Philipps-Universität Marburg
[5] Hochschule Anhalt

doi:10.18420/btw2019-ws-10

(BigDS) at BTW 2015[6] and 2017[7]. This years third edition of the BigDS workshop[8] co-located with the 18th symposium of "Database systems for Business, Technology and Web" accommodates the still growing interest in methods to efficiently and effectively manage and analyze Big Data. With workshop contributions from various disciplines we hope to promote the dialog between domain experts and data scientists.

The workshop program included a keynote talk on digital humanities by *Andreas Henrich*, where he discussed current approaches, challenges and applications in the context of data integration, data federation and data analysis for humanities. We further selected six contributions that address different challenges in the context of data-driven analytics. The papers contribute to the management and analysis of data from various domains, such as mobile data, automobile data, textual data like legal texts and bibliographic data as well as ecological data. The proposed approaches are related to the analysis and use of complex graphs and ontologies, item set mining and entity extraction as well as evaluation and quality criteria.

Two papers focus on methods and models in the context of data analytics. *Rost et al.* present an extension of the graph data management tool Gradoop to support temporal graph analytics. They added time properties to vertices, edges and graphs and used them within graph operators, e.g. to analyze temporal citation patterns as presented in a bibliographic usage scenario. *Spieß and Reimann* analyzed the regulation and control of vehicle components in automotive series production. They developed an adapted item set mining approach in order to successfully perform association analysis for the domain-specific problem of automatically identifying vehicles with high risk of failure.

Three papers deal with the analysis and extraction of information from textual data. *Cornelia Kiefer* presents and discusses quality indicators for textual data. Beside the quality of texts themselves, her aim is to predict the quality of text analysis results and to decide whether default text mining modules are likely to deal with the textual data or not. In her evaluation she investigates texts, e.g. from production, news and tweets using the proposed quality indicators. The goal of *Wehnert et al.* is to provide a decision support system for legal regulations, e.g. to inform companies about relevant regulatory changes that need to be considered. In this work, they use linked laws from their ontology of legal textbooks and developed a context selection mechanism to help users navigating in their legal knowledge base, e.g. to find all applications of a law. *Udovenko et al.* present a hybrid approach to extract entities from scientific publications in the ecological domain. They propose a framework including the use of domain-related ontologies for entity annotation, and run an initial evaluation for entity extraction from publications on biodiversity.

Finally, *Steinberg et al.* present a comparative evaluation for different software solutions that support the form-based collection of mobile data. Nowadays, mobile devices heavily

[6] http://www.btw-2015.de/?dms
[7] http://btw2017.informatik.uni-stuttgart.de/?pageId=BigDS
[8] https://btw.informatik.uni-rostock.de/index.php/de/call-for-workshops/bigds

support the data collection process, and users often build on existing infrastructure and software to collect and submit data. The paper reports on experiences with respect to the whole data collection workflow and compares eight existing tools in terms of their features and characteristics.

All contributions of this year's BigDS workshop give new domain-relevant insights and promote the use of generic as well as domain-specific methods for scientific data management and analytics. We would like to thank everyone who contributed to the workshop, in particular, the authors, the keynote speaker Andreas Henrich, the BigDS program committee, the BTW team, as well as all participants.

Workshop Organizers

Anika Groß (Hochschule Anhalt, DE)
Friederike Klan (DLR-Institut für Datenwissenschaften, DE)
Birgitta König-Ries (Friedrich-Schiller-Universität Jena, DE)
Peter Reimann (Universität Stuttgart, DE)
Bernhard Seeger (Philipps-Universität Marburg, DE)

Program Committee

Alsayed Algergawy (Friedrich-Schiller-Universität Jena, DE)
Peter Baumann (Universität Bremen, DE)
Matthias Bräger (CERN, CH)
Thomas Brinkhoff (Jade Hochschule, DE)
Jana Diesner (University of Illinois at Urbana-Champaign, US)
Johann-Christoph Freytag (Humboldt-Universität zu Berlin, DE)
Michael Gertz (Universität Heidelberg, DE)
Thomas Heinis (Imperial College London, UK)
Andreas Henrich (Otto-Friedrich-Universität Bamberg, DE)
Alfons Kemper (Technische Universität München, DE)
Jens Nieschulze (Georg-August-Universität Göttingen, DE)
Eric Peukert (Universität Leipzig, DE)
Norbert Ritter (Universität Hamburg, DE)
Kai-Uwe Sattler (Technische Universität Ilmenau, DE)
Holger Schwarz (Universität Stuttgart, DE)
Uta Störl (Hochschule Darmstadt, DE)
Andreas Thor (Hochschule für Telekommunikation Leipzig, DE)

Workshop Papers

Temporal Graph Analysis using Gradoop

Christopher Rost[1] Andreas Thor[2] Erhard Rahm[3]

Abstract: The temporal analysis of evolving graphs is an important requirement in many domains but hardly supported in current graph database and graph processing systems. We therefore have started with extending Gradoop for temporal graph analysis by adding time properties to vertices, edges and graphs and using them within graph operators. We outline these extensions and show their use in a bibliographic scenario to analyze temporal citation patterns.

Keywords: Temporal Graph; Temporal Graph Data Model; Graph Analysis

1 Introduction

The flexible analysis of graph data has gained significant interest in the last decade and is supported by graph database systems (e.g., Neo4j) and a growing number of distributed graph processing systems [Ju17]. While graphs typically evolve continuously, graph processing systems mostly focus on the analysis of static graphs representing the state (or snapshot) of a graph at a specific point in time. Changes like the addition of vertices and edges can occur comparatively slowly (e.g., in bibliographic networks) or at high frequency (e.g., as a stream of posts in a social network). An important requirement in many domains is to analyze the temporal dimension of graphs, e.g., to analyze the evolution of certain relationships like the citation patterns of publications or the development of co-authorships in bibliographic networks. For streaming-like changes there are specific analysis requirements, in particular to support fast, real-time reaction to certain changes such as the spread of hate messages in social networks.

In this short paper, we report on work in progress on temporal graph analysis using Gradoop [Ju16, Ju18], a distributed, open source framework for graph analytics. It supports extended property graphs as well as many declarative operators, e.g., for pattern matching and structural grouping, that can be used to realize complex workflows for graph analysis. Inspired by the temporal extensions in SQL:2011 [KM12] we extend the Gradoop graph data model by time properties for valid and transactional time. We also show how these temporal properties can be used within the operators for temporal graph analysis. Furthermore, we sketch the use of these operators for a bibliographic use case to find certain

[1] University of Leipzig, rost@informatik.uni-leipzig.de
[2] Leipzig University of Telecommunications, thor@hft-leipzig.de
[3] University of Leipzig, rahm@informatik.uni-leipzig.de

doi:10.18420/btw2019-ws-11

temporal citation patterns. After a discussion of related work we introduce the temporal extensions of Gradoop's property graphs (Sect. 3) and operators (Sect. 4). We then show the use of the operators in building blocks for common tasks in temporal graph analytics (Sect. 5) and outline the bibliographic use case in Section 6.

2 Related Work

Date et al. [DDL02] define three classes of time aspects for temporal relational databases that can be applied one-to-one to temporal graphs: *transaction time*, *valid time*, and their combination *bitemporal data*. *Transaction time* is defined as the time interval in which a fact is considered true in the database (graph). In most cases, the transaction time is maintained by the processing system itself and can be used for versioning so that graph states can be reconstructed for any point in the past. The *valid time* is defined as a time interval in which a fact is valid as defined by the context of the data [DDL02]. Valid time intervals can also be expressed by *time-stamps* if the duration can be neglected. Such graphs are also known as *transient* [KD13] or *contact sequences* [HS12] since their time-stamps reflect a chronological order of interactions (e.g., edge additions). Figure 1 shows an example of a temporal graph with valid times: The temporal affiliations of authors to institutions is represented as time intervals whereas the valid time of publications is the time-stamp when the publication was published.

There are only few graph processing systems that natively support the storage, analysis and querying of temporal graphs. *Immortalgraph* [Mi15] (earlier known as *Chronos*) provides a storage and execution engine designed for temporal graphs. It includes locality optimizations and an in-memory iterative graph computation based on series of graph snapshots. Snapshots are divided into groups to provide temporal graph mining approaches. *Kineograph* [Ch12] is a distributed platform for incoming stream data to construct a continuously changing graph. It is also based on in-memory graph snapshots which are evaluated by conventional mining approaches of static graphs (e.g., community detection). The snapshot approach is used to distribute the graph on different systems. *GraphStream* [Pi08] is an open-source Java library focusing on the dynamics aspects of a graph. It provides a flexible way to build user-defined analyses upon a dynamic graph structure based on a stream of graph events. None of these systems is based on a property graph model to hold detailed contextual information of vertices and edges besides the structural information. Then et al. [Th17] developed an automatic algorithm transformation to avoid multiple executions of graph analytics algorithms on snapshots of a temporal graph to reduce their runtimes.

A fairly new temporal graph analytics library is *Tink* [Li18] that focuses on several temporal path problems and offers the calculation of measures like temporal betweenness and closeness. Similar to Gradoop, Tink is build on Apache Flink and employs the Property Graph Model. Temporal information is represented by time intervals at the edges. In contrast to Tink, Gradoop supports not only graphs but also logical graphs and graph collections.

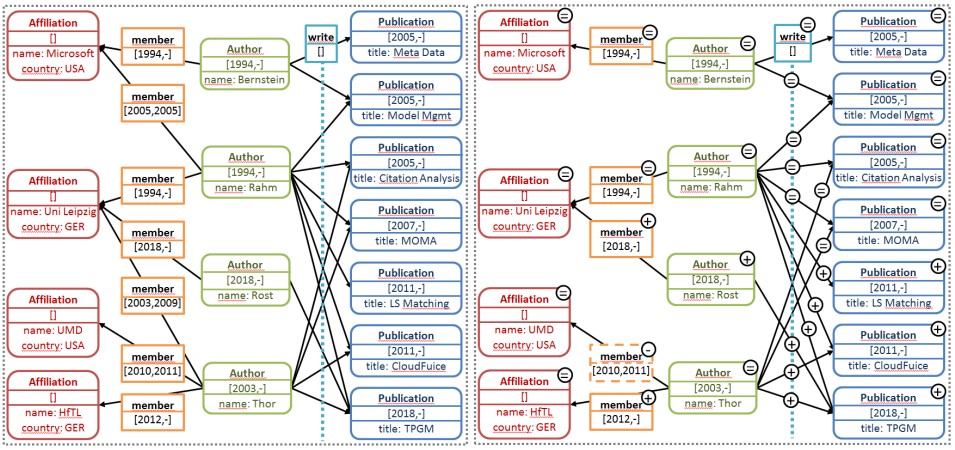


Fig. 1: TPGM example of a bibliographic graph (*left*) and the resulting annotated graph of the *Difference* operator of Sect. 4 (*right*). Vertices and edges are specified by label, valid-time (with format [*val-from,val-to*]) and properties.

Furthermore, our extension of the property graph model allows the definition of both time-stamps and intervals on both vertices and edges.

3 Temporal Property Graph Model in Gradoop

We introduce the *Temporal Property Graph Model* (TPGM) as a simple but powerful extension of the *Extended Property Graph Model* (EPGM) [Ju16] to support combinable analytical operators on directed graphs that evolve over time in Gradoop. In the EPGM, a single property graph is referred to as *logical graph*, which in turn can be part of a *graph collection*. Vertices and edges refer to one or more logical graphs and are accordingly part of them. Logical graphs, vertices, and edges consist of a unique identifier, a type label (e.g., *User* or *worksAt*), and a (possibly empty) set of properties represented as key-value pairs.

TPGM extends EPGM by adding four additional time attributes as obligatory to the schema of vertices, edges, and logical graphs: *tx-from*, *tx-to* and *val-from* and *val-to*. The first two represent the transaction time (prefix *tx*), the last two define the valid time (prefix *val*) by holding the beginning and end of the elements validity. This approach offers a flexible representation of temporal graphs with bitemporal time semantics where the valid time can be empty, a time-stamp or a time interval by setting either none, only the *val-from* or both *val-from* and *val-to* attributes. Since time attributes can be empty, also edge-centric scenarios where a graph only has time information at its edges can be modeled. Empty time attributes are interpreted as *NULL* values (e.g., in predicate functions) analogous to SQL. Fig. 1 (*left*) shows a temporal graph from the bibliographic domain modeled in TPGM. The graph represents the relationship between authors, affiliations and publications.

There are vertices and edges without a temporal specification (e.g., *Affiliation*), with a time-stamp (e.g., *Publication*) and a time interval (*member*) as valid-time. TPGM does not specify the data type of the time attributes and leaves it up to the implementation (e.g., Unix time-stamp or formatted date/time string). By holding a whole graph with both rollback and historical information, this model offers a flexible retrieval of arbitrary graph snapshots and dissociates itself from widespread snapshot approaches.

Valid times are typically embedded within the context of the data before they enter GRADOOP. The respective timestamps can be extracted while loading the elements as graph or collection into the system. The transaction times are maintained by the GRADOOP system automatically. Vertices and edges can be added to (or deleted from) a logical graph as well as a whole logical graph can be added to (or deleted from) a graph collection. For newly added graph elements the value of *tx-from* is set to the current system time (i.e., import time) and *tx-to* to infinity. If a graph element is deleted, the value of *tx-to* is set to the current system time. The deletion of a vertex automatically triggers the deletion of all corresponding edges, since they are not valid without the vertex. At the moment, the model does not support subsequent changes (updated) of an element's label, valid times or properties.

Another advantage of TPGM is its backward compatibility to the original EPGM since every EPGM operator can be applied to a TPGM graph by disregarding the temporal information of the graph elements. However, we will define appropriate operator extension for TPGM that allows the definition of temporal graph analytical workflows in the next section. Although TPGM offers bitemporal support, we limit ourselves exclusively to the valid-times for simplicity in the following sections.

4 Operators

The EPGM allows for combining multiple operators to graph analytical workflows using the domain specific language GrALa (**Gr**aph **A**nalytical **La**nguage). It already provides operator implementations for graph pattern matching, subgraph extraction, graph transformation, set operations on multiple graphs as well as property-based aggregation and selection. Some operators can be applied on logical graphs and others on graph collections [Ju16]. For simplicity we use the terms *graph* and *logical graph*, *collection* and *graph collection* interchangeably. In this work we will focus on the six graph operators that are listed in Tab. 1. They support the access and modification of the available temporal information in different ways. The following sections provide short definitions of these temporal operators with some examples. Pre-defined predicate functions that can be used by these operators are defined in Tab. 2. Furthermore, helper functions that return parts of a date or time information (e.g., year or day of week) are available. The implementation of the operators and their integration into GRADOOP is currently work in progress. Since we focus on valid-times in this section, each notation of *from* and *to* refers to the attributes *val-from* and *val-to* defined in TPGM.

Operator	Signature	Output
Transformation*	`Graph.transform(graphFunction, vertexFunction, edgeFunction)`	`Graph`
Subgraph*	`Graph.subgraph(vertexPredicateFunction, edgePredicateFunction)`	`Graph`
Snapshot	`Graph.snapshot(temporalPredicateFunction)`	`Graph`
Difference	`Graph.diff(temporalPredicateFunction, temporalPredicateFunction)`	`Graph`
Grouping*	`Graph.groupBy(vertexGroupingKeys, vertexAggregateFunction, edgeGroupingKeys, edgeAggregateFunction)`	`Graph, Collection`
Pattern Matching*	`Graph.query(patternGraph [,constructionPattern])`	`Collection`

Tab. 1: Overview of TPGM unary operators and their signature. Operators marked with * already exist in EPGM and were extended by temporal support.

Transformation. The *transform* operator defines a structure-preserving modification of graph, vertex and edge data. User-defined transformation functions can be applied to an input graph G, which results in an output graph G' [Ju16]. Within TPGM it is possible to (1) modify the temporal attributes, (2) define the time attributes from information stored in properties or (3) create properties resulting from the temporal information of the time attributes. For example, if the temporal attributes are not yet set or calculated during a workflow, this operator offers the possibility to define the valid times *from* and *to* at runtime.

Subgraph. The *subgraph* operator is used to extract a subgraph from a graph by applying predefined or user-defined predicate functions [Ju16]. Often, such a function is used to filter vertices and edges by label or the existence or value of a property (e.g., vertices with label *User* and property *age* greater than *30*). Within TPGM, the temporal information of graph elements can be used inside the predicate functions. The operator is also suitable to declare vertex-induced or edge-induced subgraphs by providing either a vertex *or* edge predicate function. Considering the graph in Fig. 1 (*left*), a subgraph with all author-affiliation memberships that last longer that 3 years can be extracted with the edge-induced operator call `graph.subgraph(null, e -> e.label = 'member' AND YEAR(e.to)-YEAR(e.from) > 3)`.

Snapshot. The *snapshot* operator allows one to retrieve a valid snapshot of the whole temporal graph either at a specific point in time or a subgraph that is valid during a given time range by providing a temporal predicate function. Besides the operator itself, several predefined predicate functions (see Tab. 2) are available. They are adopted from SQL:2011 [KM12] that supports temporal databases. In the example of Fig. 1 (*left*), `graph.snapshot(asOf(2010))` would remove the author named *Rost* and the last three publications together with their edges. Furthermore, three member edges (*Rahm-Microsoft*, *Thor-Uni Leipzig*, and *Thor-HfTL*) are removed.

Difference. The evolution of graphs over time can be represented by the difference of two graph snapshots, i.e., by a difference graph that is the union of both snapshots where each graph element is annotated as an added, deleted, or persistent element. To this end, Gradoop's structural `diff` operator consumes two graph snapshots defined by temporal predicate functions and calculates the difference graph. For example, the usage of `graph.diff(asOf(2010),asOf(2018))` at the graph in Fig. 1 (*left*) would result in the

Function	Predicate	
	Time-Stamp (only *from* defined)	Time-Interval (*from* and *to* defined)
asOf(x)	$from \leq x$	$from \leq x \wedge to \geq x$
fromTo(x, y)	-	$from < y \wedge to > x$
between(x, y)	-	$from \leq y \wedge to > x$
precedes(c)	$from \leq c.from$	$to \leq c.from$
succeeds(c)	$from \geq c.from$	$from \geq c.to$
overlaps(c)	-	$max(from, c.from) < min(to, c.to)$

Tab. 2: Predefined TPGM predicate functions that can be used by the operators. Variables *x* and y are timestamps, whereas *c* is a graph element. The predicates differ according to the definition of a time-stamp or a time-interval.

annotated graph at Fig. 1 (*right*). The symbols +, - and = represent added, removed and persisting elements, respectively.

Grouping. A structural grouping of vertices and edges is an important task in temporal graph analytics. Since temporal graphs can become very large, a condensation can facilitate deeper insights about structures and patterns hidden in the graph. In the current EPGM implementation of the *groupBy* operator, a grouping is based on vertex and edge grouping keys (e.g., the type label or property keys) as well as vertex and edge aggregation functions [JPR17]. For temporal grouping, TPGM provides three additional features: First, time-specific value transformation functions (e.g., year or day of week) can be applied to compute time values on the desired granularity for grouping. Second, the *groupBy* operator supports `GROUP BY CUBE` and `GROUP BY ROLLUP` similar to SQL. Third, aggregation on the temporal properties *from* and *to* of the vertices and edges can not only be specified by user-defined functions but by one of the predefined time-specific aggregation functions (e.g., *MinFrom* or *MaxFrom*). For example, the *AvgDuration* aggregate function can be used to determine the average duration of all membership edges at the graph in Fig. 1 (*left*).

Pattern Matching. Retrieving subgraphs matching a user-defined pattern graph is an important task within the graph analytics domain. In the EPGM a pattern matching operator *query* is already implemented [Ju16] using basic concepts of Neo4j Cypher[4] to define patterns, e.g., `(a)-[b]->(c)`. Predicate functions can be embedded in a pattern inside a `WHERE` clause by using variables defined in the pattern. The resulting embeddings can be modified by providing a construction pattern. In TPGM, we extend this functionality by using the temporal attributes *from* and *to* inside predicate definitions, i.e., by pre-defined (see Tab. 2) and user-defined predicate functions. For example, the pattern `(a:Author)-[m:member]->(f:Affiliation country : USA) WHERE m.asOf(2017)` describes an author being a member of an affiliation from the United States as of 2017.

[4] `https://neo4j.com/developer/cypher-query-language/`

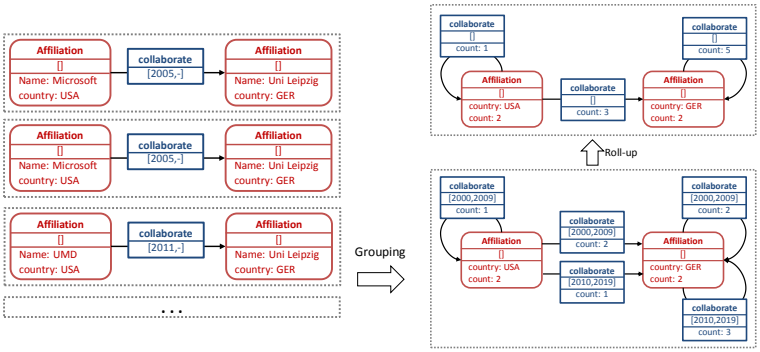


Fig. 2: Part of a graph collection containing embeddings of a temporal pattern (*left*) and the result of a time-specific grouping and aggregation (*right*) applied on these matches.

5 Temporal Graph Analytics Workflows

In this section we discuss exemplary workflows for temporal graph analytics. We illustrate how they can be supported by Gradoop and its extension to TPGM.

Snapshot generation and graph evolution: Graph systems or algorithms might focus on the analysis of static graphs representing the state (or snapshot) of a graph at a specific point in time. Gradoop therefore supports the retrieval of snapshots using the `snapshot` operator in combination with time-based predicates. To identify the difference and thus the changes between two graph snapshots, the `diff` operator can be used.

Temporal pattern matching: Searching for graph patterns using time-constraints is important for temporal analysis. In the example of Fig. 1 (*left*), a query to obtain simultaneous collaborations between affiliations from different countries must take the valid times of the *member* edges into account: `(f1:Affiliation)<-[m1:member]-(a1:Author)-[:write]->(p:Pub)<-[:write]-(a2:Author)-[m2:member]->(f2:Affiliation) WHERE a1 != a2 AND m1.overlap(p) AND m2.overlap(p)`. By applying this query together with the construction pattern `(f1)-[c:collaborate{from=p.from}]->(f2)` to Gradoop's *query* operator, a collection of matching graph embeddings is generated which is exemplified in Fig. 2 (*left*). A special case of such patterns are time-respecting paths, i.e., sequences of edges with non-decreasing times that connect vertices. Time-respecting paths not only identify all vertices that can be reached from others within some observation window [HS12] (reachability) but might also define shortest paths in terms of the overall duration time of the edges.

Time-specific grouping and aggregation: The time dimension automatically introduces a hierarchy, i.e., graphs can be grouped (summarized) at multiple levels of time-granularity. For example, the graph in the bottom-right corner of Fig. 2 summarizes the collaboration between countries per decade based on co-authored publications, i.e., the *Affiliation* vertices

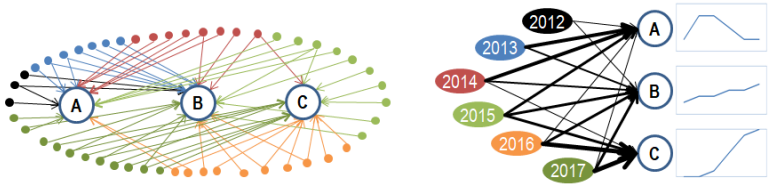


Fig. 3: Example of a citation graph (*left*) and its graph summary (*right*) for three publications *A*, *B*, and *C* that were all published in 2011. For simplicity, only citations to the three publications *A*, *B*, and *C* are illustrated, i.e., citations between the citing publications are omitted. In the graph summary, citing publications are grouped by publication year (indicated by color) and the number of citations corresponds to the line widths of the super edges. The small charts illustrate the citation dynamics: *A* follows a *common life cycle*, *B* is a *constant performer*, and *C* is a *sleeping beauty*.

are grouped by their *country* property and the *collaborate* edges are aggregated at the level of decades to reflect the temporal changes in the collaboration over time. However, this summarization can be rolled-up on the time hierarchy to have a global aggregation. To this end, Gradoop's `groupBy` operator groups all *collaborate* edges by year with `GROUP BY ROLLUP` so that the resulting graph collection contains both graphs of Fig. 2 (*right*).

Efficient maintenance and re-use of queries and analysis results: Repeated execution of Gradoop workflows, e.g., at certain times (e.g., once a month) or at certain events (e.g., when adding a new publication into a bibliographic network) requires an efficient update of graph transformations and graph summaries both in their structure and aggregated property values. In the above mentioned `GROUP BY ROLLUP` example, newly added *publication* vertices might only update the *collaborate* edges where the *publication* valid time falls into the valid time of the *collaborate* edge.

6 Use Case: Citation Analysis

The influence of past literature on current research is manifested by references cited in publications. Bibliographic networks therefore consist of publications (vertices) and their citations (directed edges) between them. Edges are time-stamped with the publication date of the citing publication to indicate when the citation happened. Fig. 3 shows an example with three publications *A*, *B*, and *C* (all published in 2011) that have been cited by 63 other publications. Besides the question of identifying exceptionally highly-cited publications ("top publications") it is also of interest to identify the citation dynamic of cited publications that usually follow a common *life cycle*: starting with low citations in the first or two years of publication, growing up to a maximum of citations a few years later, followed by a continuous decrease of citations several years after publication. However, other dynamics are also possible: a more or less long period of non-recognition with low citations is followed by a period with high citations after a sudden peak. Such publications are often referred to as *sleeping beauties* [vR04], i.e., they remained undetected over many years before their

results, methods, ideas etc. become important for current research. In contrast, *constant performers* are characterized by citation rates which are constantly at least on the average level compared to the other publications.

In the example of Fig. 3 the citation dynamics becomes visible after grouping the citing publications by the year of publication, i.e., the summary contains six super vertices (2012-2017) where each super vertex represents all vertices (i.e., citing publications) of the corresponding year. Super edges between the super vertices and the three original nodes (*A*, *B*, and *C*) store the number of elements they represent, i.e., the number of citations. For example, the super edge *2017* $\rightarrow$ *C* represents eight edges, i.e., *C* has been cited by eight publications in 2017. In Fig. 3 (right) the number of citations corresponds to the line widths of the super edges. The graph summary reveals the citation dynamics of the cited publications. For example, publication *C* did not get any citations in 2012 and 2013, few citations in 2014 and 2015 and many in 2016 and 2017. It can therefore be classified as a *sleeping beauty*. The described use case can be expressed in GrALa as follows:

```
// subgraph including edges of last six years
sub = in.subgraph(TRUE, e => YEAR(CURRENT)-YEAR(e.from) BETWEEN 1 AND 6 )
// group citation edges by year; count number of citations
group = sub.groupBy(
   ['citingpub', (citingpub.from, t => YEAR(t))],
   (superVertex, vertices => superVertex['year'] = vertices.min(YEAR(from)),
   [:label, (from, t => YEAR(t))],
   (superEdge, edges => superEdge['count'] = edges.count()))
// call external function 'classifier' (UDF) to specify citation dynamics
group.transform(
  (gIn, gOut => gIn),
  (vIn, vOut => IF (vIn.label=='citedpub') vOut['dynamic'] = classifier(vIn)),
  (eIn, eOut => eIn))
```

7 Conclusion

We reported our work in progress on temporal graph analysis by integrating our flexible temporal property graph model TPGM that supports bitemporal time semantics into the distributed graph analytic system Gradoop. By extending the available operators, we show how the evolving nature of temporal graphs can be used to answer time-respecting analytical questions. We illustrated the use of these operators within common building blocks of analysis workflow and provided a real-world use case from the bibliographic domain to find temporal citation patterns.

Temporal graphs and their analysis is an important and promising field of research. In future work we will further extend Gradoop by temporal features such as operators and algorithms to make Gradoop a powerful and flexible system for temporal graph analysis.

Acknowledgements. This work is partially funded by Sächsische Aufbau Bank (SAB) and the European Regional Development (EFRE) under grant No. 100302179.

References

[Ch12] Cheng, R., et al.: Kineograph: taking the pulse of a fast-changing and connected world. In: Proc. EuroSys. pp. 85–98, 2012.

[DDL02] Date, C. J.; Darwen, H.; Lorentzos, N.: Temporal data & the relational model. Elsevier, 2002.

[HS12] Holme, P.; Saramäki, J.: Temporal networks. Physics Reports, 519(3):97 – 125, 2012. Temporal Networks.

[JPR17] Junghanns, M.; Petermann, A.; Rahm, E.: Distributed grouping of property graphs with GRADOOP. Proc. BTW 2017), 2017.

[Ju16] Junghanns, M.; Petermann, A.; Teichmann, N.; Gómez, K.; Rahm, E.: Analyzing extended property graphs with Apache Flink. In: Proc. SIGMOD Workshop on Network Data Analytics. 2016.

[Ju17] Junghanns, M.; Petermann, A.; Neumann, M.; Rahm, E.: Management and analysis of big graph data: current systems and open challenges. In: Handbook of Big Data Technologies, pp. 457–505. Springer, 2017.

[Ju18] Junghanns, M.; Kießling, M.; Teichmann, N.; Gómez, K.; Petermann, A.; Rahm, E.: Declarative and distributed graph analytics with GRADOOP. PVLDB, 11(12), 2018.

[KD13] Khurana, U.; Deshpande, A.: Efficient snapshot retrieval over historical graph data. In: Proc. ICDE. pp. 997–1008, 2013.

[KM12] Kulkarni, K.; Michels, J.: Temporal features in SQL: 2011. ACM Sigmod Record, 41(3):34–43, 2012.

[Li18] Ligtenberg, W.; Y.Pei; Fletcher, G.H.L.; Pechenizkiy, M.: Tink: A Temporal Graph Analytics Library for Apache Flink. In: WWW (Companion Volume). ACM, pp. 71–72, 2018.

[Mi15] Miao, Y., et. al.: Immortalgraph: A system for storage and analysis of temporal graphs. ACM Transactions on Storage, 11(3):14, 2015.

[Pi08] Pigné, Y.; Dutot, A.; Guinand, F.; Olivier, D.: GraphStream: A Tool for bridging the gap between Complex Systems and Dynamic Graphs. CoRR, abs/0803.2093, 2008.

[Th17] Then, M.; Kersten, T.; Günnemann, S.; Kemper, A.; Neumann, T.: Automatic Algorithm Transformation for Efficient Multi-Snapshot Analytics on Temporal Graphs. PVLDB, 10(8):877–888, 2017.

[vR04] van Raan, Anthony F. J.: Sleeping Beauties in science. Scientometrics, 59(3):467–472, Mar 2004.

Angepasstes Item Set Mining zur gezielten Steuerung von Bauteilen in der Serienfertigung von Fahrzeugen

Marco Spieß[1] und Peter Reimann[2]

Abstract: Qualitätsprobleme im Bereich Fahrzeugbau können nicht nur zum Imageverlust des Unternehmens führen, sondern auch mit entsprechend hohen Kosten einhergehen. Wird ein Bauteil als Verursacher eines Qualitätsproblems identifiziert, muss dessen Verbau gestoppt werden. Mit einer Datenanalyse kann herausgefunden werden, welche Fahrzeugkonfigurationen Probleme mit diesem fehlerverursachenden Bauteil haben. Im Rahmen der domänenspezifischen Problemstellung wird in diesem Beitrag die Anwendbarkeit von Standardalgorithmen aus dem Bereich Data-Mining untersucht. Da die Analyseergebnisse auf Standardausstattungen hinweisen, sind diese nicht zielführend. Für dieses Businessproblem von Fahrzeugherstellern haben wir einen Data-Mining Algorithmus entwickelt, der das Vorgehen des Item Set Mining der Assoziationsanalyse an das domänenspezifische Problem anpasst. Er unterscheidet sich zum klassischen Apriori-Algorithmus in der Beschneidung des Ergebnisraumes sowie in der nachfolgenden Aufbereitung und Verwendungsweise der Item Sets. Der Algorithmus ist allgemeingültig für alle Fahrzeughersteller anwendbar. Die Ergebnisse sind anhand eines realen Anwendungsfalls evaluiert worden, bei dem durch die Anwendung unseres Algorithmus 87% der Feldausfälle verhindert werden können.

Keywords: Big Data Analytics, Data-Mining, Item Set Mining.

1 Einführung

Garantiekosten im Rahmen der Produktgewährleistungspflicht können das Betriebsergebnis eines produzierenden Unternehmens negativ beeinflussen [Ve00]. Im Geschäftsjahr 2017 ist von den Automobilherstellern weltweit eine Summe von 53,37 Mrd. USD an Gewährleistungskosten gezahlt worden [Wo18]. Demnach nehmen das Qualitätsmanagement sowie die Kundenbetreuung in der Produktnutzungsphase einen hohen Stellenwert im Unternehmen ein, um Qualitätsprobleme möglichst frühzeitig zu identifizieren und zu beheben [Ve00, BW16].

Dem in diesem Beitrag behandelten Anwendungsfall liegt ein Qualitätsproblem im Bereich Getriebe von Fahrzeugen zugrunde. Dabei wurde die Kupplung durch verschiedene Messungen als Fehlerursache identifiziert, weshalb diese daraufhin konstruktiv angepasst wurde.

[1] Universität Stuttgart, Graduate School of Excellence advanced Manufacturing Engineering (GSaME), Nobelstr. 12, 70569 Stuttgart, Marco.Spiess@gsame.uni-stuttgart.de
[2] Universität Stuttgart, Graduate School of Excellence advanced Manufacturing Engineering (GSaME), Nobelstr. 12, 70569 Stuttgart, Peter.Reimann@gsame.uni-stuttgart.de

doi:10.18420/btw2019-ws-12

Diese angepasste Kupplung soll nun bereits in der Produktion in die Fahrzeuge eingebaut werden, um zukünftige Feldausfälle zu verhindern. Durch vereinbarte Liefermengen kann die neue Kupplungsversion aber nur in jedes zweite Fahrzeug eingebaut werden. Die restlichen Fahrzeuge werden mit einer anderen Kupplung ausgestattet, welche zum Zeitpunkt der Datenanalyse als OK eingestuft wurde. Das Ziel der durchzuführenden Analyse ist die Identifizierung von Fahrzeugen, bei denen die neue angepasste Kupplungsversion eingebaut werden soll, da die andere Kupplung bei diesen Fahrzeugen sonst weiter Probleme verursachen könnte. Demnach soll mittels der Analyseergebnisse eine Steuerung von geeigneten Bauteilen in die individuellen Fahrzeugkonfigurationen erfolgen. Da Fahrzeuge, die zunächst auf der Montagelinie montiert werden, keine Daten über die tatsächliche Nutzung im Kundenumfeld enthalten, reduziert sich die Auswahl der relevanten Datenquellen für die Analyse auf die Konfigurationsdaten von bereits produzierten und sich in Nutzung befindlichen Fahrzeugen. Durch diese Daten wird ein zu produzierendes Fahrzeug in seiner Zusammenstellung beschrieben, z.B. welcher Motor, Getriebe etc. verbaut sind oder ob das Fahrzeug eine Sonderausstattung hat. Die ausfallgefährdeten Fahrzeuge werden durch Kombinationen der ermittelten Konfigurationsmerkmale erkannt und mit der angepassten Kupplung ausgestattet.

Folgende Einzelbeiträge werden in diesem Beitrag geleistet:

1. Es werden Untersuchungen mit alternativen Data-Mining Techniken auf deren Eignung für die Problemstellung durchgeführt und bewertet. Dabei stellt sich heraus, dass die Anwendung von Klassifikation sowie Clusteranalyse keinen Mehrwert generieren, da sie Standardausstattungen eines Fahrzeugs als Ergebnis erzeugen, die nicht zielführend sind. Die Assoziationsanalyse hingegen ist mit ihrer Eigenschaft, kombinierte Elemente (Item Sets) zu betrachten, am geeignetsten. Dennoch kann mit ihr alleine die Zielstellung nicht erreicht werden, da ausfallgefährdete Fahrzeuge nicht durch einzelne Item Sets, sondern durch die Kombination mehrerer Sets beschrieben werden [HKP12].
2. Im Sinne der Problemstellung schlagen wir einen angepassten Algorithmus vor, um ausfallgefährdete Fahrzeugkonfigurationen aus den Garantiedaten zu identifizieren. Die mit gleichen Konfigurationen zu produzierenden Fahrzeuge werden gezielt mit einem geeigneten Bauteil ausgestattet. Hierfür haben wir das Item Set Mining des Apriori Algorithmus angepasst, damit das Analyseergebnis in der Produktionssteuerung der Serienfertigung genutzt werden kann.
3. Der vorgestellte Algorithmus ist anhand eines realen Anwendungsfalls im Bereich der Produktion von Fahrzeugen evaluiert und als geeignet bewertet worden. Dabei konnten wir Konfigurationsmerkmale identifizieren, mit deren Anwendung 87% der Feldausfälle hätten verhindert werden können.

Dieser Beitrag gliedert sich in fünf Abschnitte. In Abschnitt 2 stellen wir den Anwendungsfall und die relevanten Daten vor. Abschnitt 3 stellt die Untersuchungsergebnisse aus der

Anwendung alternativer Data-Mining Techniken dar. In Abschnitt 4 stellen wir die Grundidee des vorzuschlagenden Algorithmus sowie das damit erzielte Ergebnis vor. Dieser Beitrag endet mit der Zusammenfassung und zukünftiger Arbeiten in Abschnitt 5.

2 Anwendungsfall

Der größte Anteil an Garantiekosten ergibt sich durch das vermehrte Auftreten von Qualitätsproblemen beim Kunden im Produktgewährleistungszeitraum. Im Kontext des Qualitätsmanagement werden ausgebaute Schadteile aus dem Feld gezielt in die Befundstellen der Original-Equipment-Manufacturer (OEM) eingesteuert, um sie einer Schadteilanalyse zu überführen [Ve09]. Unter einem Schadteil werden jene Teile verstanden, die im Zuge einer Reparatur ausgebaut und ersetzt werden. Im Kontext der Schadteilanalyse gilt es, durch Methoden aus den Ingenieurwissenschaften wie bspw. Six Sigma oder 8D diejenigen Schadteile zu identifizieren, die eine mögliche Fehlerursache für ein Qualitätsproblem darstellen [JSW17]. Der in diesem Beitrag behandelte Anwendungsfall beschreibt das Problem eines OEM, der die Kupplung als Fehlerursache identifiziert und diese in der Konstruktion angepasst hat. Diese Anpassung muss nun in der Produktionslinie nachvollzogen werden. Die Prämisse hierbei ist die limitierte Teileversorgung durch den Lieferanten, welcher lediglich 50% der Produktion mit angepassten Kupplungen versorgen kann. Durch die Prämisse der Teileversorgung soll die angepasste Kupplung gezielt in ausfallgefährdete Fahrzeuge bereits in der Produktion eingebaut werden. Dabei definieren wir ein Fahrzeug dann als ausfallgefährdet, wenn seine Konfiguration nahezu deckungsgleich ist mit den Konfigurationen bereits ausgefallener Fahrzeuge. Diese Definition ist aus dem behandelten Anwendungsfall abgeleitet, bei dem ein Fahrzeugausfall mitunter auf dessen technische Zusammenstellung zurückzuführen ist. Demnach hat die durchzuführende Datenanalyse das Ziel eine direkte Fehlerabstellung in der Produktion zu ermöglichen.

Fahrgestellnummer	Feldausfall	Konfiguration
XYZ4711L590	1	C1/C2/C3/C4/C5/C6/C7/C8/C9/C19/. . . /Cn
XYZ9718L590	0	C3/C6/C7/C8/C14/C34/C45/. . . /Cn
XYZ4707R790	1	C2/C6/C7/C10//C29/C40/. . . /Cn
...	...	...

Tab. 1: Exemplarische Datensätze der zu analysierenden Daten

Tab.1 zeigt einen beispielhaften Auszug der zu analysierenden Daten. Darin wird ein Fahrzeug eindeutig über dessen Fahrgestellnummer identifiziert. Ob ein Fahrzeug bereits einen Schaden im Feld bzgl. des betrachteten Qualitätsproblems hatte, wird über das Setzen eines Flags im Datenfeld Feldausfall erkenntlich. Des Weiteren sind im Datenfeld Konfiguration sämtliche Merkmale enthalten, die ein Fahrzeug bzgl. dessen verbauter Ausstattungen näher beschreiben. Dabei werden die Konfigurationsmerkmale als aneinandergereihte Codes der Form C1, C2, C3 usw. gespeichert. Dabei gibt es in unserem Anwendungsfall insgesamt 3.000 disjunkte Code-Elemente. Im Durchschnitt enthält ein einzelnes Fahrzeug in etwa 300

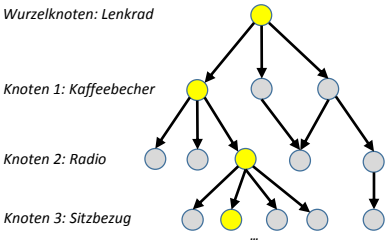
unterschiedliche Codes. Da hierunter auch Merkmale enthalten sind, welche in nahezu jedem Fahrzeug verbaut sind, wie ein Lenkrad, Reifen oder Radio, muss im Ablauf des Algorithmus ein Mechanismus etabliert werden, der diese Standardkonfigurationen erkennt. In unserem Anwendungsfall haben wir in der Analysemenge insgesamt 117.000 Fahrzeuge, wovon 2.000 bereits einen Feldausfall zum behandelten Qualitätsproblem aufweisen. Zusammengefasst ergeben sich aus der vorliegenden Datenstruktur zwei wesentliche datentechnische Herausforderungen für die Datenanalyse:

1. Standardcodes, also allgemeine Ausstattungen wie ein Lenkrad, die sehr viele Fahrzeuge aufweisen und die aber keinerlei Bezug zum Qualitätsproblem haben, können im Rahmen einer Mustererkennung nicht verwendet werden. Werden diese dennoch berücksichtigt, stehen diese in der Ergebnisliste ganz oben. Demnach müssen diejenigen Codes identifiziert werden, deren Kombination zum einen das Schadgeschehen aus den Daten möglichst vollständig abbildet und zum anderen die Menge an noch zu erwartender Feldausfälle möglichst präzise eingrenzt. Gleichzeitig soll die Gesamtmenge an produzierten Fahrzeugen mit diesen Codes maximal 50% betragen.
2. Innerhalb der Fahrzeuge kommt es bzgl. deren Ausstattungen zu einer Schnittmengenproblematik. Beispiel dafür ist Code C1, der in Kombination mit Code C2 in 150 ausgefallenen Fahrzeugen vertreten ist. Die Kombination C1 und C3 tritt hingegen in 100 reparierten Fahrzeugen auf. Da auch alle drei Codes C1, C2 und C3 in einzelnen Fahrzeugen auftreten können, resultiert aus dem Beispiel eine kumulierte Anzahl von 175 statt 250 aufsummierten Reparaturen. Die analytische Herausforderung besteht darin, diejenigen Kombinationen mit dem möglichst größten Anteil am Schadgeschehen zu entdecken, deren Anteile nicht in einer anderen Kombination bereits enthalten sind. Anhand des o.g. Beispiels sollte demnach analysiert werden, ob es weitere Kombinationen gibt, in denen die Codes C1, C2 und C3 enthalten sind und durch Hinzunahme weiterer Codes wie bspw. C4, präzisiert werden können.

3 Eignung alternativer Techniken aus dem Stand der Technik

In diesem Abschnitt werden die Untersuchungsergebnisse hinsichtlich der für die Problemstellung infrage kommenden Techniken aus dem Bereich des Data-Mining vorgestellt [HKP12]. Die Auswahl der aufgeführten Techniken ergibt sich aus der Struktur der zu analysierenden Daten sowie dem Analysezweck aus der Problemstellung. Da die Ausstattungsmerkmale eines Fahrzeugs bereits in einer warenkorbähnlichen Struktur vorliegen, kommt die Assoziationsanalyse als Miningverfahren in Betracht. Ausgehend von der analytischen Zielsetzung, relevante Konfigurationsmerkmale ausfallgefährdeter Fahrzeuge zu identifizieren, sind auch die Klassifikation und Clusteranalyse Gegenstand dieser Untersuchung. Die für die Datenanalysen genutzten Features sind bei allen Techniken jeweils die Konfigurationsdaten. Dies wird damit begründet, da das Analyseergebnis in

der Serienfertigung genutzt werden soll, in der zu einemproduzierenden Fahrzeug, außer der eindeutigen Fahrgestellnummer und dessen Ausstattung keine weiteren beschreibenden Daten vorhanden sind. Für die Untersuchungen haben wir insgesamt 6.327 Instanzen verwendet. Vor den Analysen haben wir versucht eine manuelle Datenfilterung der Codes durchzuführen indem vermeintliche Standardcodes wie Lenkrad eliminiert werden. Ebenso filterten wir weitere Sachverhalte wie Sitzbezüge und Felgen, bei denen keine technische Relevanz gegeben ist, manuell per Datenbankabfrage aus der Analysemenge heraus. Trotz dieser Datenfilterung können wir dennoch nicht sicherstellen, alle Standardcodes damit ausgeschlossen zu haben. Zumal der Anwender i.d.R. kein Wissen über diese besitzt, muss ein algorithmischer Automatismus entwickelt werden, der die Standardcodes von den relevanten Codes separiert. Unserer Einschätzung nach lässt sich damit die Analysequalität verbessern, denn im Allgemeinen stellt sich heraus, dass Standardcodes nicht eindeutig als solche erkannt werden können. Somit können sie nicht vor einer Analysedurchführung rausgefiltert werden, ohne das Analyseergebnis zu beeinflussen. Aus diesem Grund haben wir die Analysen mit den Codes im Original durchgeführt. Dabei haben wir herausgefunden, dass einzelne Codes in den Daten existieren, die aufgrund ihrer nominalen Bedeutung eine Standardausstattung sind. Dennoch sind sie wichtig, da sie besonders in den bereits ausgefallenen Fahrzeugen vorhanden sind.



Fahrgestellnummer	C1	C2	C3	C4	C5	C6	C7	C8	Cn
XYZ4711L590	1	0	0	1	0	1	1	0	...
XYZ9718L590	1	1	0	0	0	1	1	0	...
XYZ4707R790	1	1	1	1	0	0	0	0	...
XYZ4727L595	1	0	1	0	1	0	0	0	...
XYZ9736L585	1	1	1	0	0	1	1	1	...
...	...	...	...	...	...	...	...	...	...

Abb. 1: Klassifikation als Entscheidungsbaum

Im nachfolgenden Teil dieses Abschnitts gehen wir lediglich auf die Assoziationsanalyse detailliert ein. Diese Data-Mining Technik, bzw. der darin vorgelagerte Analyseschritt des Item Set Mining, hat sich im Rahmen unserer Untersuchung als am geeignetsten für die Problemstellung herausgestellt. Für diese Feststellung haben wir die Anwendung einer hierarchischen Clusteranalyse sowie einer Klassifikation als Entscheidungsbaum für die Problemstellung untersucht. Beide Verfahren erzeugen anhand der nominalen Konfigurationsdaten eine Baumstruktur, vereinfacht in Abb. 1 dargestellt, in der jeder Pfad einzeln durchlaufen werden kann. Der stattfindende Test in den Knoten basiert auf der Entscheidung, ob ein Code vorhanden ist oder nicht. Die dafür notwendige Datenstruktur ist in Abb. 1 rechts dargestellt. In diesen Pfaden ist in den oberen Verzweigungen eine signifikant hohe Anzahl von Standardcodes vertreten, die durch die Einbettung in die Baumstruktur, nicht herausgelöst werden können. Durch den o.g. Punkt, dass Standardcodes nicht als solche erkannt werden können, bewerten wir die beiden Data-Mining Techniken bzgl. dieser Problematik als ungeeignet. Diese Bewertung begründet sich darin, dass die Relevanz einzelner Codes nicht ersichtlich ist. Der in Abb. 1 gelb markierte Pfad stellt

dabei eine Konfigurationsausprägung dar. Die in der Abbildung aufgeführten Knoten sind deren jeweiligen Tests. Durch die vorhandene Struktur können Codekombinationen nicht wie beim Item Set Mining unabhängig voneinander untersucht werden, sondern es können nur diejenigen Kombinationen betrachtet werden, die einen Pfad durch den Baum beschreiben. Hingegen werden durch die Baumstruktur jegliche Überschneidungen von Codes mehrerer Konfigurationsausprägungen und daher mehrerer Pfade im Baum vermieden. Aus diesem Grund bewerten wir die Clusteranalyse und Klassifikation in der Schnittmengenproblematik als bedingt geeignet. Nur bedingt deshalb, weil es dafür erforderlich ist, alle Pfade des Baumes von Anfang bis Ende zu durchlaufen. Aufgrund der Vielfalt an möglichen Fahrzeugvarianten beträgt die Anzahl an Ästen nahezu der Anzahl an Instanzen, was nicht zielführend ist, um relevante Codes zu identifizieren.

3.1 Assoziationsanalyse

Im Kontext der Assoziationsanalyse haben wir uns für den Apriori-Algorithmus entschieden, da er am bekanntesten ist und wir keine Performancevorgaben haben [AS94].

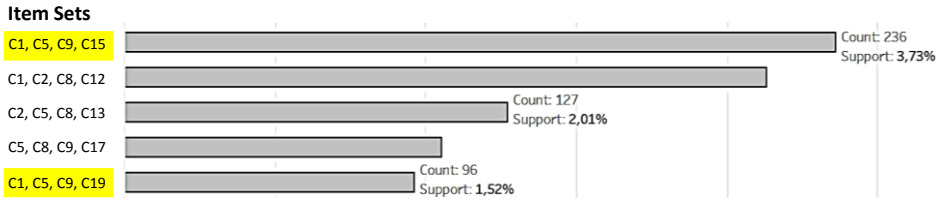


Abb. 2: Ergebnisauszug aus dem Item Set Mining

In Abb. 2 haben wir einen Auszug des visualisierten Ergebnisses aus dem Item Set Mining mit den o.g. 6.327 verwendeten Instanzen aufgeführt. Die erste gelb markierte Kombination ist von ihrer Anzahl dahinterstehender Fahrzeuge am höchsten. Da Abb. 2 das Ergebnis mit allen Instanzen darstellt und nicht zwischen bereits ausgefallenen Fahrzeugen und bisher nicht auffälligen Fahrzeugen unterscheidet, haben wir diese Codekombination gegen unsere Datenbank abgefragt. Dabei stellt sich heraus, dass die Kombination C1, C5, C9, C15 zwar den größten Anteil in den Daten bildet, dennoch sind von diesen 236 Fahrzeugen lediglich 17 bereits im Feld ausgefallen. Bei der untersten Kombination C1, C5, C9, C19 haben wir hingegen ermittelt, dass von diesen 96 Fahrzeugen, 72 davon bereits einen Feldausfall haben. Um die Relevanz aller Codekombinationen bzgl. Feldausfällen zu ermitteln, sind für das Item Set Mining zwei Analysedurchgänge erforderlich und zwar einmal mit den Fahrzeugen mit Feldausfällen und einmal mit den Fahrzeugen ohne Feldausfall. Anhand des Beispiels stellen wir fest, dass wir nur einzelne Codekombinationen als relevant identifizieren können, die lediglich einen Anteil am Schadgeschehen ausmachen. Um eine vollständige Abbildung der Feldausfälle zu erzielen, ist es erforderlich weitere Kombinationen zu betrachten. Dieser Punkt richtet sich an die Herausforderung der Schnittmengenproblematik, da dieKombination mehrerer Item Sets nicht zwangsläufig zu einem besseren Ergebnis führt.

Vielmehr ist es notwendig die daraus resultierende Abdeckung von Feldausfällen sowie die Anzahl aller mit diesen Kombinationen produzierten Fahrzeuge zu ermitteln. In Bezug auf die datentechnische Herausforderung der Standardcodes bewerten wir das Item Set Mining als bedingt geeignet, da die erzeugten Item Sets unabhängig voneinander betrachtet werden können. Im Gegensatz zu anderen Data-Mining Verfahren wie der Klassifikation entsteht hierbei keine Strukturierung, weshalb einzelne Items gezielt ignoriert werden können, ohne das Analyseergebnis weiterer Item Sets zu beeinflussen. Bei der Klassifikation hingegen wird damit die gebildete Struktur des Entscheidungsbaumes zerstört, wodurch die Ergebnisse obsolet werden. Dennoch ist für das Item Set Mining ein vorgelagerter Schritt zu entwickeln um die minimale Menge an einzelnen Codes zu bestimmen, mit denen das Schadgeschehen abgebildet werden kann. Um auf die Herausforderung der Schnittmengenproblematik einzugehen, betrachten wir dazu das erste und letzte Item Set in Abb. 2. Hierbei wollten wir wissen, wie viele Fahrzeuge es mit beiden Sets zusammen gibt. Das Resultat daraus sind 251 Fahrzeuge. Somit stellen wir fest, dass sich 81 Fahrzeuge durch die Vereinigungsmenge der beiden Vierer-Item Sets charakterisieren lassen. Daraus lässt sich schließen, dass die Codes, welche die beiden Item Sets unterscheiden, anteilig jeweils in beiden Item Sets enthalten sind. In diesem Beispiel sollte demnach ein Fünfer-Item Set gebildet werden, um die Schnittmengenproblematik der beiden Vierer-Item Sets aufzulösen. Daraus ergibt sich die Folgeproblematik, dass eine Entscheidung darüber getroffen werden muss, an welchem Punkt die Erzeugung von weiteren Item Sets gestoppt werden soll. Diese Entscheidung sollte durch testweises Kombinieren erzeugter Item Sets erfolgen. Dabei wird die eingangs erwähnte Zielgröße der Teileversorgung von maximal 50% als zu erreichende Abbruchbedingung herangezogen. Der Standard Apriori-Algorithmus kann diese Anforderung nicht erfüllen, weswegen wir das Item Set Mining bzgl. der Schnittmengenproblematik als ungeeignet bewerten. Ergänzend zum Item Set Mining haben wir die Ergebnisse aus der Erzeugung von Assoziationsregeln auf Basis der bereits berechneten Item Sets untersucht. Das Fazit dazu ist, dass die Regeln nicht den Analysezweck erfüllen, da mit diesen lediglich eine Wenn-Dann-Beziehung innerhalb der Codes aufgezeigt wird. Die Erzeugung der Item Sets als vorgelagerter Analyseschritt brachte hingegen mehr Erkenntnisse über die vorliegenden Daten. Aus den Sets konnten wir per Ranking einzelne Merkmale finden, welche nicht in Kombination mit Standardcodes ausgewiesen werden und gleichzeitig in einer signifikant hohen Menge an ausgefallenen Fahrzeugen enthalten sind.

3.2 Zusammenfassung der Untersuchungsergebnisse

Nachstehende Tab. 2 führt die Ergebnisse der untersuchten Standard Data-Mining Techniken auf deren Eignung bzgl. datentechnischer Herausforderungen auf. Dabei haben wir festgestellt, dass sich lediglich Teilaspekte der Vorgehensweisen bedingt für jeweils eine Herausforderung eignen. Die Conclusio der Untersuchung ist, dass sich das Item Set Mining für die Problemstellung grundsätzlich eignet, da es die erzeugten Item Sets unabhängig von einer notwendigen Struktur betrachtet. Somit können die von den Item Sets repräsentierten Teilmengen einzeln analysiert und bewertet werden. Aus den durchgeführten Untersuchun-

Datentechnische Herausforderung	Assoziationsanalyse (Item Set Mining)	Clusteranalyse (Hierarchisch)	Klassifikation (Entscheidungsbaum)
Standardcode-problematik	Bedingte Eignung	Ungeeignet	Ungeeignet
Schnittmengen-problematik	Ungeeignet	Bedingte Eignung	Bedingte Eignung

Tab. 2: Gegenüberstellung von Data-Mining-Techniken bzgl. analytischer Herausforderungen

gen ergeben sich Anforderungen, die im Kontext einer domänenspezifischen Anpassung des Standard Algorithmus erforderlich sind. (1) Vor der ersten Item Set Erzeugung soll algorithmisch geprüft werden, welche Codes überhaupt notwendig sind. Damit beschneiden wir den Ergebnisraum der Item Set Erzeugung, um somit minimale Item Sets zu identifizieren, mit denen wir trotzdem eine nahezu 100% Abdeckung ausgefallener Fahrzeuge erhalten. (2) Nach jeder Erzeugung von Item Sets, soll getestet werden, ob die erzeugten Item Sets bereits ausreichen, um die Prämisse von maximal 50% produzierten Fahrzeugen mit diesen Codes zu erfüllen. Hierfür wird, ausgehend von einem Startelement geprüft, welche nachfolgenden Item Sets in das Ergebnisset geschrieben werden sollen. Zielsetzung dabei ist, dass mit dem Hinzufügen eines Sets in die Ergebnisliste, die Abdeckung der Feldausfälle stetig steigt. Das Unterschreiten der maximalen Anzahl produzierter Fahrzeuge bildet dabei die Abbruchbedingung des Algorithmus.

4 Lösungsansatz und Ergebnisdiskussion

In diesem Abschnitt stellen wir den Grundgedanken des Algorithmus vor, den wir für die Problemstellung, unter Berücksichtigung der datentechnischen Herausforderungen, entwickelt haben. Um die relevanten Codes für eine nahezu 100% Abdeckung bereits ausgefallener Fahrzeuge zu identifizieren und dabei die Prämisse von 50% Teileversorgung einzuhalten, sind im Ablauf des Apriori-Algorithmus, die in Abschnitt 3.2 aufgeführten Anforderungen, notwendig. In Abb. 3 haben wir die Anpassungen am Apriori-Algorithmus exemplarisch dargestellt. Im Anwendungsbeispiel haben wir einen Input von 117.000 Instanzen, wovon 2.000 einen Feldausfall haben. Zu jeder Instanz sind die Konfigurationsdaten enthalten. Der zu unterschreitende Parameter beträgt 50%.

Der erste Schritt ist die Bestimmung der kleinstmöglichen Codemenge durch den Algorithmus A1, mit der 100% der Feldausfälle abgedeckt werden können, die gleichzeitig in einer minimalen Anzahl von produzierten Fahrzeugen enthalten sind. Hierfür testet A1 iterativ, beginnend bei den Codes mit den minimalsten Stückzahlen, ob sich bereits mit diesen das Schadgeschehen abbilden lässt. Mit jeder Iteration werden weitere Codes in die Betrachtung miteinbezogen, bis alle Fahrzeuge mit Feldausfällen abgebildet werden. Damit lassen wir algorithmisch bestimmen, welche einzelnen Codes tatsächlich benötigt werden, um aus

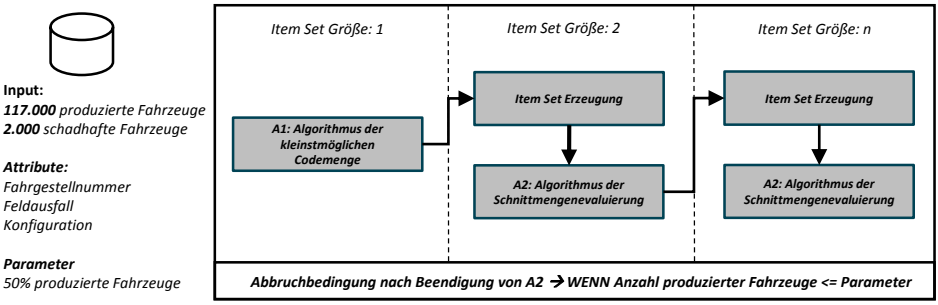


Abb. 3: Ablauf des vorgeschlagenen Algorithmus für das angepasste Item Set Mining

diesen anschließend Item Sets zu bilden. Somit löst A1 die Standardcodeproblematik und lässt sich zu Anforderung (1) einordnen.

In unserem Anwendungsbeispiel haben wir als Ergebnis von A1 25 disjunkte Codes über alle Instanzen hinweg erhalten. Die anschließende Item Set Erzeugung erfolgt nach dem Standardverfahren, jedoch werden nur Item Sets erzeugt, zu denen mindestens ein Fahrzeug mit Feldausfall existiert. Der darauffolgende Algorithmus A2 evaluiert die Item Sets bzgl. deren Schnittmengen und kumuliert in einem iterativen Prüfverfahren, welche Item Sets miteinander kombiniert werden müssen. Ziel dabei ist es, eine Abfolge von Item Set Kombinationen zu entdecken, die zusammen eine möglichst vollständige Abdeckung der Feldausfälle erreichen. Ebenso wird in A2 die von dieser Abfolge betroffene Fahrzeuganzahl berechnet. Ist die Anzahl größer als der gewählte Parameter, erfolgt erneut die Erzeugung von Item Sets der nächsten Größe. Folglich lässt sich durch A2 die Schnittmengenproblematik lösen und zur Anforderung (2) einordnen, indem die Fahrzeuganzahlen aus den Vereinigungsmengen der erzeugten Sets berechnet werden.

Item Sets	Kumulierte Feldausfälle	Kumuliertes Produktions-volumen	Kum. Anteil an Feldausfällen	Kum. Anteil am Produktionsvolumen
C1, C6, C7	1750	57.500	87,50%	49,15%
C1, C8, C12	1400	42.000	70,00%	35,90%
C4, C9, C13	850	25.000	42,50%	21,37%

Tab. 3: Ergebnisdarstellung aus dem vorgeschlagenen Algorithmus

Der Algorithmus endet mit dem Erreichen des initial gewählten Parameters. Das Ergebnis des Algorithmus ist somit eine Liste von Item Sets, die als Abfolge kombiniert werden, um einen möglichst hohen Anteil von Feldausfällen abzudecken. In Tab. 3 ist solch ein Ergebnisset dargestellt. In diesem Beispiel sind drei Item Sets identifiziert und deren kumulierten Anteile zu Feldausfällen und Stückzahlen absteigend sortiert worden. Diese Kombinationen können mittels OR-Operator kombiniert werden, um den Verbau von Bauteilen, in den damit zu identifizierenden Fahrzeugen, zu steuern. Für unseren Anwendungsfall konnten

wir durch die Anwendung des Algorithmus, der prototypisch in T-SQL implementiert ist, eine für die Produktionssteuerung geeignete Abfolge von Codekombinationen generieren [MNK15]. Diese haben wir Anfang Juli 2018 mit der o.g. Anzahl Instanzen erzeugt und mit den Feldausfällen, die bis zum Datenstand 30.11.2018 aufgetreten sind, abgeglichen. Somit haben wir das Analyseergebnis mit den bis dahin neuen Daten als Validierungsdaten getestet. Dabei kam heraus, dass wir 87% der bisher nicht betrachteten Feldausfälle mit unserer Abfolge abdecken. Diese Abfolge wird ab dem 01.12.2018 zur gezielten Bauteilsteuerung in der Serienfertigung genutzt.

5 Zusammenfassung und Ausblick

Für die Problemstellung, ausfallgefährdete Fahrzeugkonfigurationen für den Verbau einer angepassten Kupplung zu identifizieren, konnten wir eine domänenspezifische Lösung erarbeiten. Diese Lösung kann für alle OEMs genutzt werden, da die Datenbasis allgemein gehalten ist. Diese wird aus den Garantie- sowie den Produktionsstammdaten bereits produzierter Fahrzeuge gebildet. Mit der Identifizierung ausfallgefährdeter Fahrzeugkonfigurationen können wir die Erfahrungen aus den Gewährleistungsdaten zur Rückkopplung in die Serienfertigung nutzen. Somit können im Feld als Qualitätsproblem erkannte Fehler gezielt in der Serienfertigung vermieden werden. In zukünftigen Arbeiten wird die vorgestellte Problemstellung formalisiert sowie die Anpassungen des Algorithmus in Form eines Pseudocodes detailliert. [BW16]

Literaturverzeichnis

[AS94] Agrawal, Rakesh; Srikant, Ramakrishnan: Fast Algorithms for Mining Association Rules in Large Databases. In: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. S. 487–499, 1994.

[BW16] Brunner, Franz J.; Wagner, Karl W.: Qualitätsmanagement. Leitfaden für Studium und Praxis. Hanser, München, 2016.

[HKP12] Han, J.; Kamber, M.; Pei, J.: Data mining. Concepts and techniques. Elsevier/Morgan Kaufmann, Amsterdam, 2012.

[JSW17] Jung, B.; Schweißer, S.; Wappis, J.: 8D - Systematisch Probleme lösen. Hanser, München, 2017.

[MNK15] Mertins, D.; Neumann, J.; Kühnel, A.: SQL Server 2014. Galileo Press, Bonn, 2015.

[Ve00] Verband der Automobilindustrie e.V. (VDA): , Jahresbericht, 2000.

[Ve09] Verband der Automobilindustrie e.V. (VDA): , Schadteilanalyse Feld, 2009.

[Wo18] Worldwide Automobile Warranties, https://www.warrantyweek.com/archive/ww20180816.html, Stand: 11.11.2018.

Context Selection in a Heterogeneous Legal Ontology

Sabine Wehnert, Wolfram Fenske, Gunter Saake[1]

Abstract: Ontology building in the legal domain is subject to ongoing research. Taxonomic ontologies provide for instance concept hierarchies for term definitions, annotations, query expansion and support for inferences. However, the context-dependent application of statuatory legal texts is hard to model, often leading to a limited ontology scope and fixed terminology to avoid conflicts. In previous work, we presented a method to create a lightweight heterogeneous ontology from textbooks offering connections between laws, while avoiding an error-prone and costly ontology alignment step. In our ontology, laws are linked by common contexts. We propose a new data model, so that the context can be explored and selected by a user, which is necessary for many applications, such as recommender systems. To obtain the relevant user context, we added a mechanism to retrieve linked laws from our ontology, given a scope of user interest and context information for each law.

Keywords: Heterogeneous Ontologies, Legal Text Linking, Context Selection, Full-text Search.

1 Introduction

Nowadays, people are overwhelmed by the amount of legal regulations to consider. Especially for international companies, it is becoming increasingly difficult to ensure that decisions comply with all laws. Therefore, our greater research goal is to provide a decision support system which monitors legislation and informs companies about relevant regulatory changes so they can update their processes to ensure legal compliance[2]. It is not trivial to determine relevance though, since it depends on many factors (e.g., user context, conditioned law applications). There are two main approaches to incorporate domain knowledge into a system. First, expert systems define answers for manually pre-defined queries, which is very costly. Second, ontologies are an approach to ensure a common understanding of the concepts of a domain, such that a query can be answered by rule-based reasoning. Ontologies are built from terms of increasing abstraction level, forming a concept hierarchy. For the legal domain, they can fulfill several reasoning tasks, for example finding consequences of a prohibition or obligation, or determining analogies between legal cases [Na12]. There are many legal ontologies [Aj16; Bu16; Ho07; So07], but they are limited to a highly specific domain (e.g., national law) or too abstract to be used as a stand-alone knowledge representation. For laws only describing rules for abstract events, a subsumption to real-world situations is necessary to understand whether a law applies to a given scenario [Di07]. As a consequence, experts

[1] Otto von Guericke University Magdeburg, <firstname>.<lastname>@ovgu.de
[2] The work is supported by Legal Horizon AG, Grant No.:1704/00082

doi:10.18420/btw2019-ws-13

create new or extend existing ontologies to fulfill the requirements of the respective user. This is time-consuming and costly. We therefore propose a mechanism which we call *context selection*, that allows users identify the laws applicable to their business scenario. It is a challenging task to model this user context for all possible situations, so we seek to automate this step by using external sources. In previous work, we extracted concept hierarchies from legal textbooks which capture law application contexts [We18]. In particular, we annotated table of contents elements (*TOC*) and applied them as a hierarchy for any cited legal text within the respective book. Our process is depicted in Figure 1. From each sentence containing a reference (*REF*), we compute a so-called citation summary (*CS*), the reason for citing the legal text in the given section.

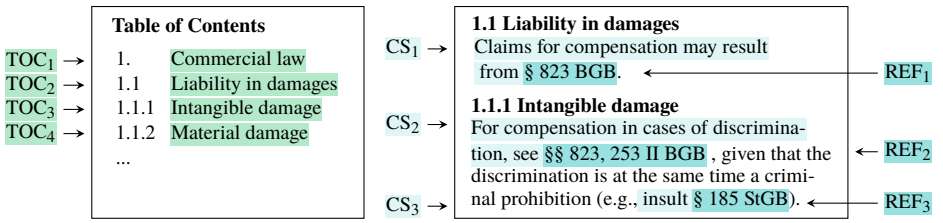


Fig. 1: Annotation process for references *(REF)*, citation summary *(CS)* and table of contents *(TOC)*.

For example, the citation summary for § 185 StBG covers the word *insult* as a reason for citing. Each book forms its own concept hierarchy from law citations within textbook sections. We follow the notion of a heterogeneous ontology by Visser and Cui [Vi98] who cluster concept hierarchies without any alignment. A cluster in our case can be a collection of similar books on one broad topic, such as banking or IT law. Clusters can be seen as knowledge modules which can be applied and queried separately. On one hand, books within each cluster may provide different perspectives on the same topic, and on the other hand they can enrich the knowledge base with their distinct content. Based on his or her interests, a user can select relevant concept hierarchies. In this work, we propose a mechanism for a user to navigate within the heterogeneous structure. Therefore, we develop a context selection method, based on two use cases:

For the first use case (a), the user searches for possible applications of a law. The user receives all occurrences of this law and context information from the concept hierarchy. Then, the user can select one context and determine the level of abstraction. After this context selection, all laws cited in the same context are retrieved. In the second use case (b), the user has a passive role and just receives an alert when a law from his or her context has changed. The context has to be selected beforehand, for example, by subscribing to one law and selecting one context description. By automatically expanding the subscription to laws referenced in the same context, users will also receive notifications about changes to relevant laws they were unaware of. To support these use cases, a suitable data model is needed. In this paper, we focus on the following aspects: First, we investigate an indexing method for law reference lookups. Second, we develop a data model for interactive graph traversal for knowledge extracted from legal textbooks with regard to the previously described use cases.

2 Context Selection

In this section, we describe the properties of our extracted data to choose appropriate search and data storage methods. Then, we explain our data model and methods to navigate within the data. Figure 2 illustrates the proposed workflow. We refer to the books which contain user-relevant contexts as the scope of interest S, which can be composed of several textbook clusters. The extracted concept hierarchies are stored in a graph database and replicated into a full-text search engine. Then, we consider one specific query for all occurrences of a reference to a law and its context. Given the query response, the user can select an appropriate context by choosing a cutoff point in the respective concept hierarchy. For instance, in Figure 1, the user can select the cutoff at section *1.1.1 Intangible damage*, so that any reference from *1.1.2 Material damage* will be excluded.

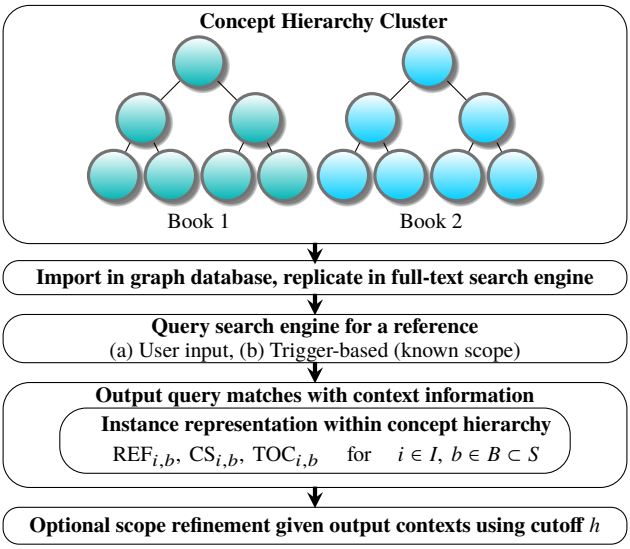


Fig. 2: Proposed workflow for context selection, given an output of matching instances i for a query. An instance consists of a reference *(REF)*, the citation summary *(CS)* and hierarchical context from the table of contents *(TOC)*. All instances I are drawn from textbooks B within the user-defined scope S. This scope is reduced by context selection: A user specifies a cutoff level h to prune higher and more abstract concept hierarchy levels and thereby removes connections to irrelevant laws.

2.1 Search Indexing

Using the context information, the user gets an overview of the concepts related to a legal text. We expect slight variations in law citations, such as roman or arabic numerals referring to a specific part of a law. Despite the need for approximate string matching, called fuzziness, the amount of variation needs to be controlled because the names of statute books can differ by a single letter, such as the German civil code *BGB* and the commercial code *HGB*.

Instead, we use exact matching for the query with references (while allowing for other present strings). In case no result is obtained (e.g., due to spelling mistakes), we employ fuzzy search. Despite the advantage of approximate string matching in full-text indexes, the data can also be stored as a graph. Graph data allow for connections regardless of hierarchy level and are optimized to process multiple outgoing relationships from one node. Later on, we plan to analyze the content of legal text documents, which can result in further links between laws, a so-called citation network. Graph data can be updated easily, however, the information from a book will not change, once it has been inserted into the ontology. Hierarchical storage of the data, for example in JSON format, is therefore also an option for search in hierarchical data, especially when approximate string matching is required. In our current prototype, we load all references to legal text as nodes into a graph database, create *PartOf* relationships with each corresponding table of contents element and replicate the data in hierarchical storage. Both systems have their own advantages - approximate string matching and graph traversal - and we can select for each query where to process it.

2.2 A Data Model for Graph Traversal over Linked Legal Texts

As a first step toward graph traversal, we transform the data which were previously extracted[3] into two separate `csv` files, one for the entities and one for their relationships. The data model is shown in Figure 3. We store all entities using the same `LABEL` *Node* in the graph and the search index. Furthermore, we define an additional field for each of them to preserve entity `TYPE` information (e.g., of type *Chapter*, *REF*). In the `FIELDSTRING`, we store the original text sequence from the book (see the highlighted sample text in Figure 1). There can be an additional `PROPERTY`, for example the statute book of each *REF* (e.g., BGB) or the noun groups within a *CS* which define the reason for citing (e.g., claims for compensation). A relationship connects two entities via an id pointer (`START_ID` for outgoing and `END_ID` for ingoing relationships). Relationships have a mandatory `TYPE` property. In our case, the relationship type is a *PartOf* relation indicating a bottom-up concept hierarchy, such that an entity of `TYPE` *Subsection* will be *PartOf* another entity of `TYPE` *Section*. References to legal text have only outgoing relationships, while the book instance at the top of the hierarchy just receives ingoing relationships. We directly access the `ID`s for scope definition and linked reference search. This data model supports our use cases as follows. Suppose a user is searching for possible applications of a law (a). The final output contains all references to that law, together with context information until the desired abstraction level. Likewise, a modified law may impact another law within the specified user context (b). An example for the latter case are changes in company size threshold values for German dismissal protection regulations (§23 Abs. 1 KSchG), which may be unknown to the user. By using graph traversal for our two given use cases, legal texts are retrieved which share the same concept hierarchy node with respect to a start reference. Nodes are traversed up to the user-chosen cutoff h by accessing the relationships to find the path to the next entity of type *REF*.

[3] An implementation of the first use case and previous work can be found at `https://github.com/anybass/HONto`

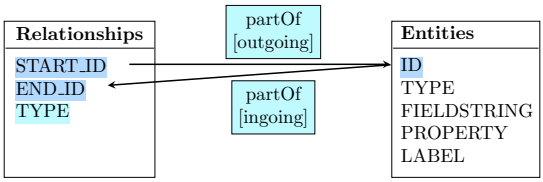


Fig. 3: Data model indicating the mapping between entity and relationship IDs in the csv files. The entity and relationship specifications contain their own TYPE information.

3 Related Work

First, we regard work in relation to legal ontology learning and second, we examine approaches for ontology-based query expansion. Legal ontology learning approaches are an automated way of constructing a legal ontology. We observe that it is possible to follow a combined top-down and bottom-up approach of ontology learning [Ag09; Ca08; El17; Fr10; Ho07; Pe07]. Unfortunately, these approarches either require expert input or use statistical language modeling methods which carry an inherent randomness and suffer from instability. Semi-automated approaches can assist in ontology population, but still face the challenge of transferring book knowledge - as it is - into an existing ontology without limiting the knowledge scope. While we identify context-dependent links between legal texts, a suitable domain ontology can be invoked for reasoning on the document level. Query expansion is a method to enrich user search terms with further related words - such as synonyms, hypernyms and homonyms - in order to retrieve more relevant items [Sc07]. Although this direction is promising, an understandable presentation of the higher amount of documents to the user is needed, while still accounting for a high recall. Using our context selection mechanism, a user can control and reduce the number of output documents by pruning irrelevant subtrees of a concept hierarchy.

4 Conclusion and Future Work

In this paper, we present a method to enable context selection in a heterogeneous lightweight ontology obtained from legal textbooks. Our ontology offers context-dependent relationships between legal texts. To this end, we propose a data model for storing the data in a graph database or in hierarchical format. We develop a context selection mechanism that helps a user navigate in our legal knowledge base and find different applications of a law, especially in two use cases: Ad-hoc search for a legal reference and a subscription service. For future work, we want to compare which storage option is better suited for other types of queries, such as topics. We will also examine how existing ontologies can be applied for document-level reasoning. Although preliminary results are promising, we will properly evaluate our approach in a user study with domain experts.

References

[Ag09] Agnoloni, T. et al.: A two-level knowledge approach to support multilingual legislative drafting. In: Proceedings of the 2009 conference on Law, Ontologies and the Semantic Web: Channelling the Legal Information Flood. 2009.

[Aj16] Ajani, G. et al.: The European Taxonomy Syllabus: A multi-lingual, multi-level ontology framework to untangle the web of European legal terminology. Applied Ontology 11/4, pp. 325–375, 2016.

[Bu16] Buey, M. G. et al.: The AIS Project: Boosting Information Extraction from Legal Documents by using Ontologies. In: Proceedings of the 8th International Conference on Agents and Artificial Intelligence. 2016.

[Ca08] Casellas Caralt, N.: Modelling Legal Knowledge Through Ontologies: OPJK: the Ontology of Professional Judicial Knowledge, PhD thesis, 2008.

[Di07] Dietrich, A. et al.: Agent Approach to Online Legal Trade. In (Krogstie, J.; Opdahl, A. L.; Brinkkemper, S., eds.): Conceptual Modelling in Information Systems Engineering. Springer Berlin Heidelberg, pp. 177–194, 2007.

[El17] El Ghosh, M. et al.: Ontology Learning Process as a Bottom-up Strategy for Building Domain-specific Ontology from Legal Texts. In: ICAART (2). 2017.

[Fr10] Francesconi, E. et al.: Integrating a bottom–up and top–down methodology for building semantic resources for the multilingual legal domain. In: Semantic Processing of Legal Texts. Springer, pp. 95–121, 2010.

[Ho07] Hoekstra, R. et al.: The LKIF Core Ontology of Basic Legal Concepts. In: Proceedings of the Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT). 2007.

[Na12] Naik, V. M. et al.: Building a Legal Expert System for Legal Reasoning in Specific Domain-A Survey. International Journal of Computer Science & Information Technology 4/5, p. 175, 2012.

[Pe07] Peters, W. et al.: The structuring of legal knowledge in LOIS. Artificial Intelligence and Law 15/2, pp. 117–135, 2007.

[Sc07] Schweighofer, E. et al.: Legal Query Expansion using Ontologies and Relevance Feedback. In: Proceedings of the Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT). 2007.

[So07] Soria, C. et al.: Automatic extraction of semantics in law documents. In: Proceedings of the V Legislative XML Workshop. 2007.

[Vi98] Visser, P. R. et al.: Heterogeneous Ontology Structures for Distributed Architectures. In: Workshop on Applications of Ontologies and Problem-solving Methods. 13th European Conference on Artificial Intelligence (ECAI-98), 1998.

[We18] Wehnert, S. et al.: Concept Hierarchy Extraction from Legal Literature. In: Proceedings of the ACM CIKM 2018 Workshops. CEUR Workshop Proceedings, 2018, URL: http://ceur-ws.org.

Software solutions for form-based, mobile data collection — A comparative evaluation

Markus D. Steinberg[1], Sirko Schindler[2], Friederike Klan[3]

Abstract:

Many citizen science projects rely on their contributors going to the field and collecting data. Due to their wide availability and increasing capability, modern mobile devices have become an indispensable tool to ease the collection process. Projects can publish mobile apps, that allow contributors to easily collect data and submit their results. The requirements of individual projects oftentimes overlap to a large extent, which triggered the development of multiple generic frameworks. They allow new projects to quickly generate customized apps and reuse existing infrastructure. However, the wide landscape of tools with diverging capabilities requires projects to compare and choose. This report supports data managers in making an informed decision. We report on our experiences primarily on the whole data collection workflow starting from setting up your own instance to finally analyzing the retrieved data. We compare eight tools – both free and commercial – according to the features provided and difficulties encountered.

Keywords: Data collection; Citizen Science; Mobile Software; Web Applications

1 Introduction

Collecting field data is an integral part of many projects in citizen science and other fields of research, especially the life sciences. Mobile applications offering form-based-surveys and making use of built-in sensors are increasingly used to facilitate this process. Since the creation of mobile field surveys from scratch can be a tedious task, multiple tools have been developed that can help with form design, data collection via mobile devices, data export and storage. Some even support simple data analysis.

With the increasing number of software options, data managers often face the question, which tool is most suitable for their current use case. To provide a clear foundation for such a decision, this paper evaluates and compares a selection of software tools for survey design

[1] Friedrich Schiller University Jena, Institute for Computer Science, Ernst-Abbe-Platz 2-4, 07743 Jena, Germany markus.daniel.steinberg@uni-jena.de

[2] German Aerospace Center (DLR), Institute of Data Science, Mälzerstraße 3, 07745 Jena, Germany sirko.schindler@dlr.de, https://orcid.org/0000-0002-0964-4457

[3] German Aerospace Center (DLR), Institute of Data Science, Mälzerstraße 3, 07745 Jena, Germany friederike.klan@dlr.de, https://orcid.org/0000-0002-1856-7334

doi:10.18420/btw2019-ws-14

and mobile data collection. To the best of our knowledge no previous, comparable study on this topic has been published yet.

All tools were tested with respect to different aspects relevant to their usage. A list of such tool-characteristics was compiled from the features that are advertised by the tools themselves. Availability and usability of these features were then evaluated by thoroughly testing each of the tools: First, general information about the tool like its open source repository (if available) and its license were identified by consulting its website and documentation. Then, if no web-based version was offered, a local, self-hosted instance of the tool was set up. Afterwards the typical data collection workflow (Fig. 1) was executed: (1) a form was designed, examining all available form-elements and form-building features like skip-logic or localization, (2) the survey was deployed to the mobile app, (3) sample data were collected, (4) submitted to the server and then (5) exported. Finally, (6) additional features like visualization or data encryption were explored. In cases where the availability of features was not obvious, the tool's community channels or its support team were consulted for clarification.

Fig. 1: Data collection workflow

Of course, such a study will never be able to cover all published tools, especially since their number is ever growing. The examined tools were selected because of their active and rapid development, their large user community, explicit recommendations by the tool's users, their extensive feature repertoire, or their professional design.

The following tools were considered:

- EpiCollect5 (EC5)[4], developed at the Imperial College London, the successor of Epicollect [Aa09] and Epicollect+ (Plus)[5],
- Open Data Kit 1 (ODK1) [Ha10] and Open Data Kit 2 (ODK2) [Br13], the two open source tool suites that are being developed by Nafundi[6] and the ODK community,
- Ohmage, an open source data collection platform that promises additional features like data analysis and visualization, developed at the University of California, Los Angeles[7], and Cornell Tech [8] [Ra12; Ta15],

[4] https://five.epicollect.net
[5] http://plus.epicollect.net/
[6] https://nafundi.com
[7] http://www.ucla.edu/
[8] https://tech.cornell.edu

- KoBo Toolbox, an open source data collection tool being developed by members of the Harvard Humanitarian Initiative (HHI) [Ha],
- SurveyCTO [Su] and Magpi [Ma], two paid subscription-based data collection platforms, which also offer free subscriptions that come with a few restrictions. For this study, the capabilities of the free versions were examined.

The Fieldtrip Open software suite developed by the EU-project Citizen Observatory Web (COBWeb), described in [Hi16] and published in [Ci], was initially considered as well, but was deemed unfit for thorough testing (see details in Sect. 2). The data collection tool BioCollect, which is popular in the biodiversity domain and was developed by the Atlas of Living Australia [Br18], was not examined in this study because, in contrast to the other examined tools, it is not generic, but tailored to recording species observations.

Sect. 2 reports on our hands-on experiences with the examined tools in the different phases of the data collection workflow and compares their features. Sect. 3 highlights the most important results and makes suggestions for future development and improvement of mobile data collection tools.

2 Comparative evaluation

The comparison evaluates the conditions for use and customization set by the considered tools and frameworks, discusses installation related aspects and goes on with usage related features grouped together according to the individual steps of the data collection workflow as illustrated in Fig. 1. A more in-depth version of this evaluation was published in [St].

Conditions for use and customization One of the first aspects that should be considered in the decision for or against a certain tool is the conditions it sets for using and extending it. The following facets were examined. The results for these elements are shown in Tab. 1.

Active Development Is the software currently under active development, i.e. can we expect that new features will be added over time and software bugs are fixed? This is judged, if possible, by commits in the past six months, otherwise by activity on social media or in forums.

License Under which conditions can the software be installed, used, or extended?

Open Source Is the source code of the software provided as open source? This includes all parts that are required to build and deploy a survey, collect data with a mobile device and store the data on a server.

Programming language Which programming languages are used for developing the tool?

Self-hosting Is it possible to host the software yourself, so the collected data is stored on your own server?

Tab. 1: Usage and development related criteria.
(●. . . criterion fulfilled; ○. . . criterion not fulfilled)

	EC5	ODKv1	ODKv2	Kobo	Ohmage	SurveyCTO	Magpi	COBWEB
Active Development	●	●	●	●	○	●	●	○
Open Source	○	●	●	●	●	○	○	●
Programming language	-	Java JavaScript Python	Java JavaScript	Java JavaScript Python	Java Objective C	-	-	JavaScript Python
License	-	Apache[a]	Apache[a]	Apache,[a] GNU[b]	Apache[a]	-	-	BSD3[c]
Self-hosting	○	●	●	●	●	○	○	●

[a] Apache License 2.0 [b] GNU Affero General Public License v3.0 [c] 3-Clause BSD License

Software installation and technical issues Some of the examined tools have components that need to be installed by the form author before they can be used. As an outcome of this step, we decided to exclude the Fieldtrip Open software suite from futher testing. Due to the missing documentation for its different parts, the software could not be fully set up in order to properly test its capabilities. Multiple attempts to set up the required persistence middleware on a Windows OS were unsuccessful. The middleware and the survey designer could finally be run on a Linux based computer, but the button that is supposed to save the designed survey resulted in JavaScript errors hinting at (1) access control problems and (2) problems inside the running middleware. It is unclear whether these errors occurred due to software problems or due to incorrect or missing configuration (which, in turn, could be the result of missing documentation).

The ODK suites require form authors to set up a server, ODK Aggregate, on a cloud-platform (AWS, Google, . . .) or host it themselves. This server is then used for survey distribution, data storage, visualization and data export. ODK2 additionally requires the setup of the ODK Application Designer which in turn requires Java, Google Chrome, NodeJS, Grunt and the Android SDK as prerequisites.

The other tools provide a web-based UI and do not involve an installation. Thus, no technical knowledge is required for their usage. All tools that offer a self-hostable version of their software provide detailed guides for the required steps. KoBo even maintains a Docker-version for a convenient setup. SurveyCTO's support team also mentioned in an email conversation that self-hosting of their software could be arranged on a case-by-case basis for their paying customers, provided that all additional costs will be settled by the user.

Survey design Almost all of the studied tools offer a form designer, a graphical user interface facilitating survey design, for example by allowing to add and arrange form elements via drag and drop. In the following, features are described that simplify form-authoring or allow to build more sophisticated and user-friendly surveys. The respective support among the tools is shown in Tab. 2.

Skip Logic Skip certain parts of a survey depending on previous answers.

Localization Define labels for questions in multiple languages, so the survey can automatically be translated to a user's preferred language.

Calculations Evaluate mathematical or logical expressions referencing answers to preceding questions in a survey and use the results in skip logic, text-blocks, etc.

Queries Read data from a structured source (e.g. a CSV file) and use the results for skip logic, as answer-options, etc.

Linked Tables Launch subforms that store data in different database tables.

Required & optional fields Mark a question as mandatory or optional to indicate whether the survey can be finished without providing an answer.

Validation Define validity constrains for form-fields, e.g., the range of valid values for a number input.

Building custom prompts Build prompts with custom functionality, typically using a markup language like HTML for the presentation and a programming language to define the functionality.

Tab. 2: The survey design features provided by the examined tools.
(●. . . feature included; ◐. . . feature partially included; ○. . . feature not included)

	EC5	ODKv1	ODKv2	Kobo	Ohmage	SurveyCTO	Magpi
Form Designer	●	●	○	●	●	●	●
Skip Logic	●	●	●	●	○	●	●
Localization	○	●	●	◐[a]	○	●	○
Calculations	○	●	●	○	○	●	●
Queries	○	○	●	○	○	○	○
Linked Tables	○	○	●	○	○	○	○
Required & Optional Fields	●	●	●	●	●	●	●
Validation	●	●	●	●	●	●	●
Building custom prompts	○	○	●	○	○	○	○

[a] Not supported in form designer, has to be added manually by exporting the form, editing the .xls file and then importing it.

The examined tools support and guide inexperienced form authors to a different extent. Particularly noteworthy are the extensive help-texts that are provided by SurveyCTO. They explain the typical workflow in the user interface as well as the different features and options that are available for each step. Inexperienced authors are thus guided through all necessary steps. SurveyCTO and Magpi also offer template forms that can be used to learn about the different form elements and their configuration. ODK1 provides explanations for the configuration of available form elements but the workflow-guidance is missing in the user interface. The reason is most likely the fact that ODK's form builder is a tool that is separated from the deployment-server and therefore also has a separate user interface.

Worth mentioning are also the wizards that EpiCollect5, KoBo, SurveyCTO and Magpi offer to build, for example, skip logic expressions. These wizards greatly simplify the formulation of complex logic statements, especially for inexperienced form authors.

As shown in Tab. 2, ODK2 offers some additional features like queries, linked tables and custom prompts that none of the other tools provide. However, ODK2 is the only one of the examined tools that does not offer any kind of form designer. Forms have to be created as .xls files and are then transformed using the ODK Application Designer. This, in addition to the more complex and technical deployment of surveys to the server and then to mobile devices, makes the usage of ODK2 more difficult compared to other tools. Thus, as the official ODK help page emphasizes, the usage of ODK2 is only recommended if the additional features are required for a certain use case [Op].

The form design also depends heavily on the input elements that are available. All of the tools provide support for the most basic types of information: text input, integer and decimal numbers, dates and times as well as single- and multiple-choice questions. They also allow to display textual information to the data collector.

Location information, which is a very important factor in many surveys, is also supported by all examined tools in the form of automated location determination using the collection device's sensors. However, manual input of a location, for example by placing a pointer on a map or explicitly stating latitude, longitude, altitude and/or accuracy, is only supported by ODK1 and SurveyCTO. More complex geographical information like paths (a sequence of locations) or an area (a closed path) are only supported by KoBo and SurveyCTO.

Images can be collected by all tools except Magpi; audio recordings, video recordings and barcodes by all except Magpi and Ohmage. Additionally, KoBo and Magpi offer some unique input elements: KoBo is currently developing and testing ratings (e.g. assigning "good", "bad" or "neutral" to a defined set of options) and rankings (ranking a predefined set of options). The latter could, for example, be used to express personal preferences, e.g. allow the user to state that he prefers apples over bananas over oranges.Magpi on the other hand allows to read information via Near-Field Communication[9].

[9] http://nearfieldcommunication.org/about-nfc.html

Data collection After the survey is designed and deployed on a server, data can be collected via a mobile device, typically using a mobile app. All of the examined tools provide support for offline data collection, meaning that data can be collected without an active internet connection and can later be submitted.

The support for mobile operating systems differs among the tools: All of them support Android-OS but only EpiCollect5, Ohmage and Magpi also offer apps for iOS. Apart from these two, no other operating systems are supported. For the open source tools, we were able to verify that the data collection apps are developed as native apps and not as cross-platform mobile applications. However, for closed source projects we were not able to obtain this information.

Another important factor in the data collection step is, if metadata are automatically gathered and stored with the collected data. Metadata describe the context of the collection process and the individual data records and therefore allow for a meaningful interpretation of the data. All tools provide information about date and time of data collection or submission as well as some kind of identification of the user who collected the data (username or e-mail address). ODK1, KoBo and SurveyCTO also allow to collect information about the mobile device that was used: the device-ID (IMEI), the subscriber-ID (IMSI), the SIM serial number or the phone number.

In cases where highly sensitive or private data is collected in a survey, the form author might want to ensure that no one will be able to get unauthorized access to the data. For such cases ODK1, KoBo and SurveyCTO provide the option to store a public key that automatically encrypts the data as soon as it is saved on the mobile device. The data can only be decrypted with the matching private key once the data is downloaded from the storage server. This ensures that the data is not only secure during the submission (which would usually be assured via the SSL/TLS protocol) but also while it is stored on the server, thus providing a strong level of security.

With automated quality checks, SurveyCTO offers an feature that can drastically improve data quality. In addition to the validation of the values inserted into form fields, such checks allow to detect submitted values that are outliers, values that occur too frequently, or other potentially faulty items. These quality checks can be configured to be run at certain time intervals and to be reported to a given e-mail address.

Data export & publication Regardless of the data collection tool, once the collected data are uploaded to the server, they can be exported as a file. The available file types differ among the examined tools: All of them support data export in the form of CSV files. As seen in Tab. 3 most of them also support some other file types that can be useful depending on the use case or user preferences.

Beside data export, data publication is directly integrated in some of the tools. EpiCollect5 provides an API to access its data and a guide on the usage of that API to publish the collected

data to Google Spreadsheets. ODK1 offers direct data publication to Google Spreadsheets, Google FusionTables, REDCap[10] servers and custom JSON servers. SurveyCTO also supports export to Google Spreadsheets and Google FusionTables. It additionally offers an integration with Zapier[11]. This is especially noteworthy since Zapier is a platform that can be used as a "bridge" to integrate the published data with hundreds of different services and applications.

A feature currently not provided by any of the examined tools is the semantic enrichment of the collected data. Embedding the assembled data in a semantic framework and interlinking individual data items with one another can give further interpretational context and allows a more seamless integration with other projects or information sources.

Tab. 3: The data export formats supported by the examined tools.
(●. . . feature included; ○. . . feature not included)

	EC5	ODKv1	ODKv2	Kobo	Ohmage	SurveyCTO	Magpi
CSV	●	●	●	●	●	●	●
JSON	●	●	○	○	○	○	○
XLS	○	○	○	●	○	●	●
XML	○	○	○	○	○	○	○
XML/KML	○	●	○	○	○	●	○
RDF	○	○	○	○	○	○	○

Data visualization & analysis All of the tools provide some kind of built-in support for data visualization, though the supported types vary as seen in Tab. 4. On the other hand, data analysis is currently not supported by any of the tools. Ohmage claims to provide such features but does not properly integrate them in the tool's interface. SurveyCTO provides a way to monitor the incoming data and visualize relationships between different fields of data, but does not offer the capabilities of full analytic software. Magpi seems to have a similar feature as SurveyCTO but it is locked for free users.

Currently, the best way to analyze the collected data, regardless of the tool, is to either export it using one of the supported file types and then import that file in a data analysis tool of one's choice or to publish the data on a cloud-based platform and then use analysis tools that integrate well with the platform.

[10] https://www.project-redcap.org
[11] https://zapier.com

Tab. 4: Visualization types, analysis and semantic enrichment features
(○. . . feature not included)

	EC5	ODKv1	ODKv2	Kobo	Ohmage	SurveyCTO	Magpi
Visualization	Map, Pie chart	Map, Pie chart, Bar chart	Map	Map, Pie chart, Bar chart, Line chart, Area chart	Map, Pie chart, Bar chart, Line chart	Map, Pie chart, Bar chart, Scatterplot, Trend plot[a]	Map
Analysis	○	○	○	○	○	○	○
Semantic Enrichment	○	○	○	○	○	○	○

[a] Plotting a numeric value over time

3 Conclusion

The presented comparison shows that due to the different features that are offered by the different software tools, the choice of a platform depends on the given use case with its unique requirements. However, it also shows that ODK1 and KoBo Toolbox are the open source tools that offer the most comprehensive set of features. SurveyCTO, on the other hand, offers the most professional and user-friendly environment if the limitations of the free subscription are of no concern. For most data collection projects, at least one of these three tools should be able to cover the requirements.

Another point that this comparison shows very clearly is that none of the tools currently provide any kind of semantic component that would provide a unique and machine-comprehensible semantics of the exported data. The only tool that seemed to take a step in this direction was the COBWeb software suite with its RDF export, which was not fit for proper testing. Since linked and semantically enriched data enable more meaningful interpretation of the data and even automated reasoning, such features could drastically improve both quality and usability of the collected data. Therefore, such features deserve some attention in the future enhancement of data collection platforms and tools.

Analysis support for the collected data is another point that could be integrated in the tools. Currently, projects have to rely on external tools for such features, which means that the data either has to be transferred to a cloud platform or manually exported and imported into some analysis software. Both options require additional time-consuming effort and could in some cases even create privacy issues if highly sensitive data is involved.

A third improvement that could be taken into account by the examined tools is the use of cross-platform technologies for mobile applications. Advantages here would be two-fold: Data collection projects would only have to maintain a single code base for all their mobile applications. At the same time this could increase the number of potential survey participants and therefore the engagement in projects that involve data collection.

References

[Aa09] Aanensen, D. M.; Huntley, D. M.; Feil, E. J.; Spratt, B. G., et al.: EpiCollect: Linking Smartphones to Web Applications for Epidemiology, Ecology and Community Data Collection. PLoS ONE 4/9, ed. by Hay, S. I., e6968, Sept. 2009.

[Br13] Brunette, W.; Sundt, M.; Dell, N.; Chaudhri, R.; Breit, N.; Borriello, G.: Open data kit 2.0: expanding and refining information services for developing regions. In: Proceedings of the 14th Workshop on Mobile Computing Systems and Applications - HotMobile '13. ACM, ACM Press, p. 10, 2013.

[Br18] Brenton, P.: BioCollect - A modern cloud application for standards-base field data recording. Biodiversity Information Science and Standards 2/, e25439, May 2018.

[Ci] Citizen Observatory Web: COBWEB - Software Outputs, URL: https://cobwebproject.eu/news/publications/software-outputs, visited on: 10/31/2018.

[Ha] Harvard Humanitarian Initiative: KoBoToolbox: Data Collection Tools for Challenging Environments, URL: https://www.kobotoolbox.org/, visited on: 10/27/2018.

[Ha10] Hartung, C.; Lerer, A.; Anokwa, Y.; Tseng, C.; Brunette, W.; Borriello, G.: Open data kit: tools to build information services for developing regions. In: Proceedings of the 4th ACM/IEEE International Conference on Information and Communication Technologies and Development - ICTD '10. ACM, ACM Press, p. 18, 2010.

[Hi16] Higgins, C. I.; Williams, J.; Leibovici, D. G.; Simonis, I.; Davis, M. J.; Muldoon, C.; van Genuchten, P.; O'Hare, G.; Wiemann, S.: Citizen OBservatory WEB (COBWEB): A generic infrastructure platform to facilitate the collection of citizen science data for environmental monitoring. International Journal of Spatial Data Infrastructures Research (IJSDIR) 11/, pp. 20–48, 2016.

[Ma] Magpi: Advanced Mobile Data Collection, Messaging, and Visualization, URL: https://home.magpi.com/, visited on: 10/27/2018.

[Op] Open Data Kit: ODK Help page, URL: https://opendatakit.org/help/), visited on: 10/28/2018.

[Ra12] Ramanathan, N.; Alquaddoomi, F.; Falaki, H.; George, D.; Hsieh, C.-K.; Jenkins, J.; Ketcham, C.; Longstaff, B.; Ooms, J.; Selsky, J., et al.: ohmage: An open Mobile System for Activity and Experience Sampling. In: Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare. IEEE, IEEE, pp. 203–204, 2012.

[St] Steinberg, M. D.: Software solutions for form-based collection of data and the semantic enrichment of form data, arXiv: 1901.11053 [cs.CY].

[Su] SurveyCTO: Collect data you can trust, URL: https://www.surveycto.com/, visited on: 10/27/2018.

[Ta15] Tangmunarunkit, H.; Hsieh, C.-K.; Longstaff, B.; Nolen, S.; Jenkins, J.; Ketcham, C.; Selsky, J.; Alquaddoomi, F.; George, D.; Kang, J., et al.: Ohmage: A general and extensible end-to-end participatory sensing platform. ACM Transactions on Intelligent Systems and Technology 6/3, pp. 1–21, Apr. 2015.

Quality Indicators for Text Data

Cornelia Kiefer[1]

Abstract: Textual data sets vary in terms of quality. They have different characteristics such as the average sentence length or the amount of spelling mistakes and abbreviations. These text characteristics have influence on the quality of text mining results. They may be measured automatically by means of quality indicators. We present indicators, which we implemented based on natural language processing libraries such as Stanford CoreNLP[2] and NLTK[3]. We discuss design decisions in the implementation of exemplary indicators and provide all indicators on GitHub[4]. In the evaluation, we investigate free texts from production, news, prose, tweets and chat data and show that the suggested indicators predict the quality of two text mining modules.

Keywords: data quality, text data quality, text mining, text analysis, quality indicators for text data

1 Introduction

Plenty research on the quality of structured data tries to capture the quality of a data set as a number, e.g., in the interval [0,1] where 0 means bad quality and 1 means high quality (e.g., see [SC13], [CKK17]). E.g., the percentage of null, out-of-domain and duplicate values indicate the quality of structured data sets and can be expressed as a number in [0,1]. These methods are based on a comparison of the structured data to a 'perfect' version of the data (or parts of it) that represent the real world, or to a rule that captures characteristics of such perfect versions of the data set. Unstructured textual data needs to be processed in natural language processing pipelines. Thus, additional means to capture how text characteristics influence the quality of text analysis modules in such pipelines are needed. However, corresponding methods for texts are missing [BS16]. The indicators suggested in this work may automatically measure text characteristics such as the percentage of spelling mistakes, the number of unknown words and the confidence of standard text processing tools. Quality indicators for text data are needed: the amount of unstructured text data is exploding. Moreover, text data in science, humanities and industry comprise various crucial information such as descriptions of experimental settings, experience reports, error reports, and machine documentations [Ka14, LW16].

[1] University of Stuttgart, Graduate School of Excellence Advanced Manufacturing Engineering, Nobelstr. 12, Germany cornelia.kiefer@gsame.uni-stuttgart.de

[2] https://stanfordnlp.github.io/CoreNLP/

[3] http://www.nltk.org/

[4] https://github.com/kieferca/quality-indicators-for-text

doi:10.18420/btw2019-ws-15

These textual data sets differ significantly in quality. Moreover, many crucial textual data sets in science, humanities and industry are of low quality (e.g., see [KM16]). While the domain experts read parts of the textual data and are thus often aware of quality problems, concrete implemented indicators, which can be used to characterize the textual data sets are missing. Moreover, the influence of certain text characteristics on the quality of text mining results can only be discussed if concrete indicators are available. By now it is not clear what data indicators are useful and how they capture the quality of different textual data sets.

A worker or analysts who is reading a text full of spelling mistakes and abbreviations may have problems to understand it. Also, a text of bad quality may result in bad quality text mining results. The quality of such text mining results can only be calculated if manual annotations are available. Usually this is not the case for data sets in industry, science and humanities. Nevertheless, the quality of operational data sets may be indicated by means of the data quality indicators presented in this work. This paper has two **main contributions** that are facing these challenges: (1) we present 9 quality indicators for texts and (2) we investigate to what extend the suggested indicators are able to predict the quality of text analysis results of a language identifier and a part of speech tagger.

We start this paper with a presentation of related work in Section 2. Then, we list concrete quality indicators and discuss design decisions in the implementation (Section 3). In Section 4 we present and characterize the data sets used in the evaluation. Finally, we test the suggested quality indicators and present the results in Section 5. We conclude our work in Section 6.

2 Related Work

Many data quality indicators for structured data exist (e.g., [SC13, WS96]). Moreover, many works present first conceptual ideas for data quality methods for text [Sc12, BS16, So04]. However, none of these works presents quality indicators with concrete implementations that are applied to texts.

Data quality research on text is still in its beginnings, but the quality of textual documents is already considered in other research areas and applications. For example, the quality of written student essays [MK00] and of posts in online discussions [WGM07] can be assessed automatically. Also, many companies provide guidelines in writing texts such as error reports. The guidelines ensure that the texts can be processed automatically in high quality, e.g., by machine translation systems [Ku13]. For example, very long and nested sentences should be avoided with respect to the quality of an automatically generated translation of a text. Researchers on text simplification develop automated methods to simplify texts [Sh14]. Genova et al. suggest a framework to measure and improve the quality of textual specifications for software [Gé13]. While these works provide interesting starting points with respect to text data quality, they do not provide means to characterize the quality of textual data sets with respect to the quality of text analysis modules, such as the language

identifier and part of speech tagger as considered in this work. If indicators are suggested at all, these are special to the respective domain and no implementation details are given.

Readability measures such as the Flesch readability index capture how easy and fast a human may read and understand a text. Flesch's formula is based on the number of words per sentence and the number of syllables per word. For an overview on readability indices, see Klare [Kl74]. Many automatic readability checkers exist[5]. For these tools, no implementation details are given, though, and the code is closed-source. These readability measures only capture a very limited set of text characteristics, namely the number of words, syllables and sentences.

Particularly in the medical domain, much work is done in automatically detecting and resolving abbreviations (e.g., see [Li18]). In these works, the focus lies on resolving abbreviations and the percentage of abbreviations is not used as quality indicator. Botha et al. [BB12] investigate the effect of text size on the accuracy of language identifiers. They found that, the smaller the text size, the lower accuracy is. In our evaluation, we confirm this result. Additionally, we add more indicators besides text size and investigate the accuracies of a language identifier as well as a part of speech tagger.

While many valuable first reference points for quality indicators for text data exist, they do not cover all necessary aspects. They are oftentimes not executable or closed-source and come from fields different than data quality research and thus have a limited perspective on text quality. Moreover, in none of these related works indicators are applied to various data sets of varying quality as characterized by text analysis modules.

3 Quality Indicators for Text Data

In a text analysis pipeline the raw textual data is processed by several text analysis modules such as a language identifier, a part of speech tagger and a named entity recognizer. These modules enrich the textual data with information on the language a text is written in, the parts of speech of the words such as *verb* and *noun* and with named entities such as on companies, countries and persons. Thus, a reasonable measurement of the quality of texts needs to consider two main components: (1) the raw text data and (2) the text analysis modules. The latter group of indicators measure text characteristics with respect to standard text analysis modules, which employ default resources (such as newspaper texts as training data). These standards and defaults are oftentimes employed in domain-specific text analysis pipelines.

In Table 1, we present a non-exhaustive list of quality indicators for textual data. We restrict the methods presented to those applicable to textual data in the context of text analysis. All indicators are freely available on GitHub[6]. In the following, we describe an

[5] e.g., hemingwayapp.com and readable.io
[6] https://github.com/kieferca/quality-indicators-for-text

exemplary excerpt of the indicators and give design decisions in the implementation. The implementation of the indicator 'percentage of abbreviations' is based on a supervised machine learning algorithm and is more complex. All other quality indicators listed in Table 1 have straightforward implementations which are based on existing natural language processing libraries.

Tab. 1: Text Data Quality Indicators with respect to Data and Text Analysis Modules

Group	Indicator ID	Indicator Description
Data	1	Percentage of abbreviations
	2	Percentage of spelling mistakes
	3	Lexical diversity
	4	Percentage of uppercased words
	5	Percentage of ungrammatical sentences
	6	Average sentence length
Text Analysis Modules	7	Fit of (default) training data
	8	Confidence of standard processing modules
	9	Percentage of unknown words

The implementation of the first indicator, which automatically measures the **percentage of abbreviations (indicator 1)** is based on the Stanford Named Entity Recognizer[7]. This is a classifier which automatically recognizes named entities such as persons, cities and companies. Therefore, it uses information gained via natural language processing, such as the part of speech tags and syntax. Also, it uses training data manually annotated with named entities. It is based on conditional random fields (CRF), a supervised machine learning algorithm for sequential classifications[8]. In our case, the sequence to classify is a sequence of words. Given the sequence of words, the method classifies each word as abbreviation or non-abbreviation. To adapt the Stanford NER classifier to the task of determining if a word is an abbreviation or not, we trained it on a new training data set, which we compiled by manually annotating all individual words in a text collection with the two labels abbreviation and non-abbreviation. The compiled training data set is based on annotated excerpts of the data sets listed in Section 4. We moreover adapted the Stanford Named Entity Recognizer to the task of detecting abbreviations by implementing additional features, which are based on natural language processing methods from Stanford CoreNLP[9]: (1) word length, (2) contains symbols, (3) contains period, (4) sentence dependencies, (5) sequence of vowels and consonants representing the current word and (6) wordform (sequence of upper and lowercased characters representing the current word). We evaluated the classifier prototype on unseen data resulting in a precision of 0,85 and a recall of 0,72. Thus, it works reliable enough for our purpose of measuring the percentage of abbreviations as data quality indicator. For the calculation of precision and recall, we used the data sets as described in Section 4 and split them into separate training and testing slices. In Section 5

[7] https://nlp.stanford.edu/software/CRF-NER.shtml
[8] The CRF sequence models used are described in [FGM05].
[9] https://stanfordnlp.github.io/CoreNLP/

we will investigate if the percentage of abbreviations in a text is useful in predicting it's quality.

The **percentage of spelling mistakes (indicator 2)** in a text corpus may be calculated using the Python implementation PyEnchant[10] or any other spelling correction module.

Lexical diversity (indicator 3) is calculated using standard methods in NLTK for counting words. It is based on a formula suggested in the NLTK book [BKL09]. The relevant code is displayed in Listing 1, where the length (len) of the set of all tokens and words in the text (set) is divided by the length of all tokens and words in the text.

```
1 def lexical_diversity(text):
2 return (len(set(text)) / len(text))
```

List. 1: 'Lexical diversity' implementation based on standard Python tools

For measuring the **fit of (default) training data (indicator 7)**, we calculate the text similarity of the operational text data set that is actually being analyzed and the default training data set. Since it is most often used as default in many processing modules in natural language processing, we use the Treebank data set as default (see Section 4). We employ the Cosine Similarity metric from the DKPro Similarity library[11]. The core method used is illustrated in Listing 2. The whole concept, design decisions in implementation and a throughout evaluation with various text similarity metrics will be presented in future work.

```
1 TextSimilarityMeasure measure = new CosineSimilarity();
2 double score = measure.getSimilarity(operational, default);
```

List. 2: Excerpt of 'Fit of default training data' implementation based on DKPro Similarity

The **confidence of standard processing modules (indicator 8)** can be calculated for many classifiers, e.g., for the part of speech tagger. A statistical classifier estimates the probabilities for each class from a fixed list of classes. These probabilities are also called confidence values (for more details, see [GFL06]). Confidence is expressed as a number in the interval [0,1]. For example, confidence measures are available and can be retrieved for the natural language processing tools in OpenNLP[12] (such as the tokenizer and part of speech tagger). To get these confidence values, we followed the documentation of the OpenNLP library (see footnote 12). E.g., for the part of speech tagger, we just call the *probs* method which returns an array of the probabilities for all tagging decisions. The method is shown in Listing 3. Then, we calculate the mean over all sentences and return it. In Section 5 we discuss if these confidence values for the OpenNLP part of speech tagger may be used as a quality indicator.

[10] http://pythonhosted.org/pyenchant/
[11] https://dkpro.github.io/dkpro-similarity/
[12] https://opennlp.apache.org/

```
1 POSTaggerME tagger = new POSTaggerME(model);
2 tagger.tag(sentence);
3 double probs[] = tagger.probs();
```

List. 3: Excerpt of 'confidence of standard processing modules' implementation based on OpenNLP

The **percentage of unknown words (indicator 9)** may be calculated by applying the standard part of speech tagger implemented in NLTK to the texts, which has an individual class for unknown words, i.e., 'X'.

The measured percentages and raw numbers need to be transferred into consistent data quality metrics in [0,1] where 0 means low and 1 high quality. We transfer the measured percentages and raw numbers by means of adequate step functions. Some indicators such as 'confidence' are already fitting numbers in [0,1] and do not need to be transferred. But other indicators such as the percentage of abbreviations and the average sentence length need to be transferred into a consistent quality metric in [0,1]. For example, the first indicator measures the percentage of abbreviations. A high percentage of abbreviations should result in a low quality metric and a low percentage of abbreviations in a high quality metric. This can be achieved by means of a step function, where, e.g. 0-1% abbreviations are transferred to the quality metric 1 and >10% to 0, etc. We will describe this transfer of indicators to data quality metrics in more detail in future work.

The indicators presented build the basis for methods that can improve the quality of texts. These will be adressed in future work. For example, if the measured percentage of spelling mistakes is high, data quality may be improved by means of an automatic spelling mistakes correction method.

4 Data Sets used in the Evaluation

We conduct experiments on 5 different data sets. They comprise prose, news, chat posts, tweets and production data. The prose and news data sets (Brown and a subset of the Penn Treebank) and chat posts (NPS Chat data) are taken from NLTK[13]. The Twitter corpus was taken from Gimpel et al. [Gi11]. Additionally, we employ a confidential data set from an industry partner in Germany. It comprises information on downtimes in a production line and contains German free text information. The data set contains information on the reasons for downtimes and the actions that were taken to put the production line running again. The workers on the shop floor can fill the free text field via text entry into a tablet. In Table 2 we list the main characteristics of the data sets. All data sets come with gold annotations for at least one text mining module. Thus, in our evaluational setting, we are able to calculate accuracies. Accuracy calculations will be discussed in the next section.

[13] http://www.nltk.org/nltk_data/

Tab. 2: Data sets used in the experiments

Data collection	Type	# of tokens
Brown	Prose	1.15M
Treebank (stub)	News	40k
Twitter corpus	Tweets	35k
NPS Chat	Chat	45k
Industry corpus	Production	153k

5 Evaluation

The quality of text mining results is judged by comparing the predictions of the tools with the gold labels annotated by human experts. For example, to determine the quality of a part of speech tagger, it's 'Token Accuracy' is calculated as shown in Equation 1.

$$ACC = \frac{(\#\ correct\ POS\ tags\ in\ tagged\ data)}{(\#\ total\ POS\ tags\ in\ tagged\ data)} \tag{1}$$

The accuracy of language identifiers is calculated by comparing the gold language annotations with the annotations made by the tool. As already mentioned in the introduction of this work, accuracies can only be calculated if manual annotations are available. This is oftentimes not the case. The calculation of the suggested quality indicators does not need such manual annotations, though. In Table 3, we show first results with respect to whether they are able to predict the quality of such tools.

In the first column in Table 3, we note the data set. In the following three columns we present the overall and single accuracies for two text analysis modules: (1) the Apache Tika language identifier[14] (LI (Tika)) and (2) the CRF part of speech tagger from NLTK[15] (POS (CRF)). Compiling manual annotations costs time and expert knowledge. Therefore, as oftentimes the case for operational data sets, for the industry data no manual annotations of part of speech are available. Thus, part of speech (POS) accuracy can't be calculated. Nevertheless, the accuracy of the language identifier (LI) and the indicators give insights on the textual characteristics. In the following columns, we present the results for our suggested quality indicators for text data.

We report the raw numbers gained for data quality indicators as described in Section 3. Thus, most indicators are measured in percent and some are plain numbers such as the average sentence length. In future work, these raw measurement results need to be transferred to uniform data quality metrics as already mentioned in Section 3. Also, further analysis modules, implementations and data sets need to be adressed in future work.

From first to last row, the overall accuracy of the two text mining modules decreases. Treebank and Brown (news and prose) can be processed in a reliable quality by these text

[14] https://tika.apache.org/

[15] https://www.nltk.org/_modules/nltk/tag/crf.html with the universal tagset and Treebank training data

Tab. 3: Evaluation results

Data	Accuracy			Indicator								
	Overall	LI (Tika)	POS (CRF)	Abbreviations (1)	Spelling (2)	Lexical Diversity (3)	Uppercased (4)	Ungrammatical (5)	Avg. Sentence Length (6)	Fit of training data (7)	Confidence (8)	Unknown words (9)
Treebank	0,90	0,86	0,94	2,0	19,0	1,5	1,6	0,1	24,0	1,0	0,9	0,0
Brown	0,86	0,84	0,88	0,6	12,0	0,1	0,9	0,4	20,3	0,9	0,9	0,1
Twitter	0,62	0,47	0,76	4,6	27,0	10,6	7,0	1,5	14,5	0,5	0,8	0,2
Chat	0,49	0,20	0,78	11,0	34,0	4,6	15,8	1,0	4,3	0,5	0,6	0,3
Industry	n.a.	0,34	n.a.	7,1	23,0	0,4	0,1	0,0	4,8	0,5	0,7	0,5

mining tools (=Overall Accuracy is high). Tweets, chat posts and industry data can only be processed in low quality (=Overall Accuracy is low). A similar classification is made by the data quality indicators: Treebank and Brown contain less abbreviations and spelling mistakes and have a low lexical diversity. The amount of uppercased characters is low. They hardly contain ungrammatical sentences. The sentences are longer when compared to tweets and especially when compared to chat and industry data. The fit of training data and confidence are high, and the amount of unknown words is low.

Low quality of Tweets, Chat posts and Industry data is indicated by many abbreviations, spelling mistakes and unknown words as well as a low fit of training data and low confidence values. Tweets contain a significantly higher lexical diversity than the other data sets. Chat posts contain particularly many abbreviations and lexical diversity is high. In both, Tweets and Chat posts, more words than in the other data sets are uppercased and they contain more ungrammatical sentences. In Chat and Industry data the sentences are very short. The industry data is full of domain-specific abbreviations, unknown words and spelling mistakes. Lexical diversity is rather low and the sentences are very short and parseable, i.e. the number of ungrammatical sentences is low.

Both text analysis modules selected are high quality modules oftentimes employed in text mining projects in industry. While the language identifier seems to be very sensible with respect to some text characteristics, the part of speech tagger is more robust. From Table 3 it can be seen that the suggested indicators are good starting points that may indicate quality. Thus, in a real analysis situation in humanities, science or industry, where accuracies are not calculable and thus not known, the suggested data quality indicators give a hint on how good processing modules may be able to cope with the data set(s).

6 Conclusion

We have presented 9 data quality indicators for text data sets. Operational text data sets usually do not come with manual gold annotations for text processing steps. Thus, the quality of many text analysis results is not known in text mining projects in the humanities, science and industry. We suggested data quality indicators which help in deciding if default text mining modules will deal easily with the textual data or not, i.e. if improvement strategies are needed or not. For each indicator, corresponding improvement strategies exist, which will be adressed in future work. Moreover, in future work we adress the transformation of percentages and raw numbers into data quality metrics in [0,1] and integrate and combine the methods into a complete framework for data quality assessment and improvement.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG) for financial support of this project as part of the Graduate School of Excellence advanced Manufacturing Engineering (GSaME) at the University of Stuttgart. Moreover, we thank Raoul Graumann and Marco Link for crucial implementation work.

References

[BB12] Botha, Gerrit Reinier; Barnard, Etienne: Factors that affect the accuracy of text-based language identification. Computer Speech & Language, 26(5):307–320, 2012.

[BKL09] Bird, Steven; Klein, Ewan; Loper, Edward: Natural Language Processing with Python. O'Reilly Media, 2009.

[BS16] Batini, Carlo; Scannapieco, Monica: Data and Information Quality. Springer International Publishing, Cham, 2016.

[CKK17] Chung, Yeounoh; Krishnan, Sanjay; Kraska, Tim: A Data Quality Metric (DQM): How to Estimate the Number of Undetected Errors in Data Sets. Proc. VLDB Endow., 10(10):1094–1105, 2017.

[FGM05] Finkel, Jenny Rose; Grenager, Trond; Manning, Christopher: Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. ACL '05, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 363–370, 2005.

[Gé13] Génova, Gonzalo; Fuentes, José Miguel; Morillo, Juan Llorens; Hurtado, Omar; Moreno, Valentin: A framework to measure and improve the quality of textual requirements. Requir. Eng., 18(1):25–41, 2013.

[GFL06] Gandrabur, Simona; Foster, George; Lapalme, Guy: Confidence Estimation for NLP Applications. ACM Transactions on Speech and Language Processing (TSLP), 3(3):1–29, 2006.

[Gi11] Gimpel, Kevin; Schneider, Nathan; O'Connor, Brendan; Das, Dipanjan; Mills, Daniel; Eisenstein, Jacob; Heilman, Michael; Yogatama, Dani; Flanigan, Jeffrey; Smith, Noah A.: Part-of-speech Tagging for Twitter: Annotation, Features, and Experiments. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2. HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 42–47, 2011.

[Ka14] Kassner, Laura; Gröger, Christoph; Mitschang, Bernhard; Westkämper, Engelbert: Product Life Cycle Analytics - Next Generation Data Analytics on Structured and Unstructured Data. In: Proceedings of the 9th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '14. Elsevier, Naples, pp. 1–6, 2014.

[Kl74] Klare, George R.: Assessing Readability. Reading Research Quarterly, 10(1):62–102, 1974.

[KM16] Kassner, Laura; Mitschang, Bernhard: Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics. In: Advances in Database Technology - EDBT 2016, 19th International Conference on Extending Database Technology, Proceedings. OpenProceedings.org, pp. 491–502, 2016.

[Ku13] Kuhn, Tobias: A Survey and Classification of Controlled Natural Languages. Computational Linguistics, 40(1):121–170, 2013.

[Li18] Liu, Yue; Ge, Tao; Mathews, Kusum S.; Ji, Heng; McGuinness, Deborah L.: Exploiting Task-Oriented Resources to Learn Word Embeddings for Clinical Abbreviation Expansion. CoRR, abs/1804.04225, 2018.

[LW16] Lemke, Matthias; Wiedemann, Gregor: Text Mining in den Sozialwissenschaften. Springer Fachmedien, Wiesbaden, 2016.

[MK00] Miltsakaki, Eleni; Kukichy, Karen: Automated evaluation of coherence in student essays. In: Proceedings of LREC, pp. 1–8. 2000.

[Sc12] Schmidt, Andreas; Ireland, Chris; Gonzales, Eloy; Del Pilar Angeles, Maria; Burdescu, Dumitru Dan: , On the Quality of Non-structured Data, 2012.

[SC13] Sebastian-Coleman, Laura: Measuring data quality for ongoing improvement: A data quality assessment framework. Elsevier Science, Burlington, 2013.

[Sh14] Shardlow, Matthew: A Survey of Automated Text Simplification. International Journal of Advanced Computer Science and Applications(IJACSA), Special Issue on Natural Language Processing 2014, 4(1), 2014.

[So04] Sonntag, Daniel: Assessing the Quality of Natural Language Text Data. In: GI Jahrestagung. pp. 259–263, 2004.

[WGM07] Weimer, Markus; Gurevych, Iryna; Mühlhäuser, Max: Automatically Assessing the Post Quality in Online Discussions on Software. In: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions. ACL '07, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 125–128, 2007.

[WS96] Wang, Richard Y.; Strong, Diane M.: Beyond accuracy: what data quality means to data consumers. J. Manage. Inf. Syst., pp. 5–33, 1996.

Entity Extraction in the Ecological Domain — A practical guide

Vladimir Udovenko[1], Alsayed Algergawy[2]

Abstract: Scientific information comes in many shapes: As data in databases or spreadsheets, but also as textual information in papers and books. In order to exploit all this information and integrate all the knowledge that is available regarding a specific entity, it is necessary to identify entities and their relationships. In this paper, we provide a guideline to setting up a pipeline that supports entity and relationship extraction from scientific publications from the ecological domain.

Keywords: Information integration; Entity extraction; Relation extraction

1 Introduction

Research builds on knowledge gained from earlier resources. Such knowledge is encoded in data stored in various data sources as well as text. An essential step for integrating data from these heterogeneous data sources is to identify similar entities represented in different sources as well as their relations[DHI12]. However, the majority of scientific data are represented in unstructured formats, i.e. information and knowledge of interest are still hidden mostly in data sets without any formalized schema. It maybe scientific publications. They may contain tables or pictures, but mostly text data. The main objective is to make such information stored in text accessible for further data processing, such as integration and analysis [YB18].

Extracting information of interest from scientific publications in general and ecological in specific including entities and relations is a critical challenge to support the automation of integrating structured and unstructured data [KN04]. Suppose that we have this sample of a scientific publication "N2O contributes to the destruction of ozone layer", it becomes more difficult to identify and recognize named entities in such a domain specific scenario. Compared to the personal domain, which is well-established, it is hard to create and/or get annotations for such named entities. This requires the need to prepare training datasets that can be used either in learning-based approaches or as a list of domain-specific entities in the rule-based approaches. In both cases, the preparation process includes the collection and organization of domain specific information resources, such as ontologies.

[1] Friedrich-Schiller University of Jena, Heinz Nixdorf Chair for Distributed Information Systems, Germany
[2] Friedrich-Schiller University of Jena, Heinz Nixdorf Chair for Distributed Information Systems, Germany
alsayed.algergawy@uni-jena.de

doi:10.18420/btw2019-ws-16

To this end, in this paper, we describe how existing building blocks can be combined to create a framework that supports in the identification and extraction of soil-related entities from scientific publications belonging to the Biodiversity Exploratory[3]. The extracted set of entities are then annotated by domain specific resources, which support the identification of relations across the entities. The proposed approach is implemented and validated using more than 100 publications and the preliminary experiments demonstrate encourage results.

2 Related work

The main goal of information extraction is the organization and structure of hidden knowledge in textual data that makes it accessible for other applications, e.g. as part of joint data integration systems [Ho02, Go18, Ch06]. In general, three main steps are needed for information extraction, namely; *text preprocessing*, *named entity recognition*, and *entity linking* (relationships between named entities). Named entity (NE) recognition is the task of identifying and classifying predefined types of named entities, like persons, location, etc. [YB18, BKL09, NS07]. In general, there are two approaches of named entity recognition [NS07]: *rule-based* and *statistical-based* approaches. In the case of rule-based approaches, manually constructed rules like regular grammars are used. Gazetteer-based annotation technique (string matching) is also an element of the rule-based toolkit. Using statistical methods of named entity recognition makes it possible to derive such rules based on *training data*. Such statistical models are general applications of *machine learning*. In these approaches, text chunk labeling is considered as a classification task and several algorithms can be used for this task, such as conditional random fields, supervised learning techniques like SVM [CL11] and deep learning [Sh17].

3 Proposed framework: An overview

To deal with the extraction of entities and relations between entities from the ecological domain, we propose a new approach. The main idea of the proposed approach is to exploit semantic information represented in domain-specific ontologies. The main components of proposed approach are depicted in Fig. 1. The figure shows that the framework has two main components to extract entities and relations as well as necessary preprocessing steps. In the following and for the space limitation, we are going to focus on the entity extraction and recognition component.

Term extraction: As Figure1 shows the proposed framework accepts three kinds of inputs: (i) text from where entities should be identified and classified, (ii) domain information resources: gazetteers or ontologies, and optional (iii) domain expert knowledge. First, the proposed framework accepts a text corpus and applies a preprocessing step, i.e. tokenization, sentence splitting, and POS-tagging. This functionality is implemented as elements of

[3] https://www.biodiversity-exploratories.de/startseite/

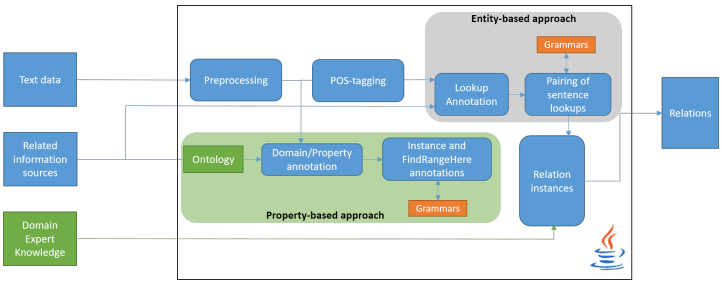


Fig. 1: Proposed framework

GATE corpus processing pipelines. Due to the size of the annotated XML files, we cannot store the whole text corpus in the main memory, as thus leads to its overflow even in the case of relatively small text set. To make this framework able to work with text corpus of any size, each document will be processed separately with new initialization of GATE resources. After preparing the input text, the next task is to extract terminologies related to the domain of interest. The term extraction process is a pattern matching problem. It may be solved using a *part of speech tagger* and a set of strict grammar rules in order to reflect the context. Following the definition in the book by [Ki14], we see that a term may be a noun phrase or a single word and also may be composed of nouns, adjectives, and prepositions. Therefore, a method to assign part-of-speech (POS) tags to tokens in textual data is used. The deployed method creates *term* annotations based on POS-tags using the grammar-based pattern matching. The expected result is a set of named entities with *term* label extracted from the given text corpus.

Keyness ranking. Once having the initial list of extracted terms, the next step is to filter it, because it will be a mix of general terms (non-specific) and domain specific terms which are needed. For each term, it is possible to compute a appearance frequency, but in the case of non-specific terms it will be always bigger than domain specific terms frequencies. Keyness ranking [Ki14] probably provides a solution for this problem. The main idea is relatively simple, we compute the frequencies of previously extracted terms in some general text corpus, also called *reference corpus*, $keyness_{term} = \frac{fpm_{focus}+n}{fpm_{ref}+n}$, where, fpm_{focus} is the term frequency normalized per million in focus corpus (our normal working corpus), fpm_{ref} is the term frequency normalized per million in reference corpus and n - smoothing parameter, by default $n = 1$. An excerpt of the result is shown Table 1.

Tab. 1: A sample of extracted terms along their keyness values

Term	beech	grassland	Microbiol	decomposer	fine root	rRNA	Soil
Keyness	653.6	641.8	564.35	525.03	421.25	351.73	344.34

Validation and filtering. Keyness ranking provides a good filtering already and helps with the selection of needed terms. But it is semantically blind, some of the extracted terms are still non-domain terms. To deal with this issue, we make use of the services provided by

BioPortal[4], which support the possibility to retrieve the definition for each term. Bioportal provides a comfortable terminology search API and returns requested information as JSON data. From such responses, we extract the definition of each term as well as the ontology that contains it. For example, here is the search for the term *soil structure*

```
    {
    "prefLabel": "structure of soil",
     - "synonym": [
        "soil structure"
    ],
     - "definition": [
        "The structure of some soil."
    ],
    "links": {"ontology":
    "http://data.bioontology.org/ontologies/AGRO",
...}
```

List. 1: Example of JSON response

This method allows us to get a list of domain ontologies for a given text corpus. Additionally, an automated semantic terminology validation/filtering may be implemented based on keywords in definitions. The appearance of word *soil* in retrieved definition speaks in favor that this term is related to the domain of soil science and so on.

Entity annotation. After preparing the input text for processing (token and sentence annotations), the next step is identify and recognize named entity. In the current implementation, we make use of two different schemes: (i) using classical gazetteers: we have a list of entities and search for them in given text data. Additionally, it may be improved with fuzzy string matching techniques. (ii) using ontology as an information resource, which requires some preparation before usage. To implement ontology-based annotation of named entities, we construct a little bit tricky architecture for GATE processing application. After the ontology is loaded as language resource, we construct two processing pipelines: one for ontology resource pre-processing (*RootFinder*) and another corpus pipeline to create annotations. RootFinder pipeline is here to prepare ontology-resources (related human-readable lexicalizations). The result set is stored in OntoRoot gazetteer module and then forwarded into Flexible gazetteer in corpus pipeline to make annotations based on extracted ontology resources.

4 Experimental evaluation

The proposed approach has been developed and implemented using Java 8 utilizing GATE 8.4.1[5] with embedded JAPE- for text annotation and grammars over annotations and Apache Jena 3.9.0- for ontology processing. To validate the performance of the approach, we carried out a set of experiments utilizing a corpus of 112 scientific works (articles, publications,

[4] http://bioportal.bioontology.org/
[5] https://gate.ac.uk/

theses, etc) from the ecological and environmental domains obtained from the Biodiversity Exploratory publication list. Originally they are in PDF format, and thus text data extraction was needed for next steps of work. Preprocessing like tokenization and sentence splitting are implemented as a part of GATE pipelines.

To evaluate the quality of the term extraction component, we asked domain experts from the soil from different scientific groups. We first run the term extraction process, selected the top-1000 terms and split them into four different sets, allowing overlap between sets. Then we asked domain experts to validate the set of extracted terms. Computing the precision of the available evaluations we get a precision of $precision_1 = 0.607$ for the first group, while the second group scores with a precision of 0.846. We believe that this initial and preliminary results are encouraging especially for this specific domain.

Keyness ranking. Here we consider the computation of keyness score on an example of the keyword *soil*. Before all, it is necessary to get normalized per million frequencies of the extracted keyword. In our working (focus) corpus there are about 2149404 tokens and *soil* occurs 16499 times. Hence, the normalized frequency is: $fpm_{focus} = \frac{16499 \cdot 1000000}{2149404} = 7676.08$. In the reference corpus, we may find the same term *soil* 3489 times, or 28.38 per million. In this regard, keyness score will be computed as follows: $keyness_{term} = \frac{fpm_{focus}+n}{fpm_{ref}+n} = \frac{7676.08+1}{28.38+1} = 261.302$. Here $n = 1$ is a smoothing parameter used to prevent division by zero if some term was not found in the reference corpus. A larger value of the keyness score (in comparison with other terms) speaks in favor that this term is domain specific. Using these computed values, we construct a terminology ranking table.

Search for domain-relevant ontologies. To achieve this task, we make use of BioPortal, which provides access to 774 ontologies (as of 28.01.2019). To find domain knowledge resources we used the top-2000 list of keyness-ranked domain terms. Technically it was relatively difficult to apply search API to the whole list, instead we have drawn three random samples with about 100 terms in each one. In settings of search process we established exact matching. Based on this, we create ranked lists of ontologies for each prepared terminology sample. Table 2 illustrates an example of this ranking. By using three samples we compute the average score and use it as a criterion for ontology selection. Note that very big ontologies like IOBC are less applicable in the context of this work due to technical limitations.

Tab. 2: Occurrence-based ranking of domain ontologies.

Ontology	Terminology samples			Average
	Sample 1	Sample 2	Sample 3	
IOBC	23	21	31	25,0
NCIT	13	15	19	15,7
NIFSTD	13	11	20	14,7
SNOMEDCT	13	7	15	11,7
CHEAR	10	11	12	11,0
NBO	8	9	11	9,3
AGRO	9	6	12	9,0
CRISP	8	10	8	8,7
ENVO	6	10	9	8,3
ECSO	6	9	8	7,7
MESH	6	8	9	7,7

5 Conclusion and future work

Many applications need extraction of named entities. To this end, we presented a framework that identifies and extracts entities from scientific publications from the ecological domain. The next step is to find not only named entities but also relations among entities. We have preliminary work on that.

6 Acknowledgements

We would like to thank A. Hildebrandt, FSU Jena and I. Schoening and T. Kloetzing, Max Planck Institute for Biogeochemistry, for evaluating extracted terms as domain experts. Also, we thank Andreas Ostrowski, FSU Jena, for providing the text corpus. This work has been mostly funded by the *Deutsche Forschungsgemeinschaft (DFG)* as part of CRC 1076 AquaDiva.

References

[BKL09] Bird, S.; Klein, E.; Loper, E.: Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, 2009.

[Ch06] Chang, Chia-Hui; Kayed, Mohammed; Girgis, Moheb R; Shaalan, Khaled F: A survey of web information extraction systems. IEEE TKDE, 18(10):1411–1428, 2006.

[CL11] Chang, Chih-Chung; Lin, Chih-Jen: LIBSVM: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3):27, 2011.

[DHI12] Doan, AnHai; Halevy, Alon; Ives, Zachary: Principles of data integration. Elsevier, 2012.

[Go18] Golshan, Parisa Naderi; Dashti, HosseinAli Rahmani; Azizi, Shahrzad; Safari, Leila: A Study of Recent Contributions on Information Extraction. CoRR, abs/1803.05667, 2018.

[Ho02] Hobbs, Jerry R.: Information extraction from biomedical text. Journal of Biomedical Informatics, 35(4):260–264, 2002.

[Ki14] Kilgarriff, Adam; Baisa, Vít; Bušta, Jan; Jakubíček, Miloš; Kovář, Vojtěch; Michelfeit, Jan; Rychlý, Pavel; Suchomel, Vít: The Sketch Engine: ten years on, volume 1. Jul 2014.

[KN04] Krauthammer, Michael; Nenadic, Goran: Term identification in the biomedical literature. Journal of Biomedical Informatics, 37(6):512–526, 2004.

[NS07] Nadeau, David; Sekine, Satoshi: A survey of named entity recognition and classification. Investigationes, 30(1):W3–W26, 2007.

[Sh17] Shen, Yanyao; Yun, Hyokun; Lipton, Zachary Chase; Kronrod, Yakov; Anandkumar, Animashree: Deep Active Learning for Named Entity Recognition. In: Rep4NLP@ACL. pp. 252–256, 2017.

[YB18] Yadav, Vikas; Bethard, Steven: A Survey on Recent Advances in Named Entity Recognition from Deep Learning models. In: COLING 2018. pp. 2145–2158, 2018.

Studierendenprogramm

Automated Architecture-Modeling for Convolutional Neural Networks

Manh Khoi Duong[1]

Abstract: Tuning hyperparameters can be very counterintuitive and misleading, yet it plays a big (or even the biggest) part in many machine learning algorithms. For instance, finding the right architecture for an artificial neural network (ANN) can also be seen as a hyperparameter e.g. number of convolutional layers, number of fully connected layers etc. Tuning these can be done manually or by techniques such as grid search or random search. Even then finding optimal hyperparameters seems to be impossible. This paper tries to counter this problem by using bayesian optimization, which finds optimal parameters, including the right architecture for ANNs. In our case, a histological image dataset was used to classify breast cancer into stages.

Keywords: CNN; Model Architecture; Breast Cancer; Histology

1 Introduction

Hyperparameters in machine learning are properties of an algorithm. There exist many hyperparameters and especially neural networks have numerous of them. An essential task is to find the hyperparameters which lead to the best results of the proposed classifier. Techniques such as *grid search* and *random search* [BB12] can be proposed to solve this task but lack of the ability to learn from previous outcomes. This is where *Bayesian optimization* [Mo75] comes into play. This paper describes how Bayesian optimization can be used to counter the problem of finding the right model architecture for Convolutional Neural Networks to yield for the best results. The final evaluation of the proposed method in this paper was done on a histological dataset [Ar17].

2 Related works

Similar works whose task is to optimize the model architecture are [ZL16], [Li17] and [Zo17]. The mentioned works use *reinforcement learning* to consider the results of the previous architectures to find better ones after each step. A *controller*, which is equivalent to an agent, trains a *child network* each step. The feedback scores for the controller are the

[1] Heinrich-Heine-Universität, Datenbanken und Informationssysteme, Universitätsstraße 1, 40225 Düsseldorf, Germany manh.duong@hhu.de

doi:10.18420/btw2019-ws-17

validation accuracies. This guides the controller to produce better models at each iteration. In contrast to this paper, the methods in [ZL16], [Li17] and [Zo17] use recurrent neural networks where hyperparameters have to be set, too. Differently, the methods used in this paper require only two hyperparameters which are the number of models to be trained and the choice of the *acquisition function*. The trade-off seems to be better as fewer settings have to be considered to optimize a function which requires much more settings.

3 Foundation

Bayesian optimization is a global optimization algorithm and works by firstly making a *prior belief* about the *objective function*[2]. Then a prediction is made to find the minimum or maximum. After observing the current evidence, the *posterior belief* is updated by the given evidence and the prior belief. This update step is repeated for a defined number of iterations so that the uncertainty decreases and the objective function can be approximated. Assuming that each point of the function is normally distributed, then this function can be modeled with *Gaussian process (GP)* regression. It can target multiple variables, thus multiple hyperparameters can be optimized.
To determine the next point x_{next} for an observation X, an acquisition function is required. This function usually noted by $a : X \rightarrow \mathbb{R}^+$, with set of points X, takes all observations and the best result into account [BCdF10]. It also has to be maximized to select the next point x_{next} to evaluate the function at e.g. $x_{next} = argmax_x a(x)$. Solving another optimization problem than the objective function seems to be impractical but the acquisition function is usually easier to optimize and much cheaper. It also limits the regression task of the objective function: not all points are going to be evaluated, only points which are important to find the minimum or maximum. An acquisition deals with the trade-off between *exploration* and *exploitation*. Exploration is discovering regions with higher uncertainty and exploitation is sampling on regions that will likely offer an improvement [BCdF10]. Exploration is needed to escape from the local optimum.

Consider the hyperparameter optimization task with n hyperparameters, the objective function is the function that samples all hyperparameter settings. Each point is n-dimensional and contains values for each hyperparameter. The objective function has to be maximized because a high accuracy is yielded. This can be formalized as $argmax_x f(\vec{x}, D, y, D_{val}, y_{val})$ where x is the n-dimensional vector containing hyperparameter values, D is the training data, D_{val} is the validation data, y and y_{val} are labels with f being a classifier which returns the validation accuracy. The search space can be defined as intervals, making it continuous as in random search.

[2] A function to minimize or maximize

4 Methods

This chapter aims to explain the method of how hand tuning has been replaced. The proposed method can be divided into two steps: finding the model and fine-tuning the model. The first subsection deals with the aspect which models can be found and the second subsection deals with the details of fine-tuning the best model that has been found. This method has been split in two because searching for all hyperparameters at once is not practical considering the exponential growth of the search space.

4.1 Model finding

As already said in Section 3, the classifier can be seen as a function f which takes $\vec{x}, D, y, D_{val}, y_{val}$ as inputs and returns the validation accuracy. Due to the fact that the used python package [Sc] minimizes the objective, the optimization problem is $argmin_x \widetilde{f}(\vec{x}, D, y, D_{val}, y_{val})$ with $\widetilde{f}$ being the classifier which returns the negative validation accuracy. Note that the goal is to find $\vec{x}$.

The objective is called by the bayesian optimization algorithm which can be seen in Algorithm 1 [BCdF10]. The attended hyperparameter $\vec{x}$ and its linked validation_accuracy $= F(\vec{x}, D, y, D_{val}, y_{val})$ are being saved to $\widetilde{D}$ which is not to be confused with the training or validation data. The Gaussian process regression is then updated in the last step. The best hyperparameters can then be observed after n iteration steps of calling the objective with different $\vec{x}$. This optimization method does not have many hyperparameters itself. Only the number of calls and the acquisition function is needed. For this paper, Expected Improvement was chosen as the acquisition function and is defined for the minimization task as follows [JSW98]:

$$EI(x) = \mathbb{E}[max\{0, f(\widetilde{x}) - f(x)\}]$$

where $\widetilde{x}$ is the best hyperparameter that has been observed so far. This formula expresses that the current $f(x)$ has to be smaller than $f(\widetilde{x})$ and their difference should be maximized. The expected improvement can then be obtained by the expected value.

The model architectures proposed by this algorithm follow the trend of how modern convolutional neural networks are built (see comparison with VGG16, VGG19 [SZ14], DenseNet [Hu17]). The created model architectures are sequential and nonrecurrent. As seen in Figure 1, the first convolutional block can be of size m, meaning it can contain m convolutional (Conv) layers activated with the ReLu function. The second Conv block can be of size n. The batch normalization (BN) layer was marked differently in the Figure as the use of it was set to be a hyperparameter. Convolutional layers that are contained in the same block have equivalent output dimensions and filter sizes. After several additional p Conv blocks, the output is fed to k dense layers. A dropout layer is added between each dense layer. The dense layers were also activated with the ReLu function. The last layer is

Algorithm 1 Bayesian Optimization

procedure BAYESOPT($\vec{x}, D, y, D_{val}, y_{val}$)
  for $i \leftarrow 1, n$ **do**
    $x_{\vec{next}} \leftarrow argmax_{\vec{x}} acq(\vec{x}, \widetilde{D})$ ▷ acq is the acquisition function
    val_accuracy $\leftarrow \vec{f}(x_{\vec{next}}, D, y, D_{val}, y_{val})$
    if val_accuracy < best_accuracy **then**
      best_accuracy $\leftarrow$ val_accuracy
      $x_{\vec{best}} \leftarrow x_{\vec{next}}$
    end if
    $\widetilde{D} \leftarrow \widetilde{D} \cup (x_{\vec{next}}$, val_accuracy)
    Update Gaussian process regression based on $\widetilde{D}$
  end for
  return $x_{\vec{best}}$
end procedure

the softmax function which produces probabilities for the final output. Note that after each Conv Block, a maxpooling (MP) layer is added consecutively. The MP layer reduces the shape it receives into a much smaller shape. Too many MP layers can lead to an exception of negative output shapes. The maximum amount of Conv blocks is therefore limited by the very first input shape. Larger image sizes can have deeper models because more MP layers are allowed. The algorithm developed for this paper is able to adapt the maximum amount of Conv blocks to be created based on the input shape. To prevent underfitting, the parameters m, n, p (see Figure 1) are at least one. Differently, it is possible that the created neural network can have zero additional dense layers ($k = 0$).

4.2 Fine-tuning

The fine-tuning was applied on the model that has achieved the best results on the model finding algorithm. The algorithm in fine-tuning is the same as described in Algorithm 1. The only difference is that the objective gets different hyperparameters. In the fine-tuning process a total of eight hyperparameters were tuned. The output dimension of the first, second and other Conv blocks make up three hyperparameters. The rest consists of the learning rate, dropout, number of dense nodes, lambda value of the L2-regularization and the kernel size of each Conv filter. All other settings that are not mentioned were preserved from the model finding task e.g. the optimizer and activation function were not changed. The hyperparameters which are to be optimized in this task can be easily switched up with other hyperparameters available. The hyperparameter vector $\vec{x}$ and the search space could be defined arbitrary.

After finding the right model and fine-tuning the model, it was aimed to train the model further to achieve even better results.

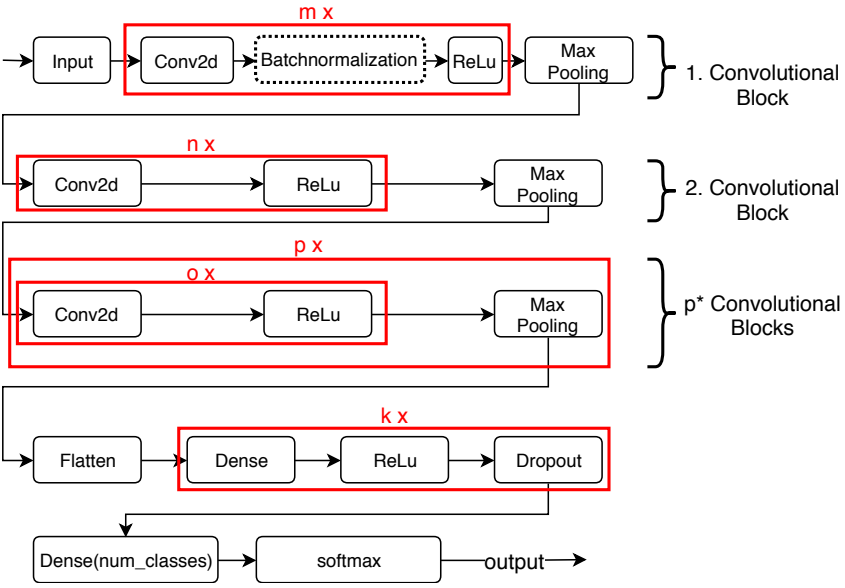
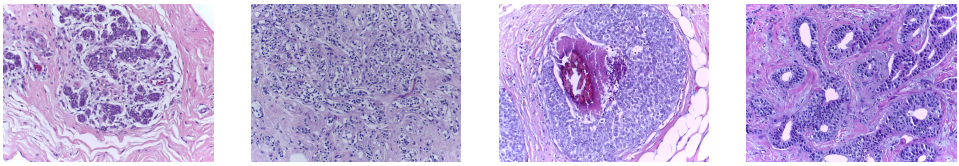


Fig. 1: Model architectures that can possibly be created

5 Evaluation

The application of the methods was done on a histological dataset of breast cancer (see Figure 2). The dataset provided by [Ar17] consists of 249 training samples and 36 test samples labeled in the following four classes: *normal, benign, in situ carcinoma* and *invasive carcinoma*. This leads to a four-class classification task. Another task that comes up by this dataset is the classification into *carcinoma* and *noncarcinoma* cells where *normal, benign* are noncarcinoma and *in situ, invasive* are carcinoma. Each image has a resolution of 2048×1536 pixels. The breast tissues were stained with haematoxylin and eosin (H&E) and were digitized under the same conditions. The magnification on this tissue is $200\times$. The provided test set contains 20 images where the images can be classified clearly (labeled as *initial* in [Ar17]) and 16 additional images with increased ambiguity. The latter were labeled as *extended*. The classes have almost evenly distributed samples.
Because neural networks require a high amount of training samples, the dataset was augmented with flippings, mirrorings, rotations and random contrast changes. The augmentation increased the number of images by eight times. This results in 1992 samples to train the neural network.



(a) Normal (b) Benign (c) In Situ (d) Invasive

Fig. 2: Samples of each stage

5.1 Inspecting bayesian optimization

Due to the use of bayesian optimization to find the optimal hyperparameters, its accomplishment in this task can be studied further in this section. From a theoretical point of view, this optimization algorithm should find better values after each evaluation of the objective function and a convergence should be seen. To see how this optimization technique performed, a plot (see Figure 3) has been made which shows the accuracy of each created model in the y-axis and the nth model in the x-axis. It was set that each created model has to be trained for 40 epochs. Note that the fine tuning process was not done yet in this figure and only the created models are considered.

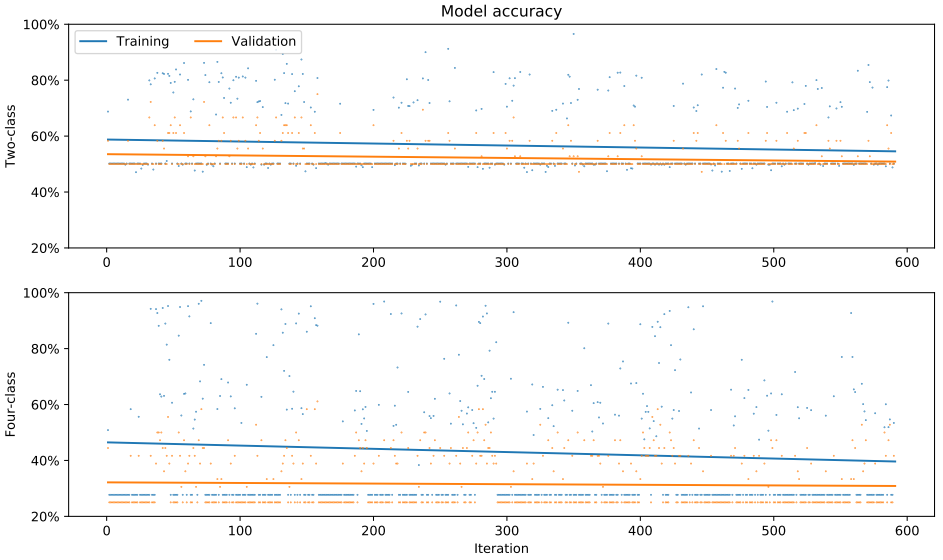


Fig. 3: Accuracy of the created models on both classes. A linear regression for each validation and training accuracy describes how bayesian optimization finds optimal models.

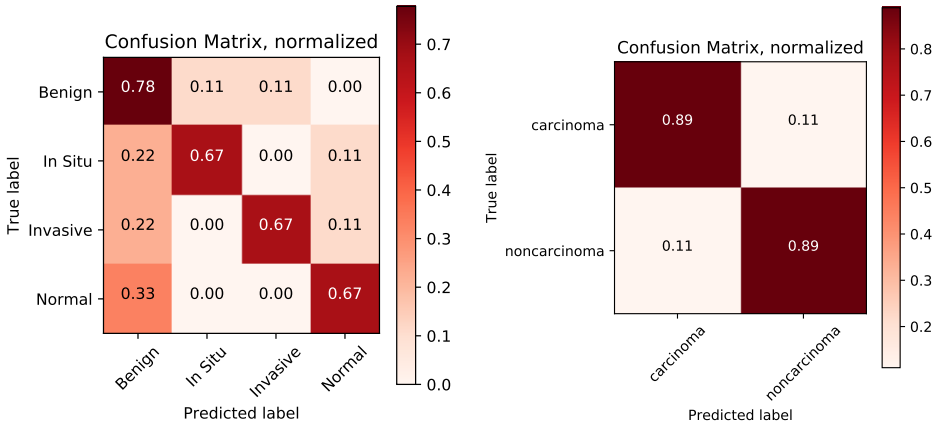
Considering the two-class classification, the values in the y-axis ranges from slightly under 50% to 100%. This is because guessing in a binary classification problem will already

result in an accuracy of 50% on average if the dataset is evenly distributed which is fulfilled in our given dataset. A lot of models that were created, scored bad accuracies. Meaning those models were not able to learn from the data. This can be seen in the figure where a lot of points lie in the accuracy of 50% which nearly creates a horizontal line visually. The same can be observed in the four-class classification where the threshold is 25%. If the plot as a whole is considered, there is no evidence of a convergence obtainable. All points appear to be randomly distributed. The linear regression even shows a decrease in both training and validation accuracy with each model that is found. This contradicts with the theoretical assumption that better models should be found. For both classification tasks, the best validation accuracy was found after 157 iterations. The achieved validation accuracy in the four-class task is 61%. For the two-class task, the achieved validation accuracy is 75%. This might be caused by randomly bad weight initializations or by the exploration step that was done too much.

5.2 Final results

The best model that was found as described in Section 5.1 was fine-tuned and trained for additional epochs. 1000 different hyperparameter settings were tested and it was chosen to let the model train for 60 epochs with each hyperparameter setting. The results which are presented in the following are based on the validation set: The confusion matrices in Figure 4 are normalized which means that the number of predictions were divided by the total number of images per class. This results in class-wise recall scores on the diagonal. The confusion matrix in Figure 4a) shows on the diagonal that nearly all classes were predicted correctly with a probability of 67%. Only benign has a recall score of 78%. It can be misleading to state that benign tissues get classified at best because the first column shows that benign was predicted at most and therefore the ANN was already likely to score best at it. The higher number of predictions on the benign class can be explained by the slightly unbalanced class distribution as it makes up 28% of the training data. The precision score for the benign class is only 50%. Because normal and benign belong to non-carcinoma cells, the most confusion happened there: $\frac{1}{3}$ of the normal tissues were labeled as benign but the opposite did not happen. The confusion matrix in 4b) shows that the classifier stages carcinoma and non-carcinoma tissue both with a probability of 89%. As seen, the false positive rate and false negative rate is 11%.

Table 1 shows different metrics to evaluate the classifiers. As seen, the metrics score for two classes is the same everywhere. This is due to the equally distributed dataset and - as seen in Figure 4b) - that all classes were fortunately predicted right with the same probability. For non-binary classes there exists different averaging methods for the measures. The chosen averaging method is *macro* which computes the given measure for each class and then takes the arithmetic mean of it. As seen, the precision score is higher than the recall score for four classes. The neural network is therefore good at classifying into the right classes but misses a few elements which belong to it.



(a) Four-class (b) Two-class

Fig. 4: Confusion matrices of the ANN on both classification tasks

Classes	Accuracy	Recall	Precision	F1-score
Two-class	89%	89%	89%	89%
Four-class	69%	69%	74%	70%

Tab. 1: Different metrics to evaluate the ANNs

5.3 Comparing with related works

The following comparison is done against [GAS18], [Na18] and [NAE18] who made their papers publicly available. The mentioned works participated in the *ICIAR 2018 Grand Challenge on Breast Cancer Histology Images* [Ar18]. The dataset provided in this challenge is based on the dataset [Ar17] used in this paper but consists of 151 more samples which is an advantage. The dataset provided by the challenge was not used here because the dataset was only made available for the participants. The comparison can be seen in Table 2 and all results are based on the validation accuracy.

Team	4-class acc.	2-class acc.	Approach
Aditya et al. [GAS18]	85%	93%	Transfer learning: Inception-v3 [Sz15]
Wajahat et al. [Na18]	81%	-	Transfer learning: AlexNet [KSH12]
Kamyar et al. [NAE18]	95%	-	Transfer learning: Inception-v3 [Sz15]
Own work	**69%**	**89%**	**Automated Architecture-Modeling**

Tab. 2: Validation accuracies compared with similar works

As seen in Table 2, not every team did the two-class classification. The comparison for two classes can only be done against Aditya et al. [GAS18]. It can be observed that the results are very similar in this case. For four classes, this work achieved inferior results against the participants of the challenge. This can be caused by the higher amount of samples the

teams had or by the use of a very common method of *transfer learning*. Though transfer learning has been used by every mentioned team, the results can still differ. The results can even vary if the teams use the same base model ([NAE18] and [KSH12]). As the number of samples is essential for neural networks, the differences in the results can likewise be caused by the different preprocessing and data augmentation methods. Apparently the validation set is not the same for every team which can also lead to an inaccurate comparison. [GAS18] and [Na18] used a fixed validation set throughout the hyperparameter search whereas [NAE18] used cross-validation. The split was set differently by every participant. The results, therefore, should be obtained with caution.

6 Conclusion and future works

Though the presented methods were applied on only one dataset consisting of two tasks, the results were quite decent as an accuracy of 89% was achieved in the binary classification task. For the four-class task, an accuracy of 69% was achieved. Also, it has been shown that the real application of bayesian optimization neglects the intuitive results as no convergence could be observed. This can be caused by many reasons, a further inspection and application of the methods could be done in future works.

Furthermore, the model finding method could be extended by allowing the creation of recurrent and nonsequential models. Considering more hyperparameters could lead to a closer goal to the global optimum. Though the problem might be the curse of dimensionality when considering too many possibilities, a bypass is to split the task into more steps instead of two. The model finding algorithm could also be optimized: Some models already lack to learn from the data and further training is not required. Some models, however, can accomplish better results when being trained on a very high amount of epochs. When considering the number of epochs as the depth and the amount of models that are going to be created as the breadth, an application of *iterative deepening depth-first search* which combines the *breadth-first* and *depth-first* search can thus be applied.

References

[Ar17] Araújo, Teresa; Aresta, Guilherme; Castro, Eduardo; Rouco, José; Aguiar, Paulo; Eloy, Catarina; Polónia, António; Campilho, Aurélio; et al: Classification of breast cancer histology images using Convolutional Neural Networks. PLOS ONE, 12(6), 2017.

[Ar18] Aresta, Guilherme; Araújo, Teresa; Kwok, Scotty; Chennamsetty, Sai Saketh; P., Mohammed Safwan K.; Varghese, Alex; Marami, Bahram; Prastawa, Marcel; Chan, Monica; Donovan, Michael J.; Fernandez, Gerardo; Zeineh, Jack; Kohl, Matthias; Walz, Christoph; Ludwig, Florian; Braunewell, Stefan; Baust, Maximilian; Vu, Quoc Dang; To, Minh Nguyen Nhat; Kim, Eal; Kwak, Jin Tae; Galal, Sameh; Sanchez-Freire, Veronica; Brancati, Nadia; Frucci, Maria; Riccio, Daniel; Wang, Yaqi; Sun, Lingling; Ma, Kaiqiang; Fang, Jiannan; Koné, Ismaël; Boulmane, Lahsen; Campilho, Aurélio; Eloy, Catarina; Polónia, António; Aguiar, Paulo: BACH: Grand Challenge on Breast Cancer Histology Images. CoRR, abs/1808.04277, 2018.

[BB12] Bergstra, James; Bengio, Yoshua: Random Search for Hyper-parameter Optimization. J. Mach. Learn. Res., 13:281–305, 2012.

[BCdF10] Brochu, Eric; Cora, Vlad M.; de Freitas, Nando: , A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, 2010.

[GAS18] Golatkar, Aditya; Anand, Deepak; Sethi, Amit: Classification of Breast Cancer Histology using Deep Learning. CoRR, abs/1802.08080, 2018.

[Hu17] Huang, G.; Liu, Z.; v. d. Maaten, L.; Weinberger, K. Q.: Densely Connected Convolutional Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2261–2269, 2017.

[JSW98] Jones, Donald R.; Schonlau, Matthias; Welch, William J.: Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization, 13:455–492, 1998.

[KSH12] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E: ImageNet Classification with Deep Convolutional Neural Networks. In (Pereira, F.; Burges, C. J. C.; Bottou, L.; Weinberger, K. Q., eds): Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc., 2012.

[Li17] Liu, Chenxi; Zoph, Barret; Shlens, Jonathon; Hua, Wei; Li, Li-Jia; Fei-Fei, Li; Yuille, Alan L.; Huang, Jonathan; Murphy, Kevin: Progressive Neural Architecture Search. CoRR, abs/1712.00559, 2017.

[Mo75] Močkus, J.: On bayesian methods for seeking the extremum. Springer Berlin Heidelberg, pp. 400–404, 1975.

[Na18] Nawaz, Wajahat; Ahmed, Sagheer; Tahir, Muhammad; Khan, Hassan: Classification Of Breast Cancer Histology Images Using ALEXNET. pp. 869–876, 06 2018.

[NAE18] Nazeri, Kamyar; Aminpour, Azad; Ebrahimi, Mehran: Two-Stage Convolutional Neural Network for Breast Cancer Histology Image Classification. In: International Conference Image Analysis and Recognition. Springer, pp. 717–726, 2018.

[Sc] Scikit-optimize. Website. Online available: `https://github.com/scikit-optimize/scikit-optimize`; visited 29/07/2018.

[SZ14] Simonyan, Karen; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556, 2014.

[Sz15] Szegedy, Christian; Vanhoucke, Vincent; Ioffe, Sergey; Shlens, Jonathon; Wojna, Zbigniew: Rethinking the Inception Architecture for Computer Vision. CoRR, abs/1512.00567, 2015.

[ZL16] Zoph, Barret; Le, Quoc V.: Neural Architecture Search with Reinforcement Learning. CoRR, abs/1611.01578, 2016.

[Zo17] Zoph, Barret; Vasudevan, Vijay; Shlens, Jonathon; Le, Quoc V.: Learning Transferable Architectures for Scalable Image Recognition. CoRR, abs/1707.07012, 2017.

Chain-detection for DBSCAN

Janis Held [1], Anna Beer [2], Thomas Seidl [2]

Abstract:

Chains connecting two or more different clusters are a well known problem of the probably most famous density-based clustering algorithm DBSCAN. Since already a small number of points resulting from, e.g., noise can form such a chain and build a bridge between different clusters, it can happen that the results of DBSCAN are distorted: several disparate clusters get merged into one. This single-link effect is rather known but to the best of our knowledge there are no satisfying solutions which extract those chains, yet. We present a new algorithm detecting not only straight chains between clusters, but also bent and noisy ones. Users are able to choose between eliminating one dimensional and higher dimensional chains connecting clusters to receive the underlying cluster structure by DBSCAN. Also, the desired straightness can be set by the user. We tested our efficient algorithm on a dataset containing traffic accidents in Great Britain and were able to detect chains emerging from streets between cities and villages, which led to clusters composed of diverse villages.

Keywords: DBSCAN, clustering, chain-detection, single link effect

1 Introduction

The human eye can easily detect areas of high density within a set of points. Derived from this human intuitive clustering method the basic idea behind density-based clustering is finding clusters by detecting areas of high density. The famous density-based algorithm DBSCAN [Es96] builds clusters around points with high density, so-called seed points, and expands them taking all density-connected points into account as described in Section 2. As long as the clusters are clearly separated, this procedure works very well but if there are e.g. some density-connected noise points creating a chain between clusters, DBSCAN expands the cluster along these chains resulting in a single huge cluster instead of the intuitive ones.

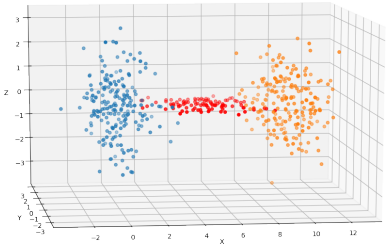


Fig. 1: The red points cause a density-connection between the intentional two clusters and thus form a chain.

[1] Ludwig- Maximilians- Universität München, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany, J.Held@campus.lmu.de

[2] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany, {beer, seidl}@dbs.ifi.lmu.de

doi:10.18420/btw2019-ws-18

While keeping the requirements of DBSCAN, like minimal domain knowledge to determine the input parameters, discovering clusters of arbitrary shape and good efficiency on large databases, we developed an algorithm which detects such chains in clusters found by DBSCAN. For that we use PCA (Principal Component Analysis) assuming that a chain has a lower dimensionality than the clusters it connects. Figure 1 shows an example where two 3D clusters are connected by a red chain with only little expansion in two of the three dimensions. Our algorithm is adaptable, users can choose which type of chains they want to connect: straight chains or bent ones, noisy or thin ones. Through recognizing those chains and eliminating them from the clustering the underlying individual clusters can be revealed by DBSCAN.

The paper is structured as follows: First, we introduce shortly the related work and basics we use in Section 2. In Sections 3 we explain our novel method to find chains in detail, giving an overview over the whole algorithm in Section 3.6. We analyze the complexity in Section 4 and prove its effectiveness in Section 5 with some experiments. In Section 6 we conclude and give a brief idea of some future work.

2 Related Work and Basics

There are already many extensions of DBSCAN, e.g. ST-DBSCAN, an extension for clustering spatial-temporal data [BK07], MR-DBSCAN, which is an efficient parallel density-based clustering algorithm using map-reduce [He11], or C-DBSCAN: Density-based clustering with constraints [RSM07]. To the best of our knowledge, there is yet no extension of DBSCAN to circumvent the disadvantages of the single-link effect or chains connecting clusters. In this section, we give the basics needed for the following sections, namely some details of DBSCAN and the Principal Component Analysis (PCA).

DBSCAN Density-based spatial clustering of applications with noise [Es96] is a density based clustering algorithm that clusters points based on their density and marks outliers lying in low-density regions. A point with at least $minPts$ points in its ϵ-range is called a core point. All points in the ϵ-range of a core point c belong to the same cluster as c and are called density-reachable from c. All reachable points are assigned to the cluster from which they are reachable, while points which are neither reachable nor core points are declared noise. Like that, it is possible that a small chain of density-reachable points connects two clusters as Figure 2 shows.

PCA (Principal Component Analysis) [JC16] transforms given data points to a new coordinate system where the greatest variance by any projection of the data lies along the first coordinate (the first principal component), the second greatest variance along the second coordinate, and so on. PCA is a good indicator of how well some data fits into a lower dimensional subspace.

PCA regards the eigenvalue decomposition of the data covariance matrix, usually after mean centering the data matrix for each dimension. Then the eigenvectors of the covariance matrix form an orthogonal basis and each eigenvalue describes how much variance is explained by its corresponding eigenvector [JC16].

Let d be the dimensionality of the data space Ω and $N = \{n_1, ..., n_m\}$ the ϵ range of some point $p \in \Omega$. The data matrix is defined as $(n_1, ...n_m)^T$. Let m_j be the mean of column j. One can now calculate the covariance matrix Θ with

$$\Theta_{ij} = \frac{\sum_{k=1}^{m}(n_{ki} - m_i)(n_{kj} - m_j)}{m}, \quad i = 1, .., d, \quad j = 1, .., d. \tag{1}$$

Note that the covariance matrix is symmetric and positive semi-definite, thus its eigenvalues are non negative. Finally the eigenvalues are normalized by dividing them by the sum of all eigenvalues, such that the sum of all normalized eigenvalues equals to 1.

3 The Approach

Let $DBSCAN_{\epsilon,minPts}(X)$ be the clustering of DBSCAN with parameters ϵ and $minPts$ of some data X and C be a cluster found by DBSCAN in the data space Ω. We want to find a set of candidates that may form chains in C. With the assumption of chains having a subdimensional shape we can utilize the definition of neighborhood from DBSCAN and look for an algorithm that decides for each point if it lies within a subdimensional neighborhood. Additionally the algorithm has to fulfill some restraints: first of all it has to be rotation invariant as the direction of the chains does not matter. Secondly it has to be error resistant, as we want to be able to allow some bending of chains and apply it on a application with noise. The idea is to use the distribution of all points in the ϵ-range of each point as an indicator for its likelihood do be part of a chain. Therefore, a point in C is considered a **shape-based chain-point candidate** if there exists a subspace with a lower dimensionality than Ω, such that all points of the ϵ range of p lie close to it. Note that "lower dimensionality" and the word "close" will become parameters for the chain-detection algorithm. Clustering all remaining points may result in some noise points. We call the union of shape-based chain-point candidates with all those noise points **chain-point candidates**. Now we can cluster the chain-point candidates and each cluster is called a **chain-candidate**. Note that all chain-point candidates which were marked as noise are not part of a chain-candidate, because we want a chain to be at least big and dense enough to form a cluster itself. The last step will be to validate if the chain-candidate indeed connects two clusters of the remaining points and is not some kind of tail.

3.1 Chains

Fig. 2: The red dots connect two clusters and thus form a chain.

Fig. 3: Since the chain-like looking red dots do not connect any clusters, they are not considered a chain.

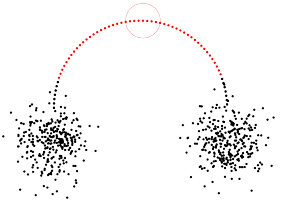


Fig. 4: The red dots may or may not be a chain, depending on the user. The red circle is one of the ϵ ranges.

Assume the user wants to detect one-dimensional chains in a two-dimensional data space and *DBSCAN* would not label the red dots in the following figures as noise, then Figure 2 shows a simple example of a chain. The red dots in Figure 3 are not considered a chain, because they do not form a connection between two clusters. The red dots in Figure 4 are not perfectly linear, because the ϵ range of each red point (the red circle is one of the ϵ ranges) does not perfectly fit inside a one dimensional subspace, and thus it depends on the user if he wants to detect those as a chain.

3.2 Chain-Point candidates

For each point in a cluster C the objective is to determine if this point is a chain-point candidate. To achieve this, for each point $p \in C$ the technique behind principal component analysis (PCA) is utilized to calculate how good the ϵ range of p fits inside a subspace with a dimensionality lower than the dimensionality of the data space Ω. To be more precise, PCA is utilized to find this subspace and then to calculate the explained variation of those ϵ-neighbors of p which do not fit inside this subspace.

Theorem 1 *Let d be the dimensionality of the data space Ω and $N = \{n_1, ..., n_m\} \subset \Omega$ be the ϵ range of some point $p \in \Omega$. Furthermore let $\lambda_1 \geq ... \geq \lambda_d$ be the sorted normalized eigenvalues of the covariance matrix Θ derived from N.*

1. *If $\lambda_d = 0$, then N lies inside a hyperplane.*
2. *If $\lambda_d = 1/d$, then N is perfectly distributed across all dimensions.*
3. *if $\lambda_i = 0$ and $1 < i < d$, then N lies inside a subspace with dimension $i - 1$.*

Proof 1

1. *If $\lambda_d = 0$, then the corresponding eigenvector ev_d describes 0 variance. Since the eigenvectors form a orthogonal basis N lies entirely in the hyperplane orthogonal to ev_d.*
2. *Since the sum of all eigenvalues equals to 1 and there are d eigenvalues and all are non negative, each eigenvalues must be equal to $1/d$. That means each eigenvector describes the same variance, thus N is perfectly distributed across all dimensions.*
3. *Since the eigenvalues are sorted, non negative and $\lambda_i = 0$ it follows that $\lambda_j = 0, \forall j \in \{i, ..., d\}$. That means the corresponding eigenvectors $ev_j, j \in \{i, ..., d\}$ of the orthogonal basis describe 0 variance. Thus, N lies entirely in the subspace spanned by $ev_j, j \in \{1, ..., i-1\}$.*

3.3 Parameters

With theorem 1 one can now define two parameters

1. $chainDim \in \{1, ..., d-1\}$, which describes the dimensionality of chains the user wants to detect.
2. $allowedVariation \in [0, 1[$, which allows variation beyond the allowed dimensionality of the chain.

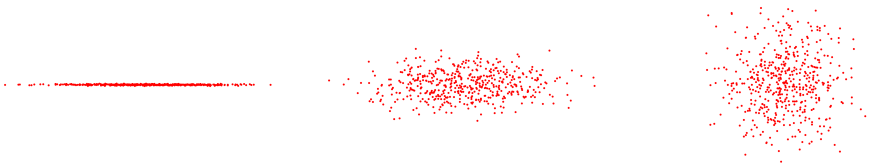
Like in Section 3.2, let $N = \{n_1, ..., n_m\}$ be the ϵ range of some point $p \in C$ and $\lambda_1, ..., \lambda_d$ the descending sorted normalized eigenvalues of the covariance matrix Θ corresponding to N. To calculate how good N lies within a $chainDim$ dimensional subspace, one calculates the accumulated error $e := \sum_{i=chainDim+1}^{d} \lambda_i$. The sum starts with $chainDim + 1$, because only the $d - chainDim$ least significant principal components explain the variation beyond the wanted chain dimensionality. It holds that $\lambda_d \in [0, 1/d]$, because the sum of all eigenvalues equals to 1, there are d eigenvalues and λ_d is the smallest one. If $\lambda_d < 1/d$ then $\lambda_1 > 1/d$, otherwise λ_1 would not be the largest normalized eigenvalue. That means the sum of the i smallest normalized eigenvalues is at most i/d, that is if all eigenvalues are $1/d$. Thus $e \in [0, (d - chainDim)/d]$ To make the user-input independent of the dimensionality of Ω and $chainDim$, one normalizes the error by

$$\bar{e} := e * \frac{d}{d - chainDim} \in [0, 1]. \tag{2}$$

Now, p is a chain-point candidate if $\bar{e} \leq allowedVariation$.

3.4 Fuzziness of Chains

In Figure 5 examples for various values of normed errors are given for a two dimensional data space with $chainDim = 1$. $\bar{e}$ describes the variation beyond a linear subspace. The closer the points get to a linear subspace the lower the error gets and vice versa. In Figure 5c the error is close to 1 since the points are almost perfectly distributed in all directions.



(a) $\bar{e} \approx 0.0002$ (b) $\bar{e} \approx 0.1563$ (c) $\bar{e} \approx 0.9997$

Fig. 5: Various degrees of fuzziness dependent on the normed error $\bar{e}$

Let us have a look at some synthetic example data. In Figure 6 the points are colored by its normed error values with $chainDim$ set to 1. Some points are clearly marked red, because they have a low normed error, indicating that they might be part of a chain. On the other hand most of the points inside those clouds have a high normed error because their ϵ range hardly fits into a one-dimensional subspace. Setting $allowedVariation$ to some value determines for each point if it is a chain-point candidate. Setting $allowedVariation$ to 0.2 on the data of Figure 6 results in the shape-based chain-point candidates seen in Figure 7.

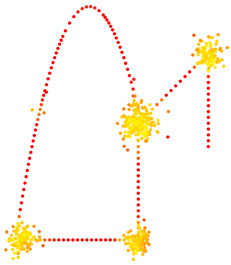


Fig. 6: Example data: Each point is colored by the normed error $\bar{e}$ derived from its ϵ range. Yellow means the error is close to 1 and red means it is close to 0.

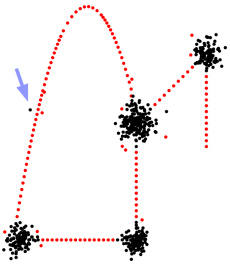


Fig. 7: Example data: With $allowedVariation = 0.2$ the red points are selected as shape-based chain-point candidates. The arrow highlights an outlier.

3.5 Finding and validating chain candidates

Let $C_{\bar{e}}$ be the set of shape-based chain-point candidates. First of all each shape-based chain-point candidate is added to the set of chain-point candidates. After clustering the remaining points $C \setminus C_{\bar{e}}$ by $DBSCAN_{eps,minPts}$ all points marked as noise are not part

of a cluster of non-candidates, indicating that they also might be part of a chain, see the highlighted black dot on the left of Figure 7. These points are now added to the set of chain-point candidates.

Clustering the set of chain-point candidates by $DBSCAN_{eps,minPts}$ results in clusters of chain-point candidates, which are the desired **chain-candidates** and noise.

Let $C_{ci}, i \in I$ be those chain-candidates, $R := C \setminus \cup_{i \in I} C_{ci}$ be the set of the remaining points and DB_R be $DBSCAN_{eps,minPts}(R)$. Note that R contains those chain-point candidates, which were marked as noise by clustering all chain-point candidates. To validate C_{ci} check for each point $p \in C_{ci}$, if their ϵ range contains points $r \in R$ and note the cluster of r found in the clustering DB_R. As soon as two clusters are noted the chain is validated and considered a chain. If all points are checked but no two clusters are noted the chain-candidate C_{ci} could not be validated and is not considered a chain.

Finally we receive a set of chains - which can now be considered clusters themselves or simply marked as chains - and a set of remaining points, which remain to be clustered to get the final clustering without chains.

3.6 The complete algorithm

Let C be the cluster found by DBSCAN with metric $dist(\cdot,\cdot)$ and parameters ϵ and $minPts$. $chainDim$ and $allowedVariation$ are the parameters of chain detection. $RangeQuery(C, dist, p, \epsilon)$ returns the set $\{q \in C | dist(p,q) \leq \epsilon\}$. For the sake of simplicity assume the result of DBSCAN contains the property "Noise", which is the set of points marked as noise and the property "Clusters", which is the set of clusters. Algorithm 1 recapitulates our complete approach. For a full implementation with example code see `https://github.com/Quesstor/DBSCAN-with-density-based-connection-detection`.

4 Runtime complexity

Let n be the number of points in the cluster, on which the chain-detection algorithm is applied, in a d dimensional data space. For each point a range query with linear complexity is calculated. Calculating the covariance matrix of the ϵ-neighborhood, which in the worst case consists of all n points, is $O(n * d^2)$. Then the eigenvalues of the $d \times d$ covariance matrix is calculated, which has runtime complexity of $O(d^3)$. So the total runtime complexity for the *for* loop is $O(n(n + n * d^2 + d^3))$. The DBSCANs on a subset of the cluster each have the worst case run time complexity of $O(n^2)$. The validation step calculates for less than n points a range query resulting in a worst case run time complexity of $O(n^2)$. So the *for* loop is causing the largest performance hit with a runtime complexity of $O(n(n + n * d^2 + d^3))$. Assuming $d << n$ one can simplify the runtime complexity to $O(n^2)$.

Algorithm 1 Chain-detection

procedure VALIDATECHAINCANDIDATE($Chain, R, DB_R, dist, \epsilon$)
  $clusterFound \leftarrow$ null
  for $c \in Chain$ **do**
    for $p \in RangeQuery(R, dist, c, \epsilon)$ **do**
      if $clusterFound ==$ null **then**
        $clusterFound \leftarrow DB_R$.labelFor($p$)
      else
        if $clusterFound \neq DB_R$.labelFor(p) **then**
          return True
  return False
procedure CHAIN-DETECTION($C, dist, \epsilon, minPts, chainDim, allowedVariation$)
  $d \leftarrow dim(C)$ ▷ The dimensionality of the data
  $C_c \leftarrow \{\}$ ▷ The set of chain-points
  for $p \in C$ **do** ▷ Find all chain-point candidates
    $N \leftarrow RangeQuery(C, dist, p, \epsilon)$
    $EV \leftarrow EigenValues(CovarianceMatrix(N))$
    $EV \leftarrow EV/EV$.sum() ▷ Norm eigenvalues
    $EV \leftarrow EV$.sorted(descending=TRUE) ▷ Sort eigenvalues descending
    $e \leftarrow EV$.sum(start=$d - chaindim + 1$) ▷ Calculate error
    $e \leftarrow e * (d/(d - chainDim))$ ▷ Norm error
    if $e \leq allowedVariation$ **then** ▷ Compare error with parameter
      $C_c \leftarrow C_c \cup \{p\}$ ▷ Add p to the set of chain-points
  if $|C_c| == 0$ **then return** $\{\}$
  $R \leftarrow C \setminus C_c$ ▷ The set of remaining points
  $DB_R \leftarrow DBSCAN(R, dist, \epsilon, minPts)$ ▷ Cluster the remaining points
  $C_c \leftarrow C_c \cup DB_R$.Noise ▷ Add noise to the set of chain-points
  $DB_{C_c} \leftarrow DBSCAN(C_c, dist, \epsilon, minPts)$ ▷ Cluster chain-points
  if $|DB_{C_c}$.clusters$| == 0$ **then return** $\{\}$ ▷ No chain-candidate found
  $R \leftarrow C \setminus \cup DB_{C_c}$.Clusters ▷ Update the set of remaining points
  $DB_R \leftarrow DBSCAN(R, dist, \epsilon, minPts)$ ▷ Cluster the remaining points
  if $|DB_R$.clusters$| \leq 1$ **then return** $\{\}$ ▷ No chain-candidate can be validated
  $Chains \leftarrow []$ ▷ The list of validated chains
  for $V \in DB_{C_c}$.Clusters **do** ▷ Validate each chain-candidate
    if ValidateChaincandidate($V, R, DB_R, dist, \epsilon$) **then**
      $Chains$.append(V)
  return $Chains$

To improve performance the range queries should be executed on a tree structure and calculating the normed error for each point, which causes the largest performance hit, can easily be parallelized.

5 Experiments

The dataset on which the experiments are performed consists of all reported traffic accident locations in Great Britain from the years 2014 - 2016. It was downloaded on February the 27th 2018 from `https://www.kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data` and clustered by DBSCAN with parameters $\epsilon := 0.01$ and $minPts := 15$. These parameters were obtained by trial and error while clustering the area of roughly 100km in each direction around London's center with the goal to obtain a cluster which contains chains of traffic accidents.

Traffic accidents in London The chain-detection will be demonstrated on the cluster found at London city, see Figure 8. The results obtained by DBSCAN are not a satisfying clustering, because the highways, on which a lot of accidents happen, connect the suburban areas outside London to a single cluster. So let us apply the chain-detection algorithm. To detect these highways, which are basically one-dimensional chains, one sets the $chainDim$ parameter to 1. Since the highways are not perfectly linear and surrounded by noise, one wants to allow some error and set the $allowedVariation$ parameter to 0.2. Figure 9 shows the resulting clustering after applying the chain-detection algorithm. Most of the suburban areas are now separated from the main cluster of London city and almost all chains are found on highways.

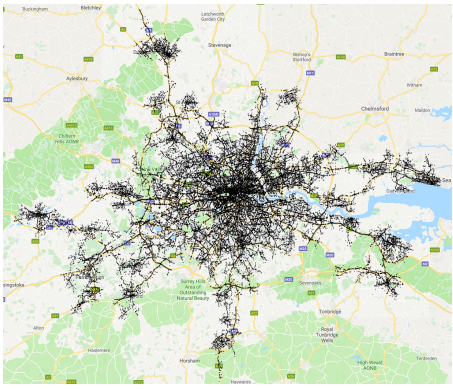


Fig. 8: The cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. The dots are stretched to fit the underlying map.

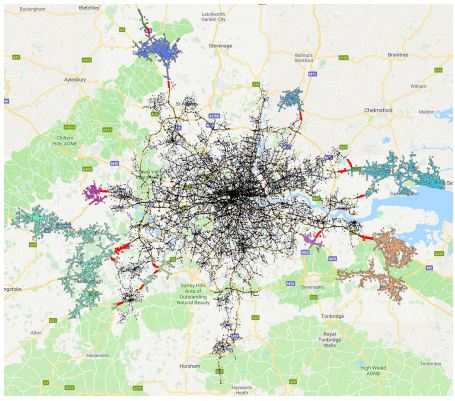


Fig. 9: Chain-detection applied on the cluster around London found by DBSCAN clustering of traffic accidents in Great Britain. Chains are marked red.

Traffic accidents in Liverpool and Manchester Another example is the cluster found at Liverpool and Manchester. As there are a lot of accidents between those cities both end up in the same cluster, see Figure 10. Let us apply the chain-detection algorithm with parameters $chainDim := 1$ and $allowedVariation := 0.2$, for the same reasons as in the previous example. In Figure 11 we can see how the traffic accident clusters are now well divided, one cluster in Liverpool and one in Manchester.

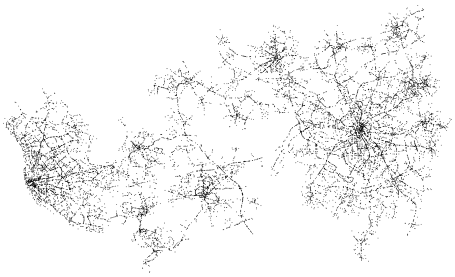


Fig. 10: The cluster of traffic accidents at Liverpool and Manchester.

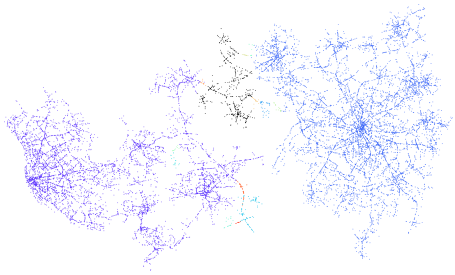


Fig. 11: Result of the chain-detection algorithm applied on the traffic accidents in Liverpool and Manchester.

6 Conclusion

In conclusion we developed the first algorithm which solves the problem that DBSCAN unintentionally detects only one cluster where several are connected by a chain or several noise points. We achieved that by recognizing chain points by analyzing the eigenvalues of the covariance matrix of their neighborhood. In our experiments we applied the algorithm on a real world dataset containing traffic accidents, where it found the intentional chains and enabled DBSCAN to find the original, smaller clusters in the dataset, instead of aggregated ones. Our approach is not limited to DBSCAN, but could also be of use after executing other clustering algorithms which tend to aggregate clusters connected by chains. Nevertheless, the ε parameter which determines in which range of each point the distribution of points is regarded, would have to be determined. We plan to examine further areas of application and experiments in future work.

References

[BK07] Birant, D.; Kut, A.: ST-DBSCAN: An algorithm for clustering spatial-temporal data. Data & Knowledge Engineering 60/1, pp. 208–221, 2007.

[Es96] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 1996, URL: https://ocs.aaai.org/Papers/KDD/1996/KDD96-037.pdf, visited on: 01/11/2019.

[He11] He, Y.; Tan, H.; Luo, W.; Mao, H.; Ma, D.; Feng, S.; Fan, J.: Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In: Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on. IEEE, pp. 473–480, 2011.

[JC16] Jolliffe, I. T.; Cadima, J.: Principal component analysis: a review and recent developments, 2016, URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4792409/, visited on: 01/11/2019.

[RSM07] Ruiz, C.; Spiliopoulou, M.; Menasalvas, E.: C-dbscan: Density-based clustering with constraints. In: International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. Springer, pp. 216–223, 2007.

Konzeption und Umsetzung einer DSL zur Informationsfusion auf verteilten heterogenen Graphen

Alexander Kern[1]

Abstract: Informationsintegration ist das Zusammenführen von Informationen aus verschiedenen Quellen. Dadurch soll eine effektivere Nutzung der Daten erreicht werden, als durch die Arbeit mit den einzelnen Quellen möglich ist. Allerdings ist Informationsintegration ein hochkomplexes Problem. Es umfasst neben der Duplikatserkennung auch das Auflösen von Inkonsistenzen auf Schema- und Instanzlevel. Diese Arbeit stellt eine domänenspezifische Sprache zur Lösung von Konflikten auf Attributwertebene für heterogene Graphdaten vor. Die Sprache stellt mit der Informationsfusion einen Teilschritt des Informationsintegrationsprozesses zur Verfügung. Neben der Gestaltung der DSL und der Entwicklung eines Prototyps mit Apache Flink und Gradoop beurteilt eine Evaluation der Fusionsergebnisse die Qualität des Verfahrens.

Keywords: Informationsintegration, Informationsfusion, Gradoop, Graphen, DSL, Apache Flink

1 Einleitung

Die Verarbeitung großer Datenmengen ist eine immer wichtigere Aufgabe in vielen verschiedenen gesellschaftlichen Bereichen. Die Arbeit mit Big Data bringt jedoch eine Vielzahl von Schwierigkeiten mit sich. Neben der großen Menge an Daten, die oft nicht mehr an einem einzelnen Rechner bearbeitet werden können, sind auch verschiedene Datenformate, fehlende oder fehlerhafte Daten und die Anforderung nach zeitnaher Datenverarbeitung Probleme, die gelöst werden müssen. Eine weitere Herausforderung ist es, eine geeignete Darstellung für die gesammelten Daten zu finden. Eine mögliche Darstellungsart für Datenobjekte mit komplexen Beziehungen sind Graphen, denn diese ermöglichen eine intuitive Abbildung und Analyse der vorhandenen Daten[JP16]. Graphen stellen Entitäten als Knoten und Beziehungen zwischen diesen Entitäten als Kanten dar.

Eine wichtige Aufgabe ist das Zusammenführen von Daten aus unterschiedlichen Quellen. Hierbei spricht man von Informationsintegration. Dadurch ist es möglich, in einzelnen Quellen fehlende Daten zu ergänzen, Fehler zu finden und beheben und durch weitergehende Analysen zusätzliche Erkenntnisse, auch über Datenquellengrenzen hinweg, zu erlangen[Li10, S. 266]. Allerdings ist Informationsintegration ein komplexes Problem, dass

[1] Universität Leipzig, Big Data Kompetenzzentrum, Ritterstraße 9-13, 04109 Leipzig, ak44xubu@studserv.uni-leipzig.de

doi:10.18420/btw2019-ws-19

unterschiedliche Teilaspekte wie Transformationen, Schemaintegration, Entity Resolution, Duplikatserkennung und Datenfusion umfasst[LN06, S. 6ff].

Diese Arbeit stellt eine Lösung für Attributwertkonflikte bei der Fusion von Daten in einem Prozess zur Informationsintegration vor. Die Lösung nutzt eine domänenspezifische Sprache zur Auswahl von Attributen aus heterogenen Graphdaten und Lösungsstrategien für Konflikte zwischen den vorhandenen Attributwerten. Mithilfe der DSL lässt sich das Schema und die Logik zur Berechnung der Ausgabeknoten festlegen.

2 Verwandte Arbeiten

Im Paper „Declarativ Data Cleaning: Language, Model, and Algorithms“[GFS01] stellen Galhardas et. al. eine an die SQL-Syntax angelehnte deklarative Sprache vor, die durch mehrere Schritte unsaubere Daten aufbereitet. Unsauberkeiten können Fehler, Inkonsistenzen und inkompatible Schemaunterschiede sein. Das Ziel ist dabei eine klare Trennung der logischen Beschreibung mithilfe der Querysprache und der physischen Ausführung durch das Programm. Dabei werden die Queries, welche die einzelnen Schritte beschreiben, zu Java-Programmen umgewandelt. Nach dem Laden und Transformieren der Daten findet Entity Resolution, das Auffinden von verschiedenen Datensätzen, die eine Entität beschreiben, statt. Der letzte Schritt ist das Merging, in dem benutzerdefinierte Aggregationsfunktionen die vorhandenen Attributwerte zu einem einzelnen Wert zusammenführen. Der im Paper vorgestellte Prozess legt den Schwerpunkt auf das Aufbereiten von unsauberen Daten. Eventuelle Heterogenität im Datenschema beseitigt der Prozess beim Laden, weshalb das Merging auf homogene Daten beschränkt ist.

Einen ähnlichen Prozess zur Informationsintegration von heterogenen Graphdaten stellen Lim et. al. in [Li10] vor. Der erste Schritt in ihrem Prozess ist die Schemaintegration. Dabei werden Daten aus den verschiedenen Quellen zuerst in ein einheitliches Schema gebracht. Anschließend werden beim Instance Matching Entitäten und Relationen einander zugeordnet. Am Ende findet die Auflösung von Attribute-Value-Konflikten statt. Die Veröffentlichung betrachtet nur den Schritt des Instance Matching im Detail.

In „Declarative Data Cleaning With Conflict Resolution“[NH02] konzentrieren sich Felix Naumann und Matthias Häussler auf die Lösung von Attribute-Value-Konflikten. Sie verfolgen einen Ansatz um Datenintegration mithilfe von SQL durchzuführen. Im Paper nennen die Autoren eine Vielzahl von möglichen Funktionen zur Auflösung von Attributwertkonflikten.

Gradoop (Graph Analytics on Hadoop) ist ein Framework für verteilte deklarative Graphanalysen. Es verbindet die Stärken von Graphdatenbanksystemen wie Neo4j und verteilten Graph Processing Systeme wie Google Pregel oder Gelly, in dem es die verteilte Ausführung von Graphabfragen und -algorithmen für EPGM-Graphen bietet. Das Extended Property Graph Model beschreibt Graphen mithilfe von Knoten, Kanten, Graphen und Graph Collections. Jedes Objekt im EPGM-Graph kann dabei durch **Properties** mit Eigenschaften als

Key-Value-Paaren versehen werden.[Ju18] FAMER ist ein Framework für verteilte Entity Resolution auf Daten aus verschiedenen Quellen. Zur Ausführung setzt FAMER Apache Flink und Gradoop ein[SPR17].

3 Konzeption

Das folgende Kapitel erläutert den geplanten Informationsintegrationsprozess und die daraus resultierenden Anforderungen an die domänenspezifische Sprache zur Informationsfusion.

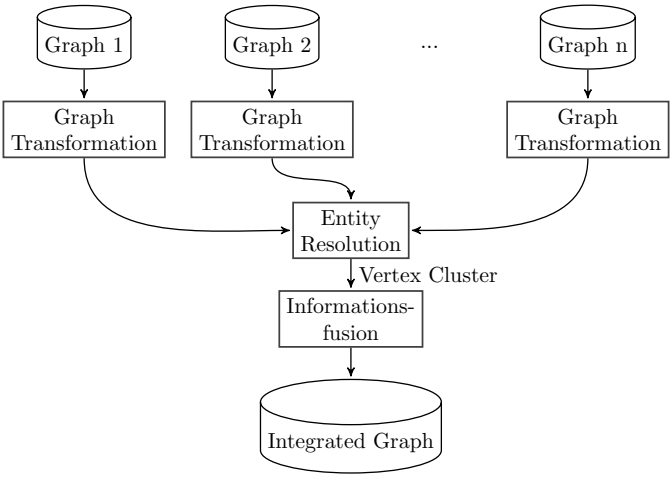


Abb. 1: Informationsintegrationsprozess mit Gradoop

Abb. 1 zeigt den Ablauf der Informationsintegration mithilfe von Gradoop. Mithilfe von Transformationen kann eine Vorverarbeitung der Daten stattfinden. Eine Angleichung der Datenschemata ist hierbei allerdings nicht zwingend nötig. Die Unterstützung heterogener Daten ist ein essentieller Unterschied zu den anderen vorgestellten Arbeiten. Anschließend findet Entity Resolution mithilfe von FAMER statt. Die dabei gematchten Datensätze beschreiben jeweils die gleiche Entität. Jeder dieser Matching-Cluster wird anschließend mithilfe der von der DSL beschriebenen Query zu einem einzelnen Knoten fusioniert. Der Ergebnisgraph umfasst alle diese generierten Knoten. Eine Fusion eventuell vorhandener Kanten findet derzeit nicht statt.

3.1 Anforderungen an die DSL

Die Hauptaufgaben der DSL sind das Auswählen der Attribute aus den heterogenen Eingangsdaten, die Definition des Schemas und die Lösung von Attributwertkonflikten in den Ausgabeknoten. Die DSL beschränkt sich derzeit auf die Informationsfusion der Knoten der Graphen, die Fusion von Relationen ist nicht Teil der Problemstellung.

Auf der obersten Ebene muss die DSL im Stande sein, einen oder mehrere Ausgabeknotentypen zu definieren. Für jeden Ausgabeknoten müssen Transformationsregeln festlegen, welche Attribute der Eingabedaten mit welcher Konfliktlösungsstrategie (vgl. Abschnitt 3.2) betrachtet werden sollen. Falls die gewählte Strategie zusätzliche Optionen benötigt, muss die Sprache diese bei der Regeldefinition akzeptieren können. Die Menge der Transformationsregeln beschreibt gleichzeitig das Schema des Ausgabeknotens.

Außerdem soll es möglich sein, weitere vorhandene Attribute ohne explizite Transformationsregeln zu übernehmen. Dies soll nicht das Standardverhalten sein, jedoch zur möglichen Verkürzung von Queries einsetzbar sein. Die Idee hierbei ist, ohne großen Aufwand einen ersten Überblick über die Daten zu gewinnen, den der Nutzer anschließend mit spezifischen Transformationsregeln verfeinern kann.

Zur Vereinfachung von Queries soll es außerdem eine Kurzform für die Listen von Eingabeattributen geben, wenn diese in allen Quellen den gleichen Namen haben. Diese Voraussetzung lässt sich durch Graphtransformationen leicht herstellen.

Die Endanwender der DSL sind einerseits Nutzer von Gradoop, bei denen von vorhandenen Java- und SQL-Kenntnissen ausgegangen werden kann. Andererseits soll die DSL auch von Domänenexperten, beispielsweise aus dem Business-Intelligence-Bereich, nutzbar sein. Bei diesen kann zwar nicht von Programmierkenntnissen, allerdings auch von SQL-Kenntnissen ausgegangen werden.

Ein wichtiger Aspekt für die Gestaltung einer domänenspezifische Sprache sind die Kosten für Programmerstellung, -verifikation und -wartung[Ba17]. Um diese gering zu halten, ist es nötig, die Problemdomäne so abzubilden, dass die Anwendung der Sprache dem Nutzer möglichst klar erscheint. Aber auch Punkte wie die Komplexität der Syntax sind hierbei von Bedeutung. Unter Beachtung der Endnutzer der Sprache sollen diese Faktoren beim Design beachtet werden.

3.2 Attribute Value Conflict Resolution

Unter Betrachtung der von Naumann und Häussler vorgestellten Strategien zur Lösung von Attributwertkonflikten [NH02] wurden die Tab. 1 gezeigten Konfliktlösungsstrategien entwickelt.

Diese Konflikte können aus unterschiedlichen Datenschemata (zum Beispiel bei Datumsangaben), verschiedenen Semantiken in den unterschiedlichen Quellen (zum Beispiel bezeichnet `name` einmal den gesamten Namen einer Person, einmal nur den Nachnamen), inhaltlichen Fehlern (zum Beispiel Tippfehler oder veraltete Daten) oder unterschiedlicher Schemata der Quelldaten stammen.

Einfache Strategien sind beispielsweise das Wählen des ersten Werts in der alphanumerischen Ordnung und das Konkatenieren aller vorhandenen Werte durch ein Separatorsymbol. Die

Strategie	Beschreibung	Optionen
Straight	Erster Nicht-Null-Wert nach alphanumerischer Ordnung	-
Retain	Konkatenation der vorhandenen Werte	Separatorsymbol
Priority	Erster Nicht-Null-Wert in der spezifizierten Ordnung der Quellen	Sortierung der Quellen
Newest	Wert mit dem aktuellsten Zeitstempel	Zeitstempel-Attribute
Majority	Wert, der in den meisten Quellen vorhanden ist	-
Source	Wert mit dem höchsten summierten Gewicht	Gewichte für Quellen
Property	Auswahl anhand der vorhandenen Attributwerte mit Substrategie	Substrategie

Tab. 1: Strategien zur Konfliktlösung

priority-Strategie nutzt eine Ordnung für die Quellen, mit der ein Anwender Präferenzen ausdrücken kann. Verfügen die Datensätze über Zeitstempel ist es möglich, den neuesten bzw. zuletzt geänderten Wert auszuwählen.

Die **majority**-Strategie ermöglicht die Auswahl des am häufigsten vorhandenen Werts. Eine Verfeinerung dazu ist die **source**-Strategie, die den Quellen zusätzlich Gewichte gibt, um so detailliertere Präferenzen auszudrücken. Bei Gleichstand soll wie bei der **straight**-Strategie der erste Wert in der alphanumerischen Ordnung gewählt werden.

Strategien basierend auf den vorhandenen Attributwerten sind in der **property**-Strategie gebündelt. Diese lassen sich in zwei Klassen aufteilen: Strategien für Strings und Strategien für Zahlenwerte. Für Strings steht die Auswahl des kürzesten, längsten oder des längsten gemeinsamen Teilstrings (LCS) zur Verfügung. Aus Zahlwerten können Minimum, Maximum, Mittelwert oder der Median gebildet werden.

4 Design der DSL

Basierend auf den Anforderungen aus dem vorigen Abschnitt entstand eine prototypische Implementierung einer DSL und der dazugehörigen Ausführungsengine. Zur Umsetzung der Sprache dient das Tool ANTLR[2], das aus Grammatiken in EBNF[3]-ähnlicher Form Lexer und Parser erzeugen kann. Das Programm parst die Queries zu Java-Objekten, mithilfe derer die Ausführungsengine durch den Einsatz von Apache Flink und Gradoop die Informationsfusion für die vorhandenen Daten ausführen kann.

Barišić stellt in ihrer Doktorarbeit[Ba17] einen iterativen Prozess zum Design von DSLs hinsichtlich der im vorigen Kapitel genannten Qualitätsmerkmale vor. Er beginnt mit der Abbildung der Problemdomäne, geht weiter über das konkrete Gestalten und Implementieren der Sprache hin zur Evaluation durch Endnutzerfeedback oder Metriken wie Effektivität

[2] https://antlr.org (07.08.2018)
[3] https://www.ics.uci.edu/ pattis/ICS-33/lectures/ebnf.pdf (07.11.2018)

im Vergleich zum Arbeitsprozess ohne DSL. Sind die Ergebnisse der Evaluation nicht zufriedenstellend, muss der Sprachentwickler die vorigen Schritte nach Bedarf wiederholen.

Insgesamt wurden vier funktional identische Sprachprototypen entwickelt und hinsichtlich ihrer Nutzerfreundlichkeit evaluiert. Der erste Sprachansatz nutzt die Ergebnisse von Galhardas et. al.[GFS01] und erweitert ihre Sprache um Unterstützung für heterogene Schemata und die vorgestellten Strategien und mögliche Optionen. List. 1 zeigt das Beispiel aus dem Paper in der DSL.

```
CREATE MERGING
LET Author
  name = property(
    (Source1.author.name, Source2.author.full_name),
    textual:longest),
  authorKey = straight(key)
```

List. 1: SQL-basierte DSL

Das Beispiel zeigt die wichtigsten Eigenschaften der Sprache. Sie generiert einen Ausgabeknoten mit dem Label `Author`, der zwei Attribute `name` und `authorKey` erhält. Die Berechnung der Attributwerte für die erzeugten Knoten erfolgt durch zwei verschiedene Strategien. Die erste Regel zeigt die Auswahl von Attributen aus heterogenen Quellen anhand des Quellennamens, Knotentyps und Attributnamens, sowie die zusätzliche Angabe einer Option für die **property**-Strategie. Die zweite Regel zur Wahl des `authorKey` ähnelt in der verkürzten Form wiederum stark der von Galhardas vorgeschlagenen Sprache. Ausführliche Queries umfassen weitere Ausgabeknoten mit mehr Transformationsregeln.

```
node Author {
  property name {
    Source1.author.name
    Source2.author.full_name
  } strategy property { textual:longest }

  property authorKey { key } strategy straight
}
```

List. 2: Kotlin-inspirierte DSL

Ein anderer Designansatz orientiert sich am Design von DSLs, die mithilfe der Programmiersprache Kotlin erstellt werden können. Diese DSL nutzt Schlüsselwörter und Blöcke um Beschreibungen näher an natürlicher Sprache zu erreichen. List. 2 zeigt die Beispielquery in der Sprache.

Die dritte DSL ist ähnlich zur SQL-basierten DSL aufgebaut und unterscheidet sich vor allem in kleinen Details wie der Definition von Attributlisten. Der vierte Ansatz nutzt statt

einer mit ANTLR generierten Sprache YAML. Die Beschreibung der Informationsfusion findet mithilfe einer YAML-Datei statt, deren Schema wohldefiniert ist.

Die letzte vorgestellte Sprache ist sowohl was die textuelle Länge der Queries als auch den Komfort der Queryerstellung angeht den anderen Sprachen unterlegen. Das manuelle Schreiben von YAML ist zwar einfacher als in anderen Auszeichnungssprachen, trotzdem deutlich anspruchsvoller als die Nutzung der anderen DSLs. Die drei weiteren Sprachen sind sich sehr ähnlich. Die Vorteile der weiten Verbreitung von SQL nennen bereits [GFS01] und [NH02]. Unter Berücksichtigung der Annahme, dass die Endnutzer bereits über SQL-Erfahrungen verfügen, fällt die Entscheidung für diese DSL. Die geringe Einstiegshürde durch Vorerfahrung und die relativ bekannte Syntax ermöglichen es, Queries in der DSL sowohl schnell erstellen als auch verstehen zu können.

Durch die Trennung der Sprache von der Ausführungsengine ist es allerdings jederzeit möglich, die Sprache zu ändern, auszutauschen oder mehrere DSLs parallel zu nutzen. Genau so kann das Backend ersetzt werden, um andere Datenquellen neben Graphdaten in Gradoop zu unterstützen, um die Sprache beispielsweise für Konfliktlösung bei relationalen Daten zu nutzen.

5 Evaluation

Um die Laufzeit und Qualität der Informationsfusion zu messen, wird diese auf den MusicBrainz-Datensatz angewandt. Er besteht aus echten MusicBrainz-Daten und mithilfe des DAPO-Datengenerators[Hi18] erzeugten Duplikaten. Der Datensatz umfasst 10780 Knoten in 4611 Clustern. Daraus entsteht ein Graph mit 15391 Knoten und 19767 Ähnlichkeitskanten. Zur Messung der Matching-Qualität kommt FAMER zum Einsatz. Die Berechnungen finden auf dem Galaxy-Cluster[4] mit Gradoop 0.4.0 und Apache Flink 1.5.0 statt.

5.1 Matching-Qualiät

Die Qualität des Matchings soll mit den Maßen Recall, Precision und F-Measure bestimmt werden. Dafür werden aus den bereits geclusterten Ausgangsdaten mithilfe der Informationsfusion die Ergebnisknoten erzeugt und zu dem Graph der Ausgangsdaten hinzugefügt. Daraus lässt sich leicht der Referenzgraph mit den gegebenen Ähnlichkeitskanten bestimmen. Anschließend werden die Informationen über die bisherige Clusterzugehörigkeit entfernt und ein neues Linking mithilfe von FAMER berechnet. Die dabei entstehenden Ähnlichkeitskanten können mit dem Referenzgraph verglichen werden. Clustering wird nicht eingesetzt. Als grobe Vergleichswerte dienen hierbei die Matching-Ergebnisse aus dem Evaluationspaper von FAMER [SPR17]. Im Paper erreicht das Matching nach Linking und

[4] https://www.urz.uni-leipzig.de/fue/sc/galaxy/ (06.12.2018)

Clustering für den MusicBrainz-Datensatz Werte im Bereich [0.5, 0.7] für das F-Measure. Die Linking-Konfiguration entspricht der aus dem Paper, d.h. Blocking nach `album` und 3-Gramme als Ähnlichkeitsfunktion. Clustering kommt nicht zum Einsatz, alle weiteren Einstellungen entsprechen gegebenen Standardwerten.

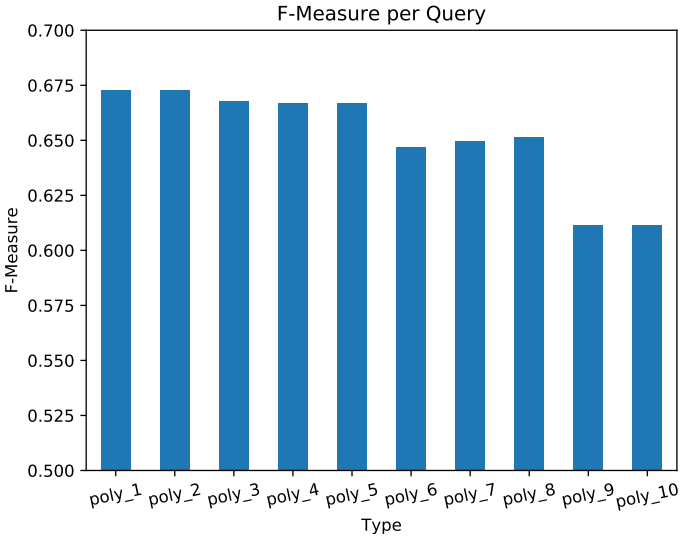


Abb. 2: Ergebnisse der Poly-Strategie-Queries

Abb. 2 zeigt die Ergebnisse von Queries, die verschiedene Strategiekombinationen einsetzen. Die meist relativ ähnlichen Ergebnisse für die einzelnen Strategien zeigen, dass das Verfahren flexibel einsetzbar ist. Da die Auswahl der Strategien letztlich abhängig von den gegebenen Daten und dem Kontext und Ziel der Informationsfusion abhängt, ist es vorteilhaft, dass die Strategien frei kombinierbar sind. Durch bessere Konfigurationen von FAMER lassen sich die Ergebnisse deutlich verbessern.

5.2 Laufzeitmessung

Zur Berechnung der Laufzeit werden die im vorigen Abschnitt vorgestellten Queries jeweils 100 mal auf dem Cluster ausgeführt. Apache Flink führt Transformationen erst aus, wenn eine Senke definiert ist. Deshalb findet die Messung innerhalb des Java-Programms um die gesamte Job-Ausführung inklusive IO-Operationen statt. Abb. 3 zeigt die gemessenen Laufzeit für eine beispielhafte Query mit verschiedenen eingesetzten Strategien.

Es zeigen sich jeweils zwei Gruppen von Laufzeiten (bei ca. 10750ms und 12750ms Laufzeit) um die sich die weiteren gemessenen Werte in einem Bereich von etwa 750ms verteilen, was einer Abweichung von etwa 6.9% bzw. 5.8% entspricht.

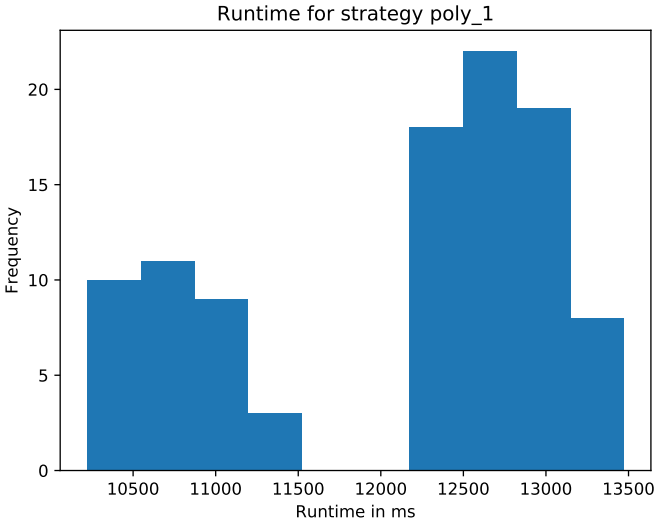


Abb. 3: Laufzeitmessung

Das Verfahren skaliert durch die Verteilung der einzelnen Cluster. Die maximale Clustergröße entspricht theoretisch der Anzahl an Quellen und ist damit eher gering, die Anzahl der Cluster ist idealerweise die Anzahl an Entitäten und dementsprechend groß. Da die parallele Berechnung der einzelnen Cluster unabhängig verteilbar ist, sollte das Verfahren gut skalieren.

6 Zusammenfassung und Ausblick

Im Rahmen dieses Beitrags wurde eine domänenspezifische Sprache zur Informationsfusion von heterogenen Graphdaten vorgestellt. Die Sprache stellt einen Teil der Informationsintegration mit Gradoop zur Verfügung. Nutzer können mit ihr definieren, wie Attributwertkonflikte zwischen vorhandenen Werten gelöst werden sollen. Mithilfe von Apache Flink und Gradoop findet eine verteilte Ausführung der Informationsfusion statt. Laufzeit und Qualität des Verfahrens wurden außerdem in Abschnitt 5 evaluiert.

Weitergehende Aufgaben sind eine ausführliche Evaluation mit Endnutzern der Sprache und die Erweiterung der DSL auf Graphkanten. Ein weiterer interessanter Punkt ist die Auswahl von Standardstrategien anhand der gegebenen Datenschemata. Auch die Nutzbarkeit für andere Anwendungsfälle sowie mögliche Implementierungen geeigneter zusätzlicher Strategien können in Zukunft untersucht werden.

Acknowledgements:

Die vorliegende Arbeit wurde teilweise gefördert durch das Bundesministerium für Bildung und Forschung innerhalb des Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B). Betreut wurde die Arbeit von Matthias Kricke[5] und Eric Peukert[6]. Berechnungen für diese Arbeit wurden mit Ressourcen des Rechenzentrums der Universität Leipzig durchgeführt.

Literatur

[Ba17] Barisic, A.: Usability Evaluation of Domain-Specific Languages, Diss., Universidade Nova De Lisboa, Dez. 2017.

[GFS01] Galhardas, H.; Florescu, D.; Shasha, D.: Declarative Data Cleaning: Language, Model, and Algorithms. In: In VLDB. S. 371–380, 2001.

[Hi18] Hildebrandt, K.; Panse, F.; Wilcke, N.; Ritter, N.: Large-Scale Data Pollution with Apache Spark. IEEE Transactions on Big Data/, 2018, ISSN: 2332-7790.

[JP16] Junghanns, M.; Petermann, A.: Verteilte Graphanalysen mit Gradoop. JavaSPEKTRUM 5/, Abgerufen am 26.07.2018, 2016, URL: https://www.sigs-datacom.de/uploads/tx_dmjournals/junghans_petermann_JS_05_16_eeNZ.pdf.

[Ju18] Junghanns, M.; Kießling, M.; Teichmann, N.; Gómez, K.; Petermann, A.; Rahm, E.: Declarative And Distributed Graph Analytics With GRADOOP. In: PVLDB. Bd. 11. 12, 2018.

[Li10] Lim, E. P.; Sun, A.; Datta, A.; Kuiyu, C.: Information Integration for Graph Databases. In: Link Mining: Models, Algorithms, and Applications. Research Collection School Of Information Systems, Kap. 10, S. 265–281, 2010.

[LN06] Leser, U.; Naumann, F.: Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt.verlag GmbH, 2006, ISBN: 978-3898644006.

[NH02] Naumann, F.; Häussler, M.: Declarative Data Merging With Conflict Resolution. In: International Conference on Information Quality (IQ 2002). 2002. S. 212–224, 2002.

[SPR17] Saeedi, A.; Peukert, E.; Rahm, E.: Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution. In: Advances in Databases and Information Systems - 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings. S. 278–293, 2017, URL: https://doi.org/10.1007/978-3-319-66917-5%5C_19.

[5] kricke@informatik.uni-leipzig.de
[6] peukert@informatik.uni-leipzig.de

Computation Offloading in JVM-based Dataflow Engines

Haralampos Gavriilidis[1]

Abstract: State-of-the-art dataflow engines, such as Apache Spark and Apache Flink scale out on large clusters for a variety of data-processing tasks, including machine learning and data mining algorithms. However, being based on the JVM, they are unable to apply optimizations supported by modern CPUs. On the contrary, specialized data processing frameworks scale up by exploiting modern CPU characteristics. The goal of this thesis is to find the sweet spot between scale-out and scale-up systems by offloading computation from dataflow engines to specialized systems. We propose two computation offloading methods, reason about their applicability, and implement a prototype based on Apache Spark. Our evaluation shows that for compute-intensive tasks, computation offloading leads to performance improvements of up to a factor of 2.5x. For certain UDF scenarios, computation offloading performs worse by up to a factor of 3x: our microbenchmarks show that 80% of the time is spent on serialization operations. By employing data exchange without serialization, computation offloading achieves performance improvements by up to 10x.

Keywords: dataflow engines; computation offloading; data exchange; native execution

1 Introduction

Dataflow engines are a popular tool for large-scale data processing, due to their user-friendly programming interfaces and their ability to scale-out on clusters. Through their numerous application programming interfaces (APIs), dataflow engines provide domain-specific abstractions for a variety of data processing tasks, such as relational processing, graph processing, and machine learning. Dataflow programs consist of chained data processing operators, while custom functionality is added by user-defined functions (UDFs). State-of-the-art dataflow engines, such as Apache Spark [Za12] and Apache Flink [Al14] provide their standard APIs in JVM-based programming languages, e.g. Scala and Java, since they are built on the JVM. JVM-based languages are a convenient choice for users, because of their higher-level abstractions, but at the same time are incapable of applying lower-level optimizations supported by modern CPUs [Cr15; Es18]. Additionally, in the case of big data applications, the garbage collection of a large number of objects causes execution stalls [Ng16]. On the contrary, specialized processing frameworks, such as database systems [Ne11] and numerical libraries are built using lower-level programming languages, and therefore provide better support for CPU-specific optimizations. To enhance dataflow

[1] Technische Universität Berlin, Database Systems and Information Management Group, Einsteinufer 17, 10587 Berlin harry_g@mailbox.tu-berlin.de

doi:10.18420/btw2019-ws-20

engines with native performance, we propose to offload computations from the JVM to native runtimes. We describe the applicability of computation offloading in current systems, and discuss the challenges of such an integration. More specifically, we make the following contributions:

- We propose two methods for computation offloading in state-of-the-art dataflow engines and describe a prototype implementation on Apache Spark.
- We evaluate our approaches with microbenchmarks and end-to-end benchmarks using several types of UDFs.

2 Background

In this section we give a brief overview of dataflow engines and the JVM memory model.

2.1 Dataflow Engines

In the following we describe the main characteristics of state-of-the-art dataflow engines, such as Apache Spark [Za12] and Apache Flink [Al14].

Programming Model. Dataflow engines employ distributed collection processing, introduced by the MapReduce paradigm. Users implement dataflow programs by chaining second-order functions, such as `map` and `reduce`. Custom functionality is added by UDFs. Higher-level APIs offer domain-specific operators, e.g. for relational processing, graph processing and machine learning. Each operator describes a transformation to the dataset. The composed operator tree forms a directed graph, used for optimization and execution.

Architecture and Execution Model. Dataflow engines use the master/worker model in a shared-nothing environment. Each worker node runs a JVM instance, which is used to execute processing tasks. Workers receive processing instructions and execute them independently. Intermediate computation results are exchanged between workers through their network interface. During execution, workers store and process data in memory, and move it to the disk only when it exceeds the memory size. The discussed dataflow engines implement the iterator model, where every physical operator is executed element-wise for all data records. For failed computations, dataflow engines use recovery mechanisms, such as lineage graphs and snapshots.

2.2 JVM Heap and Off-Heap

The JVM memory is organized around objects. New objects are created on the Heap, which is periodically scanned by the garbage collector. The garbage collector organizes objects based on their age and cleans unreferenced objects. Next to the Heap, JVM applications have access to the Off-Heap. Binary data residing on the Off-Heap is accessed as in lower-level

languages, using memory pointers. Since the Off-Heap is not managed by the JVM, it is not subject to garbage collection. However, it is necessary to implement manual memory management within applications to avoid JVM crashes.

3 Computation Offloading in Dataflow Engines

In this section, we describe our computation offloading approach by the example of UDF processing. First, we analyze the current UDF processing method in dataflow engines and discuss potential bottlenecks. Second, we discuss two mechanisms for data exchange between a worker JVM process and a native process. Finally, we describe the implementation of our computation offloading prototype in Apache Spark.

3.1 Current UDF Processing

UDFs are important building blocks of dataflow programs. Depending on the semantics of the surrounding operator, UDFs are either applied on single elements or on groups of elements. In JVM-based dataflow engine APIs, UDFs are implemented in Java or Scala. During distributed execution, workers deserialize binary data received from their network interface before processing the UDF, as illustrated in Fig. 1. Incoming data is deserialized from the Off-Heap memory to the Heap, before processing the UDF. Subsequently, the UDF result is serialized from the Heap to the Off-Heap memory, and sent over the network interface. We observe three performance bottlenecks in the current UDF processing method.

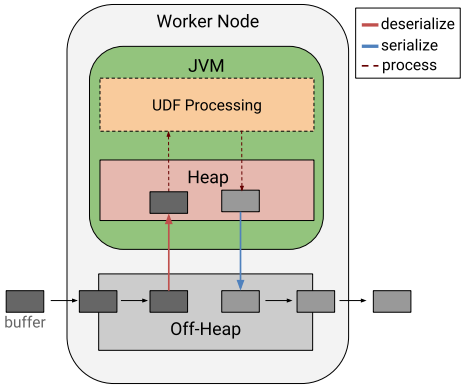


Fig. 1: Current UDF processing method.

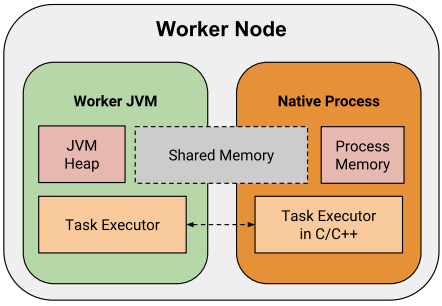


Fig. 2: UDF computation offloading.

First, instead of directly processing the data, workers must deserialize it and transform it into objects[2]. Second, because of the dataflow engine execution model, data is serialized

[2] Memory management in Flink: https://flink.apache.org/news/2015/05/11/Juggling-with-Bits-and-Bytes.html

row-wise. This means that for every row on the Off-Heap, the respective object will be created on the Heap. Garbage collection of a large number of objects leads to execution stalls [Ng16]. Third, the JVM runtime hinders lower-level optimizations, e.g. SIMD instructions, supported by modern CPUs [Cr15; Es18].

3.2 UDF Offloading Architecture

In the following, we describe an architecture for moving UDF computations from the JVM to native environments. Executing the UDF in a native environment avoids the deserialization operations, the garbage collection overhead, and allows for optimized processing using CPU-specific instructions. The architecture of our approach is depicted in Fig. 2. Using a shared memory area, the worker is able to exchange data between the the JVM and a native process. The UDF is offloaded from the JVM task executor to a native task executor. We propose two methods to exchange data between the JVM and the native runtime.

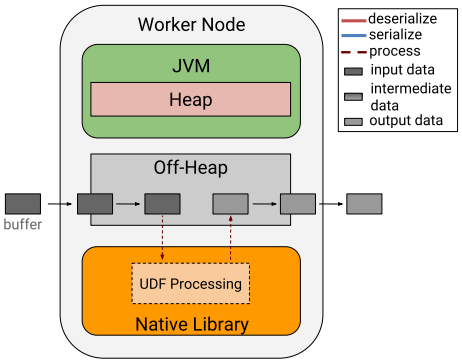


Fig. 3: Off-Heap data exchange.

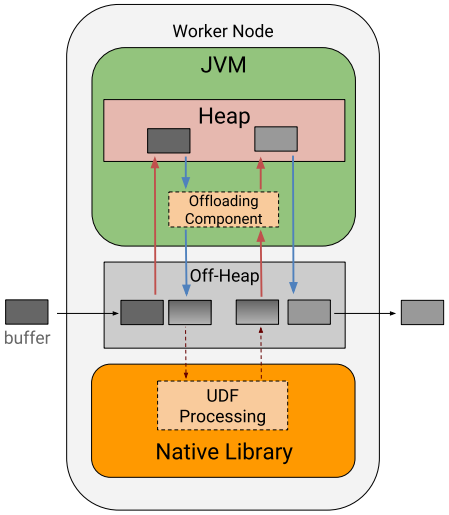


Fig. 4: On-Heap data exchange.

Off-Heap Offloading. During the *off-heap* data exchange method illustrated in Fig. 3, the Off-Heap memory is used as a shared memory region between the worker JVM and the native process. Since languages such as C and C++ process binary data, it is not necessary to transform the data before UDF execution in the native process. By exchanging a memory pointer, the native process has access to the data on the Off-Heap. A disadvantage of this method, is that native UDF execution is carried out directly on network buffers. This means that if the data is serialized row-wise in the network buffers, no optimizations for columnar processing can be applied. Moreover, network buffers contain metadata, such as event logs, next to the data records, which hinders efficient memory access during UDF processing.

On-Heap Offloading. To address the limitations of the *off-heap* method regarding data formats, we introduce the *on-heap* method. As illustrated in Fig. 4, after receiving and deserializing the data to the JVM Heap, the objects are serialized to the JVM Off-Heap, through the offloading component. The result of the native UDF is written to the JVM Off-Heap. Subsequently, the offloading component deserializes the result binary data to the Heap, which is then serialized again by the worker to send it over the network interface. An advantage of the *on-heap* approach is that its implementation is possible without altering the dataflow engine core. Moreover, since a custom serializer is provided in the offloading component, it is possible to choose the format for UDF processing, instead of being bound to the format in the network buffer. A disadvantage of the discussed approach is the overhead of the mediating computation offloading component. On the one hand, it allows to choose between data formats for improving the UDF performance. On the other hand, the intermediate serialization and deserialization operations lead to significant performance overhead, as we discuss in the next section.

The main advantage of the *off-heap* method is that serialization operations are completely avoided. However, it is not possible to implement it within current systems, without changing system components. On the contrary, the *on-heap* method is applicable without any modifications. The main disadvantage of the *on-heap* method, is the overhead of the intermediate serialization operations, which allow to change the data format before processing. Converting the data to a columnar format, is beneficial for certain operations, e.g. aggregations that reference few columns of a dataset. Additionally, aggregation UDFs have a small result size, a property that minimizes the *on-heap*'s intermediate serialization step overhead. Furthermore, using the *on-heap* method, efficient memory access is guaranteed in the native UDF, since the offloading component serializes only necessary record data.

3.3 Prototype on Spark

We implement our *on-heap* offloading prototype on top of Apache Spark. We use the interface of the `mapPartitions` operator to implement the computation offloading component, which serializes and deserializes all intermediate input and output data. For the serialization operations we use the `ByteBuffer` library[3], which allows to create `ByteBuffer` objects on the Off-Heap (`DirectBuffer` objects). We use custom row and column serialization for data exchange. For the row-oriented serialization, we write all input element attributes consequently to one `DirectBuffer`, while for the column-oriented serialization we use as many `DirectBuffer` objects as input columns. For our prototype, we implement UDFs in C, and compile them to shared libraries using gcc. We use the Java Native Interface[4] (JNI), a framework for embedding native applications within the JVM runtime, to register the native functions in our prototype. Data exchange is established by serializing data to `DirectBuffer` objects and passing pointers and the input data size as the native function parameters.

[3] https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html
[4] https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html

Our prototype uses pre-compiled compute kernels implemented in C. To build optimized kernels for the native task executor, lower-level libraries that make use of specialized hardware and SIMD instructions can be used, e.g. Intel MKL[5], BLAS[6]. For arbitrary UDF support, it is necessary to extend the prototype with a compiler component, which translates UDFs to lower-level representations. Furthermore, to implement the *off-heap* approach in current systems, UDF interfaces which provide access to serialized buffers for the native task executor are necessary[7], for both row and columnar formats.

4 Evaluation

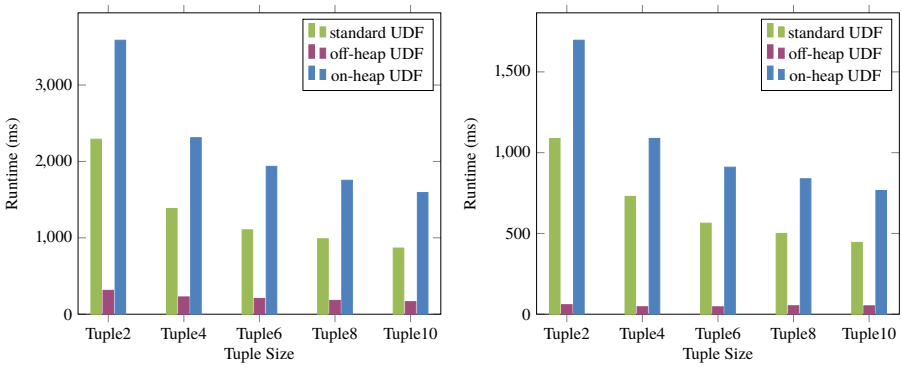
In this section, we evaluate the performance of our proposed computation offloading approach. We first present a set of Java microbenchmarks that measure the performance of standard UDF processing compared to the *on-heap* and *off-heap* offloading methods. Furthermore, we evaluate our prototype on Apache Spark in several UDF scenarios.

4.1 Microbenchmarks

We conduct a set of Java microbenchmarks to get insights about the performance of the standard approach and our introduced offloading methods.
Setup. We execute our benchmarks on a machine with an Intel(R) Xeon(R) E5530 CPU (16 cores) with a clock rate of 2.40GHz and 20 GB of main memory, running Ubuntu 14.04. For the benchmarks we use Java version 1.8 and Scala version 2.11 with the Hotspot VM 25.181-b13. We initialize the JVM with 10 GB memory. We use GCC to compile C code to native libraries and provide the compilation flags `-O3 -march=native -mtune=native`, which enable vectorization among other optimizations. We perform the microbenchmarks using JMH[8], a benchmarking library for Java. We fork each experiment three times, perform ten warmup iterations and then execute the experiment five times. We report the average execution time of five repetitions.
Offloading Approaches. To evaluate our offloading approaches, we implement UDFs of two types, a euclidean distance function that is applied to two fields of every row, and an aggregation function that sums all elements of all rows. For a fixed data size of 500MB containing integers, we scale the number of rows and columns. For example, the 2 column dataset has 62,500,000 rows, while the 10 column dataset has 12,500,000 rows.

We present the results of the two UDF microbenchmarks in Fig. 5. The standard UDF processing method involves deserialization, processing and serialization on the JVM. The *off-heap* UDF method involves only processing in C, while the *on-heap* method involves the

[5] https://software.intel.com/en-us/mkl
[6] http://www.netlib.org/blas/
[7] Current Spark and Flink both support relational operations on serialized data, but not for UDFs
[8] http://openjdk.java.net/projects/code-tools/jmh/



(a) Euclidean distance UDF.

(b) Sum UDF.

Fig. 5: Standard UDF processing compared to *Off-heap* and *On-Heap* computation offloading.

standard serialization and deserializations on the JVM, processing in C and an additional serialization step for the intermediate results, as described in Sect. 3. For the distance UDF, we observe that the *off-heap* method outperforms the standard approach by up to a factor of 10x, for a dataset of two columns. The performance gap drops while increasing the column size, since a larger column size means less created JVM objects, and hence less serialization overhead. The *on-heap* method performs almost 1.5x worse than the standard method, since it involves additional serialization operations for the intermediate result. We observe the same results for the sum UDF. The performance gap between the *off-heap* and the standard method is larger, a result possibly explained by faster aggregation processing in C.

Standard UDF Analysis. In order to explain the performance of the standard UDF processing method, we microbenchmark all three involved operations in the distance function: deserialization of the input, processing, and serialization of the output. The results are shown in Fig. 6. We observe that for a dataset size of 500MB the performance for processing a two-column dataset is 2x worse than the performance of processing a ten-column dataset. The detailed results show that when the column number is increased, the serialization and deserialization overhead is minimized, and the processing performance is increased. When the number of columns is increased, the number of rows is decreased, hence less objects are created and less serialization and deserialization overhead is created. Nevertheless, for ten-column datasets, the execution time of serialization and deserialization operations attribute to 80% of the overall UDF execution time.

4.2 Prototype Evaluation

We implement two UDFs using our *on-heap* offloading approach in Apache Spark, as discussed in Sect. 3. We use two synthetic datasets that contain elements of type double. For a fixed size of 5GB, the first dataset has 2 columns, while the second dataset has 10

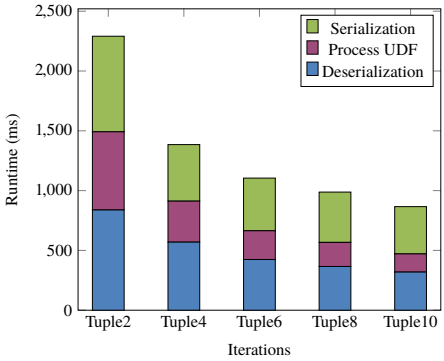


Fig. 6: Detailed standard UDF performance.

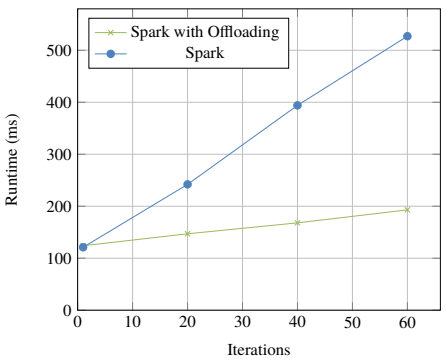


Fig. 7: Compute-heavy UDF in Spark

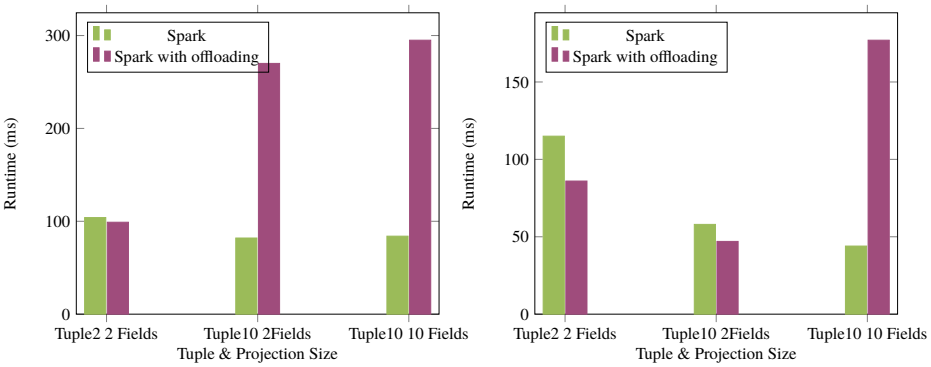
columns. In the prototype, we use a columnar format for data serialization.

Distance UDF. The first UDF, is the euclidean distance function described earlier. For the two-column dataset, we apply the distance function on both columns, while for the ten-column dataset we apply the distance function on two columns. Furthermore, for the ten-column dataset, we apply the distance function on all ten fields, calculating the distance between two five-dimensional points (for each row). For each input row, the UDF produces an output row with an additional column, the result of the distance function. The results in Fig. 8a show that the *on-heap* computation offloading approach (Spark with offloading) has similar performance to the standard Spark UDF processing (Spark) only in the case of a two-column dataset. For the ten-column dataset, the offloading approach performs up to 3x worse than the standard method. The performance behavior is explained by the intermediate serialization steps needed for the *on-heap* approach.

MinMax UDF. The second UDF, is an aggregation that finds the minimum and maximum elements of a column. As in the previous experiments, we apply the aggregation UDF to both columns of a two-column dataset, on two columns of a ten-column dataset, and on ten columns of a ten-column dataset. The results are shown in Fig. 8b. The offloading approach (Spark with offloading) outperforms the standard Spark UDF processing (Spark) by a factor of 1.5x for the two column dataset, and by a factor of 1.3x for the ten column dataset when the UDF references only two columns out of ten. When the UDF references all ten columns of the ten-column dataset, the performance of the offloading approach is 4.5x worse than the standard UDF processing. The result of the first two experiments shows that the offloading is beneficial, despite the intermediate serialization overhead. For the experiment with the ten-column dataset, we serialize only the two needed columns, this is why the two methods have similar performance. Nevertheless, when all columns are referenced inside the UDF, the serialization overhead is too high, and the offloading is not beneficial.

Compute-intensive UDF. In our last experiment, we evaluate our prototype for compute-intensive UDFs, by applying a distance function on two columns of the two-column dataset, within an iteration. To increase the UDF workload, we increase the number of iterations.

We include a factor that changes in each iteration, to avoid potential compiler optimizations. The results in Fig. 7 show that for a small workload, the performance between the two approaches is the same, similarly to the previous experiments. However, when the workload is increased, the performance of Spark with offloading gets a speedup of 2.5x.



(a) Euclidean distance UDF (b) MinMax UDF

Fig. 8: Computation Offloading prototype evaluation.

4.3 Discussion

The results of our microbenchmarks show that serialization overhead is not negligible, since it takes up to 80% of the execution time. *Off-heap* computation offloading is the most performant approach, compared to the standard and the *on-heap* offloading approach, since it does not involve any serialization operations. However, as discussed in Sect. 3 it can not be implemented in current dataflow engines without altering system core elements. The benchmarks of our prototype show that it is possible to achieve better performance than the standard processing method, despite the serialization overhead of the *on-heap* approach. This performance is achieved by serializing only necessary fields in the UDF and using a columnar format for processing. We observe that the overhead of intermediate serialization operations caused by the *on-heap* approach is negligible for compute-intensive UDFs, since in that case computation offloading achieves a speedup of 2.5x.

5 Related Work

To enable native performance in dataflow engines, Essertel et al. propose Flare [Es18], a code generation approach. Flare accelerates SparkSQL programs, by carrying out execution and data readers in its own runtime, completely cutting of the Spark execution engine. Rosenfeld et al. propose a similar approach, where SparkSQL programs are transformed to database query plans and executed within a database engine [Ro17]. On the contrary, our

approach does not completely cut off the execution from the dataflow engine runtime, but it is used to offload specific parts of dataflow programs, e.g. UDFs.
Project Tungsten[9] is an effort of Spark developers to enhance the platform with native performance, by introducing binary processing, explicit memory management and code generation. However, Project Tungsten is built on the JVM and does not use a separate native runtime, as we propose in this paper.

6 Conclusion

We proposed an approach to offload computations from a JVM-based dataflow engine to a native environment. To that end, we described two data exchange mechanisms for computation offloading, which allow to process UDFs outside the JVM Heap, and to exploit modern CPU characteristics. Our evaluation showed that computation offloading is beneficial for compute-intensive UDFs and in certain cases for aggregation UDFs.

References

[Al14] Alexandrov, A. e. a.: The stratosphere platform for big data analytics. The VLDB Journal 23/6, pp. 939–964, 2014.

[Cr15] Crotty, A. e. a.: Tupleware:"Big"Data, Big Analytics, Small Clusters. In: CIDR. 2015.

[Es18] Essertel, G. e. a.: Flare: Optimizing Apache Spark with Native Compilation for Scale-Up Architectures and Medium-Size Data. In: 13th USENIX Symposium on Operating Systems Design and Implementation. USENIX Association, pp. 799–815, 2018.

[Ne11] Neumann, T.: Efficiently compiling efficient query plans for modern hardware. Proceedings of the VLDB Endowment 4/9, pp. 539–550, 2011.

[Ng16] Nguyen, K. e. a.: Yak: A High-Performance Big-Data-Friendly Garbage Collector. In: OSDI. Pp. 349–365, 2016.

[Ro17] Rosenfeld, V.; Mueller, R.; Tözün, P.; Özcan, F.: Processing Java UDFs in a C++ environment. In: Proceedings of the 2017 Symposium on Cloud Computing. ACM, pp. 419–431, 2017.

[Za12] Zaharia, M. e. a.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX, pp. 15–28, 2012.

[9] https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html

A Comparison of Distributed Stream Processing Systems for Time Series Analysis

Melissa Gehring[1] Marcela Charfuelan[2] Volker Markl[3]

Abstract:

Given the vast number of data processing systems available today, in this paper, we aim to identify, select, and evaluate systems to determine the one that is better suited to use in conducting time series analysis. Published studies of performance are used to compare several open-source systems, and two systems are further selected for qualitative comparison and evaluation regarding the development of a time series analytics task. The main interest of this work lies in the investigation of the *Ease of development*. As a test scenario, a discrete Kalman filter is implemented to predict the closing price of stock market data in real-time. Basic functionality coverage is considered, and advanced functionality is evaluated using several qualitative comparison criteria.

Keywords: Stream data; Stream processing; Time series analysis; Predictive analytics

1 Introduction

Today's batch and stream processing systems possess vastly different characteristics and are designed to tackle diverse classes of problems. Batch processing systems (BPS), such as MapReduce-based systems, are well-suited for querying stored historical data (a.k.a. data-at-rest). In BPS, data processing is efficient, and administration overhead is minimal (relative to real-time processing systems [SG17]), but they cannot face the constraint of real-time. In contrast, stream processing systems (SPS) are able to process data in real-time (a.k.a. data-in-motion), which is necessary since streaming data is unbounded. Stream processing is performed at the event, window, or micro-batch level [SG17].

Time series data (TSD) are comprised of a series of measurements (e.g., stock market data, medical data, meteorological data [BKF17]), that are taken at a given time-scale (e.g., second, minute). Traditionally, time series management systems (TSMS) and time series data bases (TSDB) have been used to process and store TSD, as discussed in recent surveys (e.g., TSMS [JPT17], TSDB [BKF17]). The frameworks/systems analyzed in these surveys offer querying and data storage capabilities and additionally support some data analytics

[1] Technische Universität Berlin, FG DIMA / Fakultät IV, m.gehring@campus.tu-berlin.de
[2] DFKI / Technische Universität Berlin, marcela.charfuelan@dfki.de
[3] Technische Universität Berlin / DFKI, volker.markl@tu-berlin.de

doi:10.18420/btw2019-ws-21

and may include stream processing capabilities. However, in general, stream processing is not a requirement for TSMS and TSDB. Instead, these systems are designed to handle historical TSD and are ill-suited for real-time processing. Additionally, the majority of these systems are unable to perform advanced analytic tasks, such as prediction, forecasting and similarity search. Recently, real-time SPS, capable of performing stream and batch analytics, have emerged. Thus, we seek to conduct a practical evaluation of these state-of-the-art systems for time series analysis. For our experiments, we implemented a discrete Kalman filter to predict the closing stock market prices in real-time. In this paper, the focus is on the examination of the *Ease of development* of the various implementation steps. Qualitative criteria are defined in order to compare the implementation steps of the predictive task in a streaming test scenario.

The paper is organized as follows. In Section 2 we discuss the selection and evaluation of two open-source SPS for time series analysis using published studies of performance. The qualitative comparison criteria are also defined within this Section. In Section 3 we describe the test scenario and summarize the qualitative assessment that we underwent. In Section 4 we present our conclusions.

2 Comparison Methodology

2.1 Stream Processing Systems

Several criteria are available to characterize SPS. An active field of research is the performance analysis, applied to different systems using quantitative measurements, such as latency and throughput. In this paper, the latest versions of the one-at-a-time based SPS **Apache Storm**[4], **Apache Flink**[5], **Apache Samza**[6] and the microbatch based systems **Storm Trident**[7] and Spark Streaming of **Apache Spark**[8], are compared utilizing several published studies of performance [Wi16, KKR15, Ch16, Ka18]. In Table 1 the characteristics of the previously-mentioned systems are summarized. According to the Yahoo Streaming Benchmark [Ch16], Flink and Storm offer lower latencies, whereas Spark is able to handle higher throughputs, while having somewhat higher latencies. This finding confirms the common statement that there is a difference between micro-batch and one-at-a-time based SPS. According to the Karimov et al. Benchmark [Ka18], (1) Spark is better suited to overcome streams containing skewed data, (2) Flink and Spark are very robust to fluctuations in the data arrival rate in aggregation workloads, (3) Flink is best at handling fluctuations in the data arrival rate on join queries, (4) Flink provides the lowest average latency, (5) Spark (with higher average latency) manages to bound latency best, and (6) Flink has higher throughput with a low latency, in use-cases containing large windows.

[4] http://storm.com
[5] http://flink.com
[6] http://samza.com
[7] http://storm.com/trident
[8] http://spark.com

	Storm	**Storm Trident**	**Spark Streaming**	**Flink**	**Samza**
Processing model	one-at-a-time	micro-batch	micro-batch	one-at-a-time	one-at-a-time
Delivery guarantee	at-least-once	exactly-once	exactly-once	exactly-once	at-least-once
Backpressure mechanism	yes	yes	yes	yes	buffering mechanism
Ordering guarantees	no	between batches	between batches	no	within stream partitions

Tab. 1: Comparison of stream processing systems adapted from [Wi16]

Choosing a processing model always means trading off between latency and throughput. Latency must also be traded against other desirable properties, such as message delivery guarantees and ease of development, as they increase the per data-item overhead (messaging and state replication). Rich processing guarantees at the same time make a system more reliable. Thus, a variety of characteristics can influence the selection of a suitable framework. Although it is still challenging to select an appropriate stream processing system considering performance issues and reliability, two frameworks provide great support. Apache Spark and Apache Flink stand out to bring better performance, unified with the most advanced features in comparison to Storm, Storm Trident and Samza.

Support for	**Storm**	**Storm Trident**	**Spark Streaming**	**Flink**	**Samza**
Languages	J, P, L, R	J, S, C	J, S, P	J, S	J
Event-time	no	no	yes	yes	no
Watermark, allowed lateness, trigger	no	no	partly	yes	no
Windows, joins, filter, aggregations, etc.	no	yes	yes	yes	yes
Stream SQL	no	yes	yes	yes	no

Tab. 2: Functionality across stream processing systems (J: Java, S: Scala, P: Python, L: Perl, R: Ruby, C: Clojure)

Unfortunately, the evaluation criteria did not further analyze the *Ease of development*, which for a beginner in working with SPS is an important property. The usability of SPS can be increased, for example, by offering richer language support as well as the availability of higher-level APIs with support for common functions, like windowing, joins, filtering, aggregations, or stream SQL support. Furthermore, usability is enhanced if advanced features, like event-time processing, watermarking, and triggers are offered. Table 2 summarizes key features commonly associated with the different SPS. Storm supports a wide range of languages but does not provide a high level-API. Samza and Storm Trident offer some advanced features like support for windowing, filtering, etc. but lack of event-time support. Flink and Spark support these advanced features and also provide an API to use SQL within streams. Storm Trident does as well, but a lack of event-time support and

active support by the community [Ka18] exist. Spark took a big step forward by integrating event-time support with Spark Structured Streaming Version 2.1.

Both Apache Spark and Apache Flink stand out, achieving higher performance and offering the most advanced features. Thus, these two systems are selected for qualitative comparison.

2.2 Qualitative Criteria

Our selection of criteria is inspired by the work of Armstrong [Ar01], where various quantitative and qualitative criteria are considered when selecting among various time series forecasting methods. *i) Ease in using available data*, *ii) Ease of use*, *iii) Ease of implementation* and *iv) Flexibility* are among the top ten criteria. These qualitative aspects are affected by the selection of implementation framework. Therefore, in our test scenario, we conduct our qualitative analysis on the basis of these criteria. The following measures are defined for rating the first three criteria: **Extent**, **Simplicity** and **Documentation**. The fourth criterion, *Flexibility*, is considered to evaluate the implementation of more advanced functions in the prediction task. Yet, another measure, the **Adaptability**, is defined to rate the systems regarding the criterion of *Flexibility*. Table 3 shows the various rating levels defined for the measurements and their corresponding meaning.

	Basic functions			*Advanced functions*
Rat.	**Extent**	**Simplicity**	**Documentation**	**Adaptability**
++	Additional features available	Simple and understandable concept. Automated functionality.	Good access and high quality.	Adaptation integrable with low user side implementation overhead
+	Function and essential features available	Clear concept. Reasonable user side implementation overhead.	Access insufficient.	Adaptation integrable with reasonable user side implementation overhead.
0	Function available.	Concept is not clear or disproportionate user side implementation overhead.	Mentioned in documentation	Adaptation theoretically possible but disproportionate implementation overhead.
-	Function not available	Function not available	Documentation not available	Adaptation not possible/ integrable.

Tab. 3: Criteria and Rating for qualitative comparison of *Ease of Development* in Spark and Flink

3 Test Scenario and Qualitative Comparison

Test Scenario Pipeline. In SPS, sources and sinks are typically employed. Data is consumed from a source, then processed within the system before being sent to a sink. The pipeline shown in Figure 1 is employed in the test scenario. Flink and Spark consume their data

from Apache Kafka [9], which is a distributed streaming platform serving as a message broker. A Kafka topic is created, and data is then sent to this topic, where it is persistently stored. The SPS can subscribe to the topic and will then always directly get the newest data published by Kafka in the particular topic. The prediction task in the test scenario requires the implementation of various tasks or functions in several steps. Table 4 shows the main steps organized in two groups (basic and advanced), the corresponding comparison features and the qualitative measurements and ranking for each step. For visualization and verification purposes, the pipeline will be extended by a persisting instance, i.e., the special TSDB InfluxDB [10]. InfluxDB can be used as source for Grafana [11], a modern time series visualization tool.

Fig. 1: Stack for test scenario implementation and example of stock data visualization

Test Data. Stock market data time series will be used for the prediction task. The used data set available at Kaggle[12], contains data from the S&P 500 (i.e., Standard & Poor's 500) index. The dataset contains historical data over a five-year period (2012-08-13 through 2017-08-11) across all current 500 companies listed on the S&P 500 index.

3.1 Qualitative Comparison

Step 1 - Setup. Setting up the pipeline is not challenging for either system. Although both systems support several APIs for various kinds of data processing, batch and streaming, Flink is more focused on streaming, which is also reflected in the documentation. In Spark the user needs to take a deeper look to find the right documentation and information for setting up the streaming environment.

[9] http://kafka.com
[10] http://influxdb.com
[11] http://grafana.com
[12] https://www.kaggle.com/

	Simplicity	Documentation
Flink	+ Different libraries need to be included either working in Java or Scala	++ Short and clear documentation. Easy to find as directly provided in Download area.
Spark	+ Different libraries need to be included, as wide range of libraries exist, all offering interesting functions; there is no centralized library.	+ Short and clear documentation. Harder access due to spread of dependencies, they are located in the documentation areas of different APIs.

Step 2 - Time handling. Handling time is a very important aspect to consider in developing streaming applications. Very often it is desired to process data using the event-time, the time when the event occurred, instead of the processing-time, the time of the machine when the data is processed. Although not all SPS provide support for this type of time processing, both Flink and Spark do. Flink's event-time support has been available for quite some time. It is mature and further offers a wide range of additional features. Event-time support in Spark is a drawback, since it is still a new feature. The integration of event-time support is not in the Spark Streaming library, but rather is in the Spark Structured Streaming library.

	Extent	Simplicity	Documentation
Flink	++ Wide range of features available (watermark, allowed lateness, triggers)	+ Complex concept for event-time support. The user needs to define parts of the concept (TimeStampExtractor) himself or can use predefined ones (not automated).	++ Very clear and helpful documentation with examples and detailed description and easy to find.
Spark	+ Basic event-time support available and watermark configurable.	+ Easy and understandable concept, the user can easily group by window due to saving event-time within columns, but the system breaks (transformation to other data format is necessary)	0 Hard access due to lack of remark about event-time support within Spark Streaming documentation, just in Spark's Structured Streaming documentation (other library). Documented functions are then not directly applicable since other data structure has to be used.

Step 3 - Stream Source Connector. As Kafka is a widely adopted source for SPS it is used to provide access to event streams. The integration is easy to realize within both systems.

	Extent	Simplicity	Documentation
Flink	++ Wide range of features available (deserializer, offset control, etc.)	++ Easy and clear concept for Kafka integration, especially the stream creation (2 actions for configuration, 1 for stream creation as a DataStream).	++ Very clear and helpful documentation with examples and detailed description and easy to find.
Spark	++ Wide range of features available (deserializer, offset, etc.)	++ Easy and clear concept for Kafka integration, especially the configuration (1 action for configuration, 1 complex action for stream creation as a DStream)	+ Clear and helpful documentation but missing examples and description for advanced features. Confusing location of documentation part.

Step 4 - Preprocessing. The functions necessary to apply preprocessing are very basic ones within a stream processing system, as the preprocessing is a step that always needs to be applied. There is almost no difference between Flink and Spark in this step. The functions make it possible to implement user-defined functions that can process custom data items.

	Extent	**Simplicity**	**Documentation**
Flink	++ Very flexible. Expression and self defined functions can be used within transformation functions.	++ Easy concept, easy to use, understandable application. Just 1 action necessary for integration of self defined functions.	++ Good access. Detailed documentation of concepts with understandable examples and explanations.
Spark	++ Very flexible. Expression and functions can be used within transformation functions.	++ Easy concept, easy to use, understandable application. Just 1 action necessary for integration of self defined functions.	+ Good access. Detailed documentation of concepts and understandable examples available, but missing explanations of the examples.

Step 5 - Stream Processing. Stream processing operations are typically applied using event-time. A basic operation is the implementation of sliding windows and transformations afterwards. Within this step, the implementation of a moving window average is compared. As the window is applied using event-time, Spark's new Structured Streaming API needs to be used, which makes operations on multidimensional data possible. Stream functions are easy to implement, as SQL-based operations on streams are available. In Flink the concept is clear, but implementations need to be done manually due to missing support of multidimensional aggregations in predefined functions.

	Extent	**Simplicity**	**Documentation**
Flink	+ No predefined function available for advanced features, in form of multidimensional aggregation on windows, but realizable through UDFs. Sliding window and event-time support available.	+ High complexity of the concept to implement UDFs. 2 actions necessary (1 implementation of user defined function and 1 for application of user defined function).	++ Good access. Detailed documentation of concepts with understandable examples and explanations.
Spark	++ Advanced features available. Aggregation of multidimensional points on windows realizable through SQL based aggregations. Sliding window and event-time support available.	++ Very easy and understandable concept due to SQL based operations. 1 user action necessary for operation.	++ Good access. Detailed documentation of concepts with understandable examples and explanations.

Step 6 - Time Series Analysis. In a first step of the TSD analysis, a version of the discrete Kalman filter is implemented in both systems. For simplicity, assumptions made by Welch et al. [WB01] were used. The code is adapted to fit the use case in the form of processing stock data in streaming fashion. In the second step of the TSD analysis, an online evaluation is implemented. To do so, an error calculation has to be integrated in the Kalman filter algorithm that compares the previous predicted value with the current actual value.

	Adapting to stock market use case	Integrating error calculation
Flink	++ Due to easy implementation as a function applicable as a flatmap transformation on stream the adaptation to the use case was easily realizable, just the input and output had to be customized as well as the calculation to be using closing price.	++ Integration without any issues due to easy implementation of the algorithm as a flatmap function.
Spark	+ The implementation of the Kalman Filter is using a more complex concept. The data source needed to be adjusted so that a DStream could be processed. Due to the stream coming in as batches that adaptation resulted challenging. Furthermore processing of stock data items had to be enabled, as well as outputting predicted items. Calculation had to be adjusted to work on closing price. These adaptations were possible to realize without problem.	++ Easy integration of error calculation into returned object.

Step 7 - Evaluation. Two means of evaluation are considered, i.e., an online evaluation during stream processing and another, conducted after collecting and extracting processed streaming results. Both Flink and Spark, provide different possibilities to analyze data resulting from the application. Both provide the functionality of applying SQL to the data after transforming it to a new data format. The results of the queries were not possible to extract, so further analysis was not possible. Another option tested was to apply aggregation functions on the streams directly in Flink and on the RDDs in Spark. In Flink, the results were calculated continuously on the entire stream; in Spark the calculation only considered items within the RDD. Due to these problems, sinking the results to a .csv file was preferred. This is possible in Flink, but not in Spark due to the partitions made for each RDD.

	Perform evaluation
Flink	+ Wide range of possibilities to perform evaluation available: great support for evaluation directly on stream of errors and export to csv easily possible. Challenging when converting stream to a data set and trying to export the results achieved applying SQL.
Spark	0 Wide range of possibilities to perform evaluation available, but none of them could be applied correctly due to problems handling the RDDs.

Step 8 - Visualization. To integrate the visualization using InfluxDB and Grafana a connector is necessary. InfluxDB connectors exist for both Flink and Spark. The integration of the connectors was analyzed according to their *Adaptability*, considering the integration in general, the customization necessary to write stock data into the data base and the possibility of adding other sinks to other streams (at various stages of TSD processing). The connector available for Flink achieved very good results for all these criteria, Figure 1 shows a screenshot of this visualization. Unfortunately, the connector available for Spark could not be integrated due to missing information and documentation.

	Integration of visualization connector
Flink	++ Available connector provides high flexibility, easy adaptation to the use case, so that custom data formats are accepted. Very easy handling and integration to the use case (Just adding customized sink to DataStream).
Spark	- No easy-to-integrate connector available.

4 Conclusion

Table 4 summarizes the results of Section 3. Generally, both Flink and Spark, provide qualitatively high basic functionality, necessary for the development of basic streaming applications. When developing advanced applications for TSD, Flink appears to be better suited than Spark, since the programming abstraction complexity is lower in Flink. Furthermore, Spark requires the use of a wide range of frameworks to incorporate diverse functionality and requires transformations across several APIs.

The DataStream API provided by Flink is straightforward and thin. Flink's programming abstractions for streaming are simple and ease the development within advanced tasks. Spark's Streaming API cannot directly be compared to Flink's DataStream API, as the essential event-time support is not integrated in the Spark Streaming API. A fair comparison would be using Spark's Structured Streaming API. Due to the relatively new API, many advanced algorithms and implementations are not yet available for Spark Structured Streaming from contributors. This was also an issue during the implementation of the test scenario. Implementations of the Kalman filter were only available for Spark Streaming.

Step	**Criteria**	**Flink**	**Spark**
1 - Setup	Simplicity	+	+
	Documentation	++	+
2 - Time Handling	Extent	++	+
	Simplicity	+	+
	Documentation	++	0
3 - Stream Source Connector	Extent	++	++
	Simplicity	++	++
	Documentation	++	+
4 - Preprocessing	Extent	++	++
	Simplicity	++	++
	Documentation	++	+
5 - Stream Processing	Extent	+	++
	Simplicity	+	++
	Documentation	++	++
Basic functions:		24	20
6a - Time Series Analysis - Adaptation to stock market use case	Adaptability	++	+
6b - Time Series Analysis - Integrating error calculation	Adaptability	++	++
7 - Evaluation	Adaptability	+	0
8 - Visualization	Adaptability	++	-
Advanced functions:		7	2
$\sum$	Total	31	22

Tab. 4: A qualitative comparison of a predictive analytics task between Flink and Spark.

Regarding the evaluation step, the functionality offered by Spark is less straightforward than the functionality offered by Flink. Visualization using the widely adopted tool, Grafana, can be easily integrated using Flink. Based on our analysis, it appears that integrating Flink with

Grafana for time series analysis is the preferred option for stream processing, monitoring, and the visualization of data streams. In Spark, there is no easy connector yet available for Grafana, despite the broad range of adopted tools. The focus of Spark does not lie on its streaming functionality, but more on batch. Flink's focus on streaming is remarkable in the majority of the tasks tested within this work and therewith brings qualitative higher *Ease of development*.

5 Acknowledgments

This work was supported by the German Federal Ministry of Economics and Technology (BMWi) funded *SePiA.Pro* project, under grant *FKZ: 01MD16013*.

References

[Ar01] Armstrong, J. Scott: Selecting Forecasting Methods. In: Principles of Forecasting, volume 30, pp. 365–386. Springer US, 2001.

[BKF17] Bader, A.; Kopp, O.; Falkenthal, M.: Survey and Comparison of Open Source Time Series Databases. In (Mitschang, Bernhard et al., eds): Datenbanksysteme für Business, Technologie und Web (BTW2017) – Workshopband. volume P-266 of Lecture Notes in Informatics (LNI). Gesellschaft für Informatik e.V. (GI), pp. 249–268, 2017.

[Ch16] Chintapalli, S.; Dagit, D.; Evans, B.; Farivar, R.; Graves, T.; Holderbaugh, M.; Liu, Z.; Nusbaum, K.; Patil, K.; Peng, B. J.; Poulosky, P.: Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. IEEE, pp. 1789–1792, 2016.

[JPT17] Jensen, S. K.; Pedersen, T. B.; Thomsen, C.: Time Series Management Systems: A Survey. IEEE Trans. on Knowledge and Data Engineering, 29(11):2581–2600, November 2017.

[Ka18] Karimov, J.; Rabl, T.; Katsifodimos, A.; Samarev, R.; Heiskanen, H.; Markl, V.: Benchmarking Distributed Stream Data Processing Systems. CoRR, abs/1802.08496:12, 2018.

[KKR15] Kejariwal, A.; Kulkarni, S.; Ramasamy, K.: Real Time Analytics: Algorithms and Systems. Proc. VLDB Endow., 8(12):2040–2041, August 2015.

[SG17] Saxena, S.; Gupta, S.: Practical real-time data processing and analytics: distributed computing and event processing using Apache Spark, Flink, Storm, and Kafka. Packt, 2017. OCLC: 1008968663.

[WB01] Welch, G.; Bishop, G.: An Introduction to the Kalman Filter. Technical report, 08 2001.

[Wi16] Wingerath, W.; Gessert, F.; Friedrich, S.; Ritter, N.: Real-time stream processing for Big Data. deGruyter, (58), August 2016.

Lock-free Data Structures for Data Stream Processing

Alexander Baumstark[1]

Abstract: The ever-growing amounts of data in the digital world require more and more computing power to meet the requirements. Especially in the area of social media, sensor data processing or Internet of Things, the data need to be handled on the fly during its creation. A common way to handle these data, in form of endless data streams, is the data stream processing technology. The key requirements for data stream processing are high throughput and low latency. These requirements can be accomplished with the parallelization of operators and multithreading. However, in order to realize a higher degree of parallelism, the efficient synchronization of threads is a necessity. This work examines the design principles of lock-free data structures and how this synchronization method can improve the performance of algorithms in data stream processing. For this purpose, lock-free data structures are implemented for the data stream processing engine Pipefabric and compared with current implementations. The result is an improvement for the tuple exchanging between threads and a significant improvement for the symmetric hash join algorithm based on lock-free hash maps.

Keywords: Concurrent Data Structures, Lock-Freedom, Stream Processing, Parallelism

1 Introduction

This work investigates the design principles of lock-free data structures and examines their potential use for data stream processing. The article from [SÇZ05] provide eight requirements for (real-time) data stream processing. Three of these requirements are directly dependent on the system architecture, the effectiveness of the used algorithms and the performance. Stream processing requires algorithms that produce constant progress as a minimum. Due to the fact that the data appears mostly in form of unbounded data streams and high costs for storage operations, the algorithms must be able to process data on the fly. Other requirements are stable and robust algorithms that are not prone to errors, because high availability is a must-have within data stream processing. Modern approaches take advantages of the multithreading paradigm to achieve this requirement, for instance, with concurrent operations on partitioned streams. The key component of such stream processing algorithms are concurrent data structures, for example, linked list data structures for the stream partitioning, where selected elements of the original stream are stored. In order to obtain consistency and a high degree of parallelism efficient synchronization methods are needed. Conventional techniques of thread synchronization make use of blocking

[1] TU Ilmenau, Databases and Information Systems Group, Helmholtzplatz 5, 98693 Ilmenau,
alexander.baumstark@tu-ilmenau.de

doi:10.18420/btw2019-ws-22

mechanisms like locks and mutual exclusions (*lock-based*). The major disadvantage of these methods is that they can suffer from problems like deadlocks, livelocks or priority inversion. Since the critical sections of shared resources cannot be executed in parallel by multiple threads, the possible degree of parallelism is decreased.

A different technique is thread synchronization without locks, called non-blocking synchronization. Basically, there are three classes of non-blocking methods: *obstruction-free*, *lock-free* and *wait-free*. The difference between these classes lies in the guarantee they provide for the progress. In short, lock-free synchronization guarantees that at least one thread makes progress, whereas wait-free makes sure that all threads do so. Obstruction-free synchronization is the weakest class and can only guarantee that an isolated thread makes progress. None of the mentioned problems of lock-based synchronization can occur with non-blocking implementations. This can lead to a higher degree of parallelism which may result in a performance gain. Certain modern database systems already use lock-free algorithms in order attract with their achieved performance ([Re], [Me17]). The goal of this work is to examine whether or not the benefits of lock-free synchronization are attainable in data stream processing. The primary research questions of this work ([Ba18]) can be summarized as follows: **(1)** What design principles exist for lock-free data structures? **(2)** For which data structures does a lock-free design exist? **(3)** How does lock-free synchronization affect the overall performance, especially for the use case data stream processing? Can this method fulfill the requirement of low latency and high throughput? The data stream processing engine Pipefabric[2] is used for benchmarks, in order to give an answer to the third research question.

To summarize, we make the following contributions:

1. We improved the tuple exchange algorithm in Pipefabric with lock-free synchronization.
2. We proposed a lock-free hashmap design that supports multiple elements with equivalent key, similar to the C++ unordered multimap structure.
3. We improved the scalability and performance of the symmetric hash join algorithm in Pipefabric.

2 Design Principles of Lock-free Data Structures

The conventional way to synchronize data structures is to use locks and mutual exclusions. Lock-free synchronization takes another approach and uses atomic operations, memory barriers and fences to synchronize and guarantee consistency. Atomic operations are indivisible and uninterruptible instructions [HP06]. These operations can be compared with transactions from database systems. Transactions follow the *ACID* property [HR83], which can be adapted to atomic operations. The ACID property guarantees that every

[2] `https://github.com/dbis-ilm/pipefabric`

operation must be uninterruptible (atomicity) and that every operation from a consistent state is followed by a consistent state too (consistency). Operations are executed concurrently but the effect of these is the same as if the operations would be executed sequentially (isolation). Each operation remains after it has committed (durability). Due to these properties, synchronization can be done without the use of locks. There are two classes of atomic operations: The first is the class of atomic read and write operations. The other class is for complex atomic read-modify-write operations, like *compare-and-swap* (CAS)[3] or *fetch-and-add*.
CAS takes three arguments: a memory location, the expected value of the memory location and a new value. Only if the value of the memory location matches with the expected value, the new value will be stored in the memory location. If the compare-and-swap is successfully executed, it returns true, otherwise false. The failure of a CAS operation means that a thread changed the value in the interim, so the expected value does not match with the value of the memory location. A common technique is to execute the CAS operation (with refreshed expected values) within a loop until it is successful. [He91] has shown that the consensus number of the CAS operation is unbounded with the consequence that CAS can implement all other atomic operations.
Similar to the back-off strategies of network protocols that serve to limit the rate of retransmission, back-off strategies can be used to limit the rate of failed CAS operations. The reason for using a back-off strategy is that a high rate of successively failed CAS operation causes unnecessary CPU time, which could be used by other threads to make progress. Consequently, the use of a correct back-off strategy can increase the performance of a lock-free data structure [Kh15]. An example is the *elimination back-off strategy* for a lock-free stack [HSY04]. It is based on the following observation. If a pop operation follows a push, the state of the stack does not change. Therefore, a pair of push and pop operations can meet at a different location to exchange data, without performing actions on the stack.

Another problem in the context of CAS and lock-free synchronization is known as the *ABA problem*. It is defined as a false-positive execution of a CAS-based operation through an unobserved change of a memory location in the interim [DPS10]. A CAS operation cannot consider a change from the value A to B and back to A. Therefore, the CAS operation falsely executes its swap and returns a true as a result. It is clear that this behavior, caused by the ABA problem, can lead to inconsistency and must be prevented. [MS96] described a efficient solution to the ABA problem with tagged pointers. After each successful CAS operation the tag of the pointer is incremented and each modification can be considered. Other approaches use reference counters described by [Va95] or hazard pointers [Mi04].

3 Lock-free Implementations

Stream processing operations rely on concurrent data structures. One of the research questions of this work is: For which data structures does a lock-free design exist at all? The

[3] An equivalent instruction (pair) for Load/Store architectures is load-linked/store-conditional (LL/SC).

answer is simple: There are no real restrictions. Several thread-safe lock-free data structure designs exist for almost all data structures. [Tr86] published a simple lock-free stack design. It is assumed that this is the first published non-blocking implementation. Another classical lock-free design that is implemented in a variety of libraries and systems, is known as the (multi-producer, multi-consumer) Michael and Scott queue [MS96]. Single-producer and single-consumer queues are widely implemented by multi-threading libraries, for example, Intel's Threading Building Blocks (*TBB*)[4], Facebook *Folly*[5] or by C++ *Boost* Libraries[6]. These implementations are based on lock-free ringbuffer data structures and achieve incredible fast execution performance.
Join operations in stream processing use hash maps to probe their entries against others to find a match. A disadvantage in conventional concurrent implementation is, that the whole hash map has to be locked to obtain consistency. Several lock-free designs exist for hash maps, like [FLD13] based on multi-level arrays or [Mi02] based on linked lists, only to name a few. [BP12] published a lock-free implementation of a B^+-tree which is an alternative to blocking lock-coupling techniques. The next section examines lock-free data structures in the use case of the data stream processing engine Pipefabric.

4 Use Case: Data Stream Processing

As already mentioned, the key requirements for data stream processing are high throughput and low latency. The goal of this section is to examine whether or not these requirements are more attainable with lock-free synchronization. Due to the fact that lock-free synchronization allows theoretically a higher degree of parallelism, it is expected that algorithms that rely on this technique achieve better performance results than their equivalent blocking approaches.

4.1 Pipefabric

Pipefabric is a data stream processing engine, developed by the Database and Information Systems Group at the TU Ilmenau. It is open source, written in C++, supports different network protocols like ZeroMQ, MQTT or AMQP and can get tuples from Apache Kafka servers or RabbitMQ. For multi-core machines there are several operations available in order to enhance the stream processing. A partition operator can split the data streams, so that each partitioned stream can be processed concurrently. Sub-stream can be merged into a single-stream again. The supported window operations are the tumbling and sliding window. Elements of data streams are represented in Pipefabric as a tuple data structure. These tuples and their components can be processed with several operations. Another component of Pipefabric is the topology, an interface for the data stream processing pipelines, similar to the implementation from Apache Spark.

[4] https://www.threadingbuildingblocks.org/
[5] https://github.com/facebook/folly
[6] https://www.boost.org/doc/libs/1_63_0/doc/html/lockfree.html

4.2 Benchmark System

For the benchmarks that are to be done in this section, an Intel Xeon Phi KNL 7210 processor is used with 64 cores and four threads for each core. The base frequency runs on 1.3 GHz and can boost up to 1.5 GHz (turbo). Each core owns an L1 cache of 32kB. This hardware setup allows to run a benchmark for high scalability and concurrency at the same time. The Intel compiler version 17.0.6 is used because it offers better results in benchmark scenarios, compared with gcc. Additionally, the code is compiled with the supported AVX-512 instruction set, but without further code optimization that would take full advantage of these instructions.

4.3 Tuple Exchanging

Sometimes data needs to be exchanged between two threads, for example, in exchanging information of the status or tuples for partitioning. An approach to realize this is to implement it with a buffer, with a single reader and writer. The underlying data structure of the current implementation is the STL queue, protected with locks and condition variables. In the first benchmark, the current queue data structure for tuple exchanging is compared with equivalent lock-free variants from the C++ Boost libraries, Intel's TBB and Facebook Folly.

For the first scenario, the producer and consumer has to process five million tuples/elements in order to simulate an unbounded situation. In the second scenario, the maximum size of the lock-free queues is reduced to the size 1024, in order to show results with realistic parameters. A naive waiting back-off strategy is used in case of a full or empty queue.

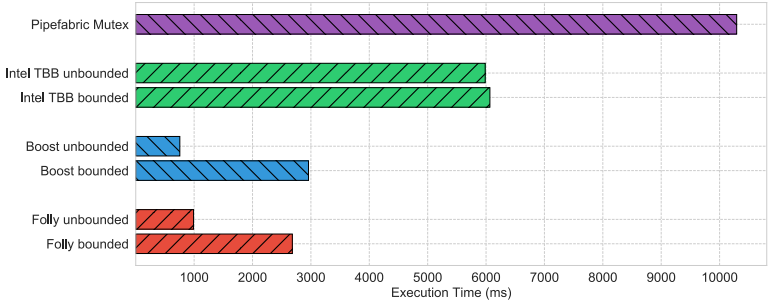


Figure 1: SPSC queue benchmark: execution time, unbounded and bounded

The results in Figure 1 show clearly that the non-blocking queues outperform the lock-based implementation from Pipefabric. Each thread in the blocking technique is executing its operation alternately in a way that no real parallel execution is possible and the amount of time that a thread waits for an unlock of the critical section is unused. The non-blocking implementations use fast atomic load and store instructions, the Boost and Facebook Folly implementations are even wait-free. Intel TBB's fine-grained lock queue is in the medium

range in this benchmark. A slow consumer, like in the bounded situation, can decrease the overall performance but is still faster than the blocking approach. It is recommended to implement the Boost queue for the tuple exchange, because of its speed and the reason that the Boost library is already used in Pipefabric. This benchmark shows clearly that lock-free queues can improve the tuple exchanging procedure significantly. The next benchmark examines in which way it is attainable in complex stream processing algorithms like the symmetric hash join.

4.4 Symmetric Hash Join

A commonly used join algorithm in data stream processing is the symmetric hash join, which is also available on relational database systems. The symmetric hash join algorithm for stream processing continuously generates results while tuples from the streams arrive. Figure 2 shows the idea of a symmetric hash join algorithm with two data streams sliced into windows and joined after the algorithm into a single stream.

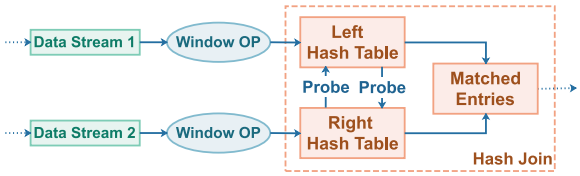


Figure 2: Symmetric hash join operation.

The symmetric hash join algorithm processes two input streams, denoted as left and right input. After each arrival of a tuple, either on the left or the right input, it is inserted in the corresponding hash map. In a next step the hash maps probe their entries against the others for a match. Entries with no match are removed from the hash map. The gained entries with a match are forwarded to the following operator as a single data stream.

For a symmetric hash join algorithm, two hash tables are mandatory for the left and right stream elements. Another requirement is that multiple stream elements are mapped to the same key. Therefore, the buckets of the hash map must be able to contain multiple elements, and the probing must also iterate through all available elements in the bucket. The algorithm is organized in three steps: *(1)* Insert the tuples in the corresponding hash table, or remove them if they are outdated, *(2)* probe for possible join partner in the other hash table and *(3)* the actual join of the tuples.
The implementation of the symmetric hash join algorithm in Pipefabric is based on the STL data structure unordered_multimap. This structure is an STL container, that contains key-value pairs, similar to the unordered_map structure but with the addition that elements can have equivalent keys. The internal structure of the multimap is a hash map which supports forward iterators with an average constant-time complexity. In order to guarantee thread-safety in a concurrent execution, each operation of the data structure is protected with a lock. The lock-free symmetric hash join is more challenging to realize. This relies on

the bucket structure, which allows that multiple values have the same hash key. A lock-free hash map with this bucket property needs combined data structures for the hash map and bucket structure. The following lock-free hash map and bucket implementation is based on the lock-free linked list structure by [Ha01].

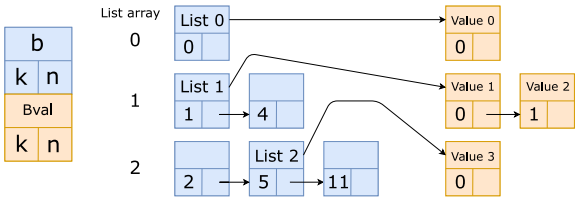


Figure 3: Lock-free hashmap supporting same key elements
A node in the hashmap list contains a key (k), pointer to the next node (n) and a pointer to the bucket as value (b). The bucket node contains the actual value (Bval).

Each node of the list consists of a key, a value and a pointer to the *next* node. A key makes it possible to distinguish between entries located at the same index. The hash map itself is an array of n lists, where n is the size of the hash map with the hash function $h(x) = x \mod n$. In order to insert a new element, the *insert* operation computes the hash of the key to find the corresponding list within the array. Then, the new element is inserted into the bucket structure of the node with the corresponding key. *Find* hashes the key, iterates through the corresponding list, compares each key and returns a pointer to the bucket if the key is found. Additionally, the bucket structure is based on the same lock-free linked list structure with the addition of an atomic size counter, that increments on each insertion with an atomic fetch-and-add operation. A new element is inserted into the list with the current value of the size counter as its key (see Figure 3).

Head and *Tail* pointers are used to iterate through all elements, by swinging to the *next* element of the node. The general behavior of this lock-free design (named Lock-free/Linked List in Figure 4) corresponds to the STL unordered multimap structure. Equivalent implementations based on lock-free skip lists (named Lock-free/Skip List in Figure 4) and a blocking implementation based on the unordered multimap from Intel TBB are used for reference in the benchmark.

In the benchmark of Figure 4, two tuple generators publish tuples into the left and right sliding window. The benchmark measures the execution time of the concurrent symmetric hash join (with up to 256 threads) with constant distributed 10.000 tuples in total. The buckets are preallocated in order to measure the relevant parts of the symmetric hash join.

The benchmark result given in Figure 4 shows clearly that these two approaches differ in performance. When applying the execution with two threads, the blocking hash map is slightly faster than the lock-free one. The reason for this lies in the implementation of the lock-free hash map. A lock-free insert operation generally needs more instructions than an equivalent lock-based implementation, because it operates within a loop with a CAS

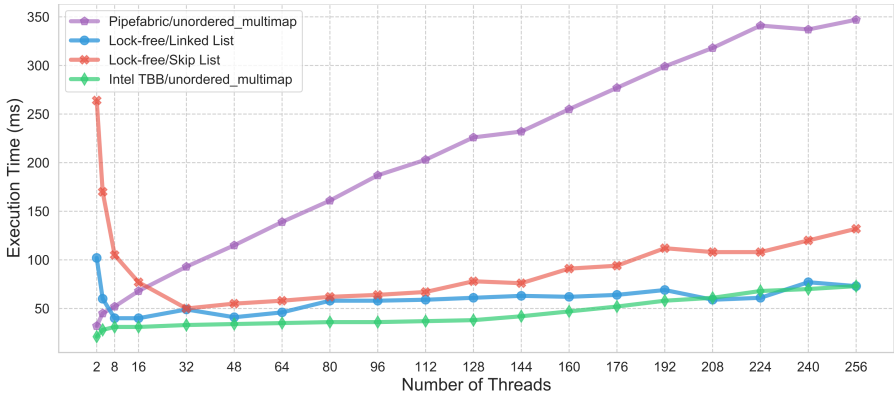


Figure 4: Symmetric Hash Join Benchmark: Execution Time

instruction. An insert operation of a lock-free data structure needs possibly more attempts to add an element into the lists, because CAS may fail, unlike the lock-based approach.

With the increasing number of threads the performance of the lock-based approach decreases drastically. This observation is based on the fact that only one thread can access a critical section. However, the lock-free implementation can guarantee that at least one thread makes progress resulting in a higher degree of parallelism and throughput. A benefit of the implementation based on singly linked lists is, that a constant number of stream elements is processed in an approximately constant time. Another observation of this benchmark is, that an optimized solution with fine-grained locks achieves the same or, in some situations, even better performance results than a lock-free implementation. Due to the reason that this design uses small critical sections, it achieves a similar degree of parallelism and throughput. It should be mentioned that the lock-free implementations are not further optimized. With additional lock-free techniques, back-off strategies and other optimizations even better results are possible.
The reason for the poor performance of the skip list-based structure lies in the probabilistic behavior: higher levels (express lanes) are created randomly. Consequently, the threads can not take full advantage of these in the worst case. At higher thread numbers, this implementation scales similar to the linked list structure, because the additional layers are created by multiple threads. This approach is not recommended for practical usage and just shown for reference, due to additional overhead for the higher layers.

5 Conclusion

The results of the benchmarks show that lock-free implementations can achieve similar results and at higher thread numbers even better results than the lock-based implementations. Pipefabric uses a blocking implementation of a concurrent queue in order to exchange tuples

between threads. Every modification on that queue can only be executed by one thread at a time, which is the major disadvantage of blocking implementations. An equivalent lock-free implementation allows that every thread can access the data structure simultaneously. This can boost the tuple exchanging process up to a a tenth, compared to the blocking variant, if the consumer is as fast as the producer. In case of a slow consumer, where the queue is frequently full, the lock-free implementations are still significantly faster. The stream processing can benefit from the higher degree of parallelism at the tuple exchanging, which lead to a higher throughput for stream operations, for instance, window operations or joins.

Another significant performance boost can be achieved with a lock-free symmetric hash join operation. The benchmark results have shown that the lock-free implementations are slower at lower thread numbers but faster and scale very well at higher thread numbers. Reasons for the results at lower thread numbers are the additional consistency checks before an actual stream processing operation takes place. The implemented lock-free data structures can also be used for other stream processing operations in order to improve the degree of parallelism, for example, the scale join or for the window operations. However, another observation is that optimized fine-grained locking methods achieve better results at lower thread numbers, due to small critical sections and consequently more parallelism. Hence, lock-free synchronization is not the so-called silver bullet in thread synchronization.

To summarize it all, lock-free designs can improve the performance of concurrent operations and deliver scalable and robust algorithms, which are free from problems like deadlocks and priority inversion. Thanks to these properties it is possible to achieve reliable latency and throughput in data stream processing and exceed the performance of blocking designs. This work has shown that lock-free implementation can fulfill the demands of data stream processing algorithms.

References

[Ba18] Baumstark, A.: Lock-free Data Structures for Data Stream Processing, Bachelor's Thesis, TU Ilmenau, Aug. 17, 2018.

[BP12] Braginsky, A.; Petrank, E.: A lock-free B+ tree. In: Proceedings of the 24th ACM symposium on Parallelism in algorithms and architectures - SPAA '12. ACM Press, 2012.

[DPS10] Dechev, D.; Pirkelbauer, P.; Stroustrup, B.: Understanding and Effectively Preventing the ABA Problem in Descriptor-Based Lock-Free Designs. In: 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE, 2010.

[FLD13] Feldman, S.; LaBorde, P.; Dechev, D.: Concurrent multi-level arrays: Wait-free extensible hash maps. In: 2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). IEEE, July 2013.

[Ha01] Harris, T. L.: A Pragmatic Implementation of Non-blocking Linked-lists. In: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 300–314, 2001.

[He91] Herlihy, M.: Wait-free synchronization. ACM Transactions on Programming Languages and Systems 13/1, pp. 124–149, Jan. 1991.

[HP06] Hennessy, J. L.; Patterson, D. A.: Computer Architecture: A Quantitative Approach, 4th Edition. Morgan Kaufmann, 2006, ISBN: 0-12-370490-1.

[HR83] Haerder, T.; Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys 15/4, pp. 287–317, Dec. 1983.

[HSY04] Hendler, D.; Shavit, N.; Yerushalmi, L.: A scalable lock-free stack algorithm. In: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures - SPAA '04. ACM Press, 2004.

[Kh15] Khiszinsky, M.: Lock-Free Data Structures. The Evolution of a Stack, Feb. 24, 2015, URL: https://kukuruku.co/post/lock-free-data-structures-the-evolution-of-a-stack/, visited on: 06/14/2018.

[Me17] MemSQL: How does MemSQL's in-memory lock-free storage engine work?, 2017, URL: https://docs.memsql.com/introduction/latest/memsql-faq/#how-does-memsql-s-in-memory-lock-free-storage-engine-work, visited on: 08/04/2018.

[Mi02] Michael, M. M.: High performance dynamic lock-free hash tables and list-based sets. In: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures - SPAA '02. ACM Press, 2002.

[Mi04] Michael, M.: Hazard pointers: safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems Record 15/6, pp. 491–504, 2004.

[MS96] Michael, M. M.; Scott, M. L.: Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing - PODC '96. ACM Press, 1996.

[Re] RethinkDB: How are concurrent queries handled?, URL: https://www.rethinkdb.com/docs/architecture/, visited on: 08/04/2018.

[SÇZ05] Stonebraker, M.; Çetintemel, U.; Zdonik, S.: The 8 requirements of real-time stream processing. ACM SIGMOD Record 34/4, pp. 42–47, Dec. 2005.

[Tr86] Treiber, R. K.: Systems programming: Coping with parallelism. IBM Research Center, 1986.

[Va95] Valois, J. D.: Lock-free linked lists using compare-and-swap. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing - PODC '95. ACM Press, 1995.

An Actor Database System for Akka

Sebastian Schmidl, Frederic Schneider, Thorsten Papenbrock[1]

Abstract: System architectures for data-centric applications are commonly comprised of two tiers: An application tier and a data tier. The fact that these tiers do not typically share a common format for data is referred to as *object-relational impedance mismatch*. To mitigate this, we develop an actor database system that enables the implementation of application logic into the data storage runtime. The actor model also allows for easy distribution of both data and computation across multiple nodes in a cluster. More specifically, we propose the concept of *domain actors* that provide a type-safe, SQL-like interface to develop the actors of our database system and the concept of *Functors* to build queries retrieving data contained in multiple actor instances. Our experiments demonstrate the feasibility of encapsulating data into domain actors by evaluating their memory overhead and performance. We also discuss how our proposed actor database system framework solves some of the challenges that arise from the design of distributed databases such as data partitioning, failure handling, and concurrent query processing.

Keywords: Actor Model; Database System; Akka; Distributed Computing; Parallelization

1 Introduction

Today's applications need to process data at ever growing rates. Regardless of its origin and kind, data is ever growing and needs to be stored and queried. Cluster or cloud deployments and multi-core hardware architectures allow scaling application logic in terms of computational power. Traditional data management systems, however, are at risk of becoming the bottleneck in data-centric software systems, because the separation into data and application tier costs performance, impacts code maintainability, and increases error susceptibility.

The performance costs are due to the fact that relational database management system (RDBMS) model their data in terms of relations while applications usually model the data as objects. The translation of relations into objects and vise versa is known as the *object-relational impedance mismatch* and requires some additional effort. The use of object-relational mapping (ORM) tools, such as Hibernate for Java or Active Record for Ruby on Rails, is a convenient yet expensive approach to provide a middle tier for the translation. Some key-value stores solve the impedance mismatch more elegantly, but they suffer from worse join and aggregation costs. Furthermore, code maintainability decreases

[1] Hasso Plattner Institut, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, {sebastian.schmidl, frederic.schneider,thorsten.papenbrock}@student.hpi.de

doi:10.18420/btw2019-ws-23

when stored procedures are being used to push application logic closer to the data, i.e., into the data tier for performance reasons, and the error susceptibility increases, because large, monolithic RDBMSs suffer from hand-crafted, non-standardized, and inconsistent attempts to fault-tolerance, parallelization, data encapsulation, workload distribution, and replication. The actor programming model, on the contrary, offers an effective solution for all these challenges.

Using the actor programming model to fuse application and data tier is a concept originally proposed by Shah; Salles [SS17a] and tested on the Orleans actor framework. The authors call for a new paradigm by designing a scalable data storage solution using the actor model. The core primitive in this model are actors, which are objects comprised of state and behavior that execute computational tasks concurrently. Individual actors communicate with each other exclusively via asynchronous message passing. Incoming messages are stored in a mailbox allowing for the separate and independent processing of each message. An actor's internal state is only available to said actor, which encourages a shared-nothing system architecture. The self-contained nature of actors and the fact that actors provide a lock-free concurrency model, allows for naturally scaling out applications and systems [Ve15].

We build on this idea and present an application development framework for actor-based data-centric applications. In contrast to Shah; Salles [SS17a], our actor database system targets the Akka actor framework that offers different mechanisms for fault tolerance and actor lifecycles than Orleans. In detail, we make the following contributions: We introduce domain actors (Dactors) to model application data in an Akka-based actor database system. Similarly to the work of Shah; Salles on reactors [SS17b], Dactors encapsulate application data and logic. Since these actors are not part of a dedicated database runtime, but are defined using the application framework, data objects share the same representation throughout business logic and data storage. This approach bridges the aforementioned impedance mismatch between data and application logic tier. In contrast to reactors, Dactors are not relational entities, but employ relational structures internally. Dactor state can be manipulated via an SQL-like interface. To define application logic relying on data contained within multiple Dactors, we provide the concept of Functors, which make the usage of asynchronous and concurrent computations explicit.

To our knowledge, we present the first implementation of this concept using the Akka framework. Comparable approaches are discussed in Sect. 2. In Sect. 3, we outline our concept for an actor database system in more detail, before presenting the results of our experimental evaluation using our framework in Sect. 4. We offer a concluding statement about this and future work in Sect. 5.

2 Related Work

The actor model that we introduced in the previous section has been implemented as libraries for various programming languages. The most popular implementations are Erlang's in-build

actors, the Orleans framework for .NET, and the Akka framework for Java [Ar07; Be14; Li18a]. Although most research in the area of actor-based database systems targets the Orleans framework, Akka is probably the most widely used actor model implementation – not least because of the popularity of Java and the fact that it is used in frameworks such as Apache Spark and Apache Flink. For this reason and because Akka differs in various aspects from Orleans, we focus on this framework in our research.

Despite their popularity for building distributed applications, all current actor programming frameworks lack database-like state management capabilities, specifically for data-centric applications. The developer has to decide how to handle state persistence and how to satisfy failure, replication, and consistency requirements of an application – the actor model implementations neither provide atomicity nor consistency guarantees for state across actors. Shah; Salles [SS17a] therefore stated the need for state management in actor systems and proposed to integrate database functionality into the actor model. The authors postulate that *Actor Database Systems* should be designed as a logical distributed runtime with own state management guarantees. More specifically, their manifesto specifies four tenets that define an Actor Database System [SS17a]:

Tenet 1 Modularity and encapsulation by a logical actor construct
Tenet 2 Asynchronous, nested function shipping
Tenet 3 Transaction and declarative querying functionality
Tenet 4 Security, monitoring, administration and auditability

Our actor database system (currently) covers the first two of these four tenets: For **tenet 1**, we use actors to achieve a modular logical model for data encapsulation. Dactor instances are in-memory storekeepers for application data. They satisfy the actor definition and support high modularity. For **tenet 2**, Dactors provide a model for the concurrent computation of predefined functionality that enforces locality of data accesses. All communication between Dactors is asynchronous to leverage the advantages of increasingly parallel hardware. Our concept of Functors allows for the definition of functionality using multiple actors' data. To meet **tenet 3**, Functors and Dactor behavior can be defined in a declarative way. Due to their single-threaded computation model, Dactors basically enforce internal consistency by default. In principle, Functors also enable the implementation of further transaction protocols to ensure inter-Dactor consistency guarantees.

In contrast to the actor database system prototype introduced in [SS17a], we developed our prototype using the Scala programming language and the Akka framework (instead of .NET and Orleans). In contrast to Orleans and its convenient *virtual actors*, Akka offers more control over an actor's lifecycle, has a more explicit failure handling, and models actors in hierarchies – aspects that enable more fine-grained control over the system but also demand for more thorough architectural system design decisions.

Most related research in the field of actor database systems has been presented in conjunction with the Orleans framework and the Erlang programming language [Be17; EB16; SS17b].

Biokoda [Bi18] takes another approach and encapsulates a full relational SQL database inside an actor. Cardin [Ca17] uses actors to build a scalable key-value store and others use the actor model to build soft caching layers and cloud applications for various purposes [Er18; Li18b; NE18].

3 Domain Actor Database Framework

Our actor databases system consists of two building blocks: *domain actors* and *Functors*. These two concepts allow for the definition of application data within the application itself. Since both are based on actor model principles, they make the database system modular, cloud-ready, and scalable. In this section, we introduce both domain actors and Functors.

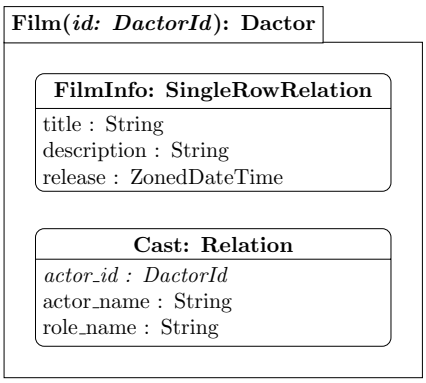
3.1 Domain Actors – Encapsulation of Data

Similar to Shah; Salles [SS17b], we introduce a special type of actor, called Dactor, that acts as an application-defined scaling unit. Dactors can be used to model application-domain objects and encapsulate the object's state and application logic in an actor. Using actors for this enforces technical encapsulation of state access due to the purely private state in actors and the need of explicit asynchronous messaging between the actors. The encapsulation also makes it easier to reason about state changes, bugs, and other failures, as only code within the Dactor can change the corresponding state.

In-memory data contained within a Dactor instance is managed in a data structure called *relation*. One Dactor can contain multiple relations. A relation is, similarly to a table in the relational database model, defined as a multiset of tuples following a predefined schema. Relations provide an SQL-like interface to query and manipulate the contained data, so known and proven syntax and semantics can be used to define Dactor-behavior. Relations form a typed, Dactor-internal data model. Using Dactors to implement a database leads to a modeling approach that is different to Entity-relationship modeling. The following example discusses the conceptual differences between the two in more detail.

We consider the example of a web application with information on movies similar to the imdb.com or rottentomatoes.com websites. A standard query for those websites is to display a film with its description and cast. A traditional data layout might be comprised of two entities: *Film* containing the film's ID, title, description and release date and *Actor* containing the actor's ID and name. Those two entities might be in a N-to-M relation (*Cast*) with an attribute showing the actor's role in the film. In contrast to the relational model, our model, shown in Fig. 1a, consists of one Dactor type and two relations and is denormalized. The information contained in the *Actor* entity is distributed across the `Cast` relations. This allows us to answer the standard queries from one single actor instance without needing to join the answers from different, possibly physically distributed Dactor instances.

This approach to layout an application's data results in much smaller data sizes per Dactor compared to typical database tables and enables many business-logic-driven approaches to scaling, data partitioning, and caching. The trade-off, however, is a large number of Dactor instances and a (partially) denormalized schema.



(a) Graphical representation of the `Film` Dactor type definition.

```
class Film(id: DactorId) extends Dactor(id) {
  override protected val relations = {
    Film.Info -> SingleRowRelation(Film.Info),
    Film.Cast -> RowRelation(Film.Cast)
  }
  override def receive: Receive =  //Dactor behavior
}
object Film {
  object FilmInfo extends RelationDef {
    val title = ColumnDef[String]("title")
    val description = ColumnDef[String]("description")
    val release = ColumnDef[ZonedDateTime]("release")
  }
  object Cast extends RelationDef {
    val actorId = ColumnDef[DactorId]("actor_id")
    val name = ColumnDef[String]("actor_name")
    val rolename = ColumnDef[String]("role_name")
  }
}
```

(b) Example code using our framework.

Fig. 1: `Film` Dactor type definition with two relations from the example application.

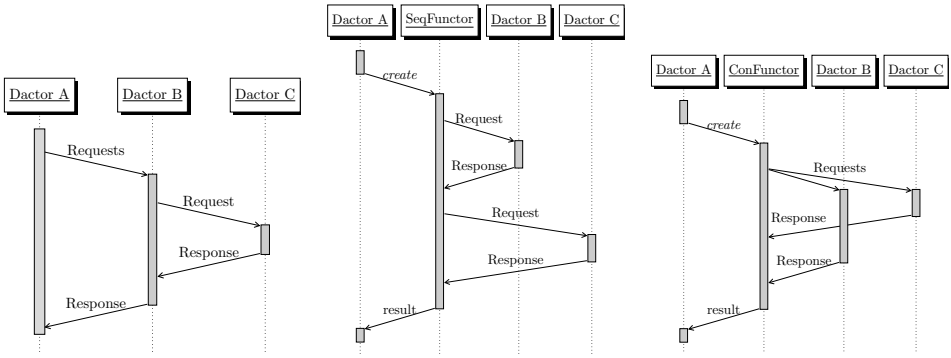
As Dactors not only contain data, but also the corresponding domain logic, computation is executed concurrently. Actors provide single-threaded semantics, which makes enforcing constraints on data stored inside one Dactor easy. While state querying and modification within Dactors is possible in a declarative way, the application developer can explicitly define the communication across all kinds of actors via asynchronous messages. The explicit messaging differentiates Dactors from Shah; Salles [SS17b]'s reactors, as reactors can be used as relational entities and hide the message passing from the developer.

To illustrate the definition of a `Dactor` in code, we show the definition of `Film`'s data model in Fig. 1b. Developers can model the application's domain objects by defining Dactor types as subclasses of the framework-provided `Dactor` class in a declarative way. Instances of such user-defined Dactor types are managed by the framework and are available for messaging in a consistent namespace. Using the column's predefined data types, all functions support compile-time type-safety. Due to the Dactor system sharing the application's runtime and programming environment, these data or object types are equal to the types handled in any application logic. Thus, this approach helps eliminate the impedance mismatch between application logic and data tier with regard to handled data types and object (de-)serialization.

3.2 Functors – Encapsulation of Queries

Dactors can answer queries via explicit, asynchronous messaging, i.e., they answer a query with their local data. Sometimes, however, queries need be answered by several actors. In

such cases, it makes sense to encapsulate the processing in a new, short-living actor that we call Functor. Functors are the framework's concepts that enable inter-Dactor communication and computations. They communicate with (usually multiple) Dactors, track the completion of a query, handle the state of pending requests, and resolve failure cases. Every actor can create a new Functor to encapsulate multiple requests to Dactors. The Functor handles the message processing and sends the final result or a failure message back to its creator. Since all Akka actors live in hierarchical parent-child relationships, Functors are always created by an actor as a child. This Akka-specific hierarchical relationship enables notifying the calling actor even in case of unforeseen crashes of the Functor themselves, which in turn allows to trigger error handling, e.g. retrying the Functor execution. In our actor database system, we consider three messaging patterns for inter-Dactor communication, which are shown in Fig. 2. These patterns are provided as messaging primitives by the framework and can be combined to create more complex message flows and computational models:



(a) Cascading Computation (b) Sequential Computation (c) Concurrent Computation

Fig. 2: Inter-Dactor communication patterns. Gray bars indicate that an actor holds state that is related to the showed message flow.

Cascading Computation is a pattern where a high-level message to an initial Dactor (Dactor A) triggers successive messages to other Dactors, which are hidden from the original requester. Following Dactors can also trigger further messages themselves. As one can see in Fig. 2a, this pattern is comparable to function calls in Object-oriented Programming. But contrary to simple function calls, messages in this pattern are sent asynchronously. This means that the requesting Dactor has to manage the state of pending responses. This clutters the domain logic in the Dactor and leads to complex and error-prone code. If used sparsely, this pattern supports separation of concerns and the tell-don't-ask paradigm.

Sequential Computation is used for queries that consist of consecutive steps, where each step depends on the previous step's result, such as filter chains. This pattern can be implemented via the aforementioned Functors. Using a Functor to process the consecutive steps of the computational chain relieves Dactor A from dealing with intermediate state, because it is managed by the Functor. Each Functor deals with only one request-response pair at a time, which leads to simple state and processing logic for the Functor itself.

Concurrent Computation is a another messaging pattern based on Functors to encapsulate the processing of multiple request-response pairs. The concurrent Functor sends messages to several Dactors in parallel and collects the results when they are finished to forward them to its creator. It allows for highly parallelized computations as all involved Dactors are messaged at the same time and calculate their responses concurrently.

In summary, explicit message handling in Dactors is used to implement the cascading communication pattern; sequential and concurrent Functors are the framework's concepts to enable inter-Dactor communication and computations. Returning to the web application example, we now want to add a new film to the database using the Functor concept. This involves changes to a new `Film` and the corresponding `Studio` Dactor instances. We can combine the concurrent and sequential Functors to implement this functionality, which is displayed in Fig. 3. Both sequential Functors are comprised of two subsequent steps: They retrieve information from one Dactor to update the other one. They are independent of each other, so they can be executed in parallel, which is done by using the concurrent Functor. It only sends a successful response to its caller after both sequential Functors have sent their responses to the concurrent Functor.

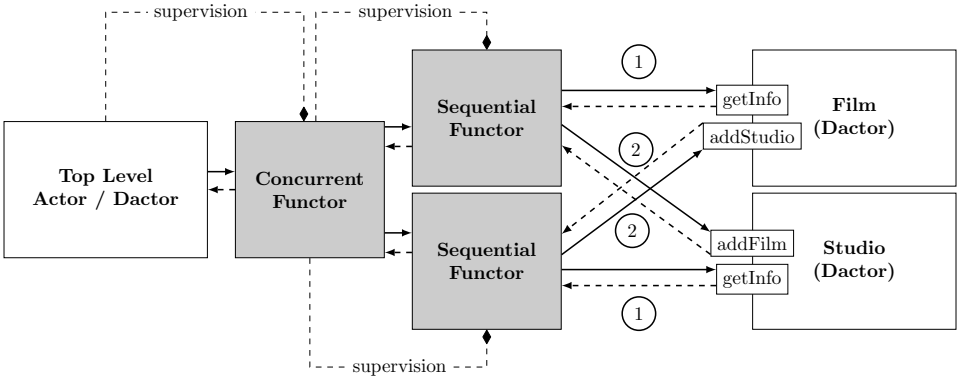


Fig. 3: Component diagram indicating the message flow through Functor objects and their supervision by the calling actor. Arrow and dashed arrow pairs indicate corresponding request and response messages. The outgoing requests of each sequential Functor are numbered to indicate their order.

3.3 System Details

Data partitioning in an actor database system differs fundamentally from common partitioning techniques used in relational databases. While large tables are typically partitioned based on a specific column's value or the hash thereof, our framework provides Dactors as entities for data encapsulation and partitioning. Dactors can be provisioned across multiple virtual runtimes and physical machines, because every Dactor instance is independent of the others and the only mean of communication is message passing. As such, they provide flexible, fine-grained data partitioning based on application needs.

The distributed nature of the database system introduces the new problem of partition or *actor discovery*. The framework maintains a unified namespace, in which each Dactor instance is identified by its Dactor type and a unique ID. In fact, querying a specific Dactor just requires obtaining the messaging address from the name-service and sending a message to it. In case of a multi-node deployment, this is complemented by Akka's Cluster Sharding component, which routes the messages to the right physical host.

Finally, *failure handling*, especially with regard to computations relying on multiple Dactors' data, requires careful monitoring due to Dactor distribution. Building on Akka's parent-child supervision concept, our framework allows for transparent failure handling configurations. Failures can be handled within Dactors if appropriate. In case of multi-Dactor queries a fail-fast approach is chosen to allow calling actors to react to exceptions in a timely manner.

4 Performance and Memory Overhead Experiments

We present a short evaluation of the framework implementation with regard to query performance and the memory overhead introduced by storing data in possibly hundreds of thousands of Dactors, each storing only a relatively small amount of data respectively. All tests were executed on a single consumer computer fitted with an Intel Core i5-7600K CPU running at 3.8 GHz and 16 GB of RAM.

We performed experiments using an exemplary actor database system, which is modeled based on a real-world scenario. It consists of four different Dactor types, each containing one to three relations. The data stored in one Dactor ranges from seven to about 700 kilobytes depending on its type. We generated four different datasets emulating the scaling of the system by increasing the number of Dactor-instances and keeping the data size stored in one Dactor nearly constant. The datasets include various primitive and complex data types and are distributed across Dactors and relations. The dataset sizes are reported in Tab. 1.

For all runtime performance experiments the median runtime of $N = 1000$ concurrently executed queries are reported. A point-query for data contained in a single instance of a Dactor object presents a response time latency of 22 ms. A concurrent insert of related data points into two Dactor objects, managed by a concurrent functor ensuring a consistency constraint, runs 111 ms. For each dataset we performed three different tests in order to evaluate the memory overhead introduced by encapsulating the system's data in a large number of Dactors and relations:

Single string Convert all data into its `String` representation and load it as a single big `String` into memory. This test serves as baseline for the other tests.

Relations Load the data into their respective `Relation`s, preserving the type information and using the in-memory data storage objects from our framework.

Framework Use the full-fledged framework to load the data into memory. This approach stores the data distributed across Dactors in `Relation` objects.

To obtain the used memory of the objects in our test approaches, we used VisualVM[2] to create heap dumps. After the data was completely loaded into memory, we triggered a garbage collection run and created a heap dump. VisualVM is able to compute the retained sizes of object hierarchies in those dumps. This allowed us to investigate the memory usage of selected objects and their members in detail. We report the results in Tab. 1.

Dataset	**Disk size**	**# Dactors**	**Heap size**			**Overhead / Dactor**
			Single string	***Relations***	***Framework***	
D_1	10 MB	829	11 MB	28 MB	29 MB	526 B
D_2	25 MB	2 578	18 MB	43 MB	44 MB	539 B
D_3	50 MB	4 373	47 MB	116 MB	119 MB	532 B
D_4	100 MB	8 618	101 MB	233 MB	237 MB	534 B

Tab. 1: Used heap size of our three different methods to load data into memory and the memory overhead of Dactors compared across the four datasets.

If the data is loaded into memory as a single big `String` object, it takes up around the same amount of heap as the dataset is big. Storing the data in *Relations* introduces a overhead of about 150%. Even for the smallest dataset the data is split up across thousands of relations, which each use a two dimensional `Array` to store the individual values. In addition to that, relations also store metadata about the contained data, such as column names and data types. Using the full framework with Dactors does not introduce much additional memory overhead. On average, using a Dactor only needs an additional 533 B more.

Let us assume that we have a 1 TB database and we chose to store 1 MB per Dactor. This requires the actor database to instantiate about one million Dactors. Using the average overhead of 550 B per Dactor, this would yield a relative memory overhead of only 0.05 %. Doing the same thought experiment with storing 10 MB per Dactor, results in about 100 000 Dactors and reduces the relative memory overhead to 0.005%.

5 Conclusion

In our research, we study the question how database features can be incorporated into the actor programming model. This work presents a proof-of-concept implementation of an actor database framework, which enables developers to declaratively define a data model using Dactors. Dactors model application-domain objects by encapsulating both the object's state and its application logic. The framework provides a shared, distributed runtime for database functionality and application logic, mitigating the *object-relational impedance mismatch* between data and business logic tier. The introduced Functor concept, which are temporary actors that manage multi-Dactor queries, provides a transparent computation model and failure handling capabilities. First experiments with our Akka-based actor database system

[2] `https://visualvm.github.io/`

show that the memory overhead introduced by using actors for data management is low. Hence, the approach is feasible and pays off especially if large amounts of data need to be stored for highly concurrent data manipulation workloads. As future work, we aim to develop inter-Dactor consistency guarantees by extending Functors with a rollback and, e.g., a *two-phase commit* protocol implementation.

References

[Ar07] Armstrong, J.: A History of Erlang. In: Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages. Pp. 6-1–6-26, 2007.

[Be14] Bernstein, P. A.; Bykov, S.; Geller, A.; Kliot, G.; Thelin, J.: Orleans: Distributed Virtual Actors for Programmability and Scalability, tech. rep., Microsoft Research, 2014.

[Be17] Bernstein, P. A.; Dashti, M.; Kiefer, T.; Maier, D.: Indexing in an Actor-Oriented Database. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR). 2017.

[Bi18] Biokoda: ActorDB – 1. About, 2018, URL: http://www.actordb.com/docs-about.html, visited on: 08/16/2018.

[Ca17] Cardin, R.: Actorbase, or "the Persistence Chaos", 2017, URL: https://dzone.com/articles/actorbase-or-quotthe-persistence-chaosquot, visited on: 08/18/2018.

[EB16] Eldeeb, T.; Bernstein, P.: Transactions for Distributed Actors in the Cloud, tech. rep., Microsoft Research, 2016.

[Er18] Erlang Solutions Ltd: Case Studies & Insights, 2018, URL: https://www.erlang-solutions.com/resources/case-studies.html, visited on: 08/17/2018.

[Li18a] Lightbend, Inc.: Akka, 2018, URL: https://akka.io/, visited on: 08/15/2018.

[Li18b] Lightbend, Inc.: Lightbend Case Studies, 2018, URL: https://www.lightbend.com/case-studies#filter:akka, visited on: 08/17/2018.

[NE18] .NET Foundation: Who Is Using Orleans?, 2018, URL: https://dotnet.github.io/orleans/Community/Who-Is-Using-Orleans.html, visited on: 08/17/2018.

[SS17a] Shah, V.; Salles, M. V.: Actor Database Systems: A Manifesto. CoRR abs/1707.06507/, 2017.

[SS17b] Shah, V.; Salles, M. V.: Reactors: A Case for Predictable, Virtualized Actor Database Systems. In: Proceedings of the International Conference on Management of Data (COMAD). Pp. 259–274, 2017.

[Ve15] Vernon, V.: Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka. Pearson Education, 2015.

PgCuckoo — Injecting Physical Plans into PostgreSQL

Denis Hirn[1]

Abstract: Plan forcing is the most capable plan hinting that a database system can implement. It allows the specification of almost every aspect of an execution plan. The open source database system PostgreSQL does not implement any plan hints by default. We will show how an extension can be used to provide plan forcing for PostgreSQL. This extension allows to use the query executor directly and independently of SQL. We sketch some of the interesting use case scenarios for plan forcing in PostgreSQL.

Keywords: PostgreSQL; Physical Algebra; Plan Tree Execution; Query Unnesting

1 Plan Forcing

High-level *declarative* query languages are used to access data stored in a RDBMS. The declarative nature of SQL requires a RDBMS to derive an efficient query evaluation strategy on its own, largely without user guidance. Two key components of the evaluation mechanism of a SQL DBMS are the *query optimizer* and the *query execution engine*. The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query as input and generates an *efficient* execution plan for the given SQL query from the space of possible execution plans. This is a nontrivial task because there can be a large number of possible operator trees for a given SQL query [Ch98].

PostgreSQL uses a disk-aware cost model which combines CPU and I/O costs with certain weights. The cost of a query plan equals the summarized costs of all operators [Le15, 5.1 The PostgreSQL Cost Model]. Using cardinality estimates as its principal input, the query optimizer relies on the cost model to choose the cheapest option from semantically equivalent plan alternatives. Theoretically, as long as the cardinality estimates and the cost models are accurate, this architecture obtains the optimal query plan. In reality, cardinality estimates are usually computed based on simplifying assumptions like uniformity and independence. For real-world data sets, these assumptions are frequently wrong, which may lead to sub-optimal and sometimes disastrous plans [Le15, 1. Introduction].

Plan forcing allows to design and compile physical plans *directly* — there is no SQL or SQL compilation involved. PostgreSQL is well suited for such a far-reaching modification, because it is open source and widely known for its extensibility. Several interesting applications result from the complete control over execution plans provided by plan forcing.

[1] University of Tübingen, Department of Computer Science, denis.hirn@uni-tuebingen.de

doi:10.18420/btw2019-ws-24

- The *planner* is one of the most complex modules of PostgreSQL. This complexity can make it hard to determine whether a new feature or a change to the optimization process may improve query execution performance. Instead of investing time in changing the optimization process, plan forcing can be used to explore portions of the query plan space that the *planner* is unable to enter on its own. Planner functionality that is not available can be simulated this way. In Section 2.1 we will discuss how the textual representation of a plan can be used to alter or create a completely new execution plan. This way, we can manually apply a particular optimization to multiple sample plans to assess the impact on execution performance. Usually it is very hard to construct a plan from scratch, but this process can be simplified significantly, as we will explain in Section 3.2.
- There are plenty of applications that use PostgreSQL's stable execution engine as backend, e.g. for foreign frontend languages. Although some of these tools generate logical or physical algebra internally, intermediate SQL code generation is required because they cannot feed their plans directly into PostgreSQL. Plan forcing can be used to omit this intermediate SQL code and use physical plans instead.
- PostgreSQL's query optimizer is conservative and does not support (advanced) algebraic rewriting of plans. Plan forcing allows to *externalize* system-internal query processing steps. An application of algebraic rewriting is unnesting of *correlated subqueries* as will be explained in Section 3.1. Query unnesting can improve the execution performance significantly but PostgreSQL's planner cannot de-correlate queries in general. Instead of integrating this optimization into PostgreSQL, we can utilize external tools to perform algebraic rewriting. Plan forcing is then used to execute the rewritten plan.
- We can use plan forcing in combination with PostgreSQL's planner by using plan fragments as a part of regular queries. Figure 4 shows how we can incorporate multiple occurrences of the table-valued function `plan_execute` to "stitch together" a plan. But if we use plan forcing for an entire plan we omit PostgreSQL's planner completely, as shown in Figure 3.

2 Physical Plan Injection in PostgreSQL

The logical basis of a RDBMS is usually an extended version of the *relational algebra*, as proposed by Edgar F. Codd [Co70]. An implementation of relational algebra operations is called *physical algebra*. In general, there is no bijective mapping between a SQL query and physical operators because non-trivial SQL queries can be evaluated by numerous semantically identical plans, i.e. constellations of physical operators.

PostgreSQL performs several steps to transform a SQL query from the textual representation into a *plan tree*. The first step involves to *parse* the query, which generates the *parse tree*. Following this, the *analyzer* performs a semantic analysis of the parse tree and creates a

new representation called *query tree*. The *planner* uses the query tree to generate the most efficient plan tree. In the end, the executor evaluates the plan tree. The plan tree is based on a set of data type definitions that represent expressions over the physical algebra. This tree structure serves as the input for the executor. Each node of the tree specifies a physical operator and contains any information needed for its evaluation. These steps are illustrated in Figure 1.

The *parse* and *analyze* modules construct context information in addition to the parse- and query tree. This information is important for the type inference performed in the analyze phase and for the optimization process. The executor does not require any information besides the plan tree of a query, however. This is an important property, because it implies that we can safely skip the *parse*, *analyze* and *optimization* phase without affecting the executor.

To use the PostgreSQL executor independently of SQL, an interface that allows to import, export, and execute plan trees is required. PostgreSQL offers nothing along these lines by default. Still, an extension can be used to implement such an interface. A C language extension can access internal functions and modify the overall behavior of the system [DD03].

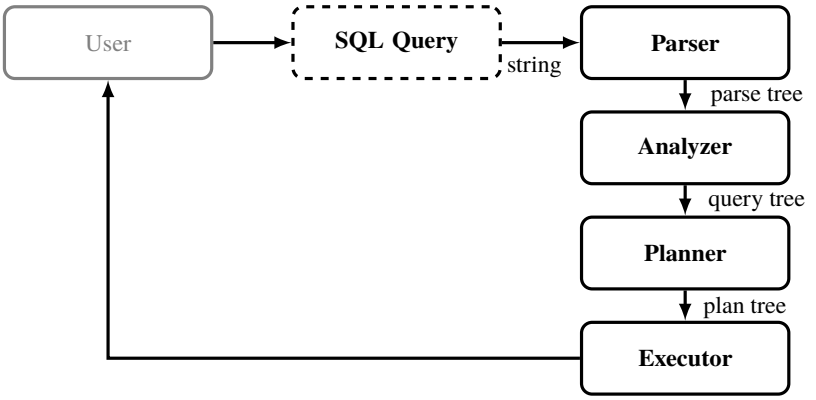


Fig. 1: PostgreSQL performs the same transformation steps for each received SQL query. Each module generates a new data structure ultimately resulting in a `plan tree`.

2.1 Plan Tree Printing and Parsing

The PostgreSQL-internal module defined in the file `outfuncs.c` can be used to *serialize* a plan tree into a *human readable* textual representation. The module defines the function `char* nodeToString(const Node* obj)` which takes a plan tree as input and generates the corresponding textual representation. Every node that can appear in a plan tree is associated with an output function.

In addition to the output function, every node must have an input function in the file `read.c`. The function `Node* stringToNode(char* str)` implements the *deserialization* of the textual representation and generates the corresponding internal node. The following applies: `stringToNode(nodeToString(node))` ≡ `node`.

However, the `stringToNode` function does not perform any *sanity checks*. This is problematic because the textual representation of a faulty plan will be parsed unnoticed as long as the syntax is correct. Normally, the `stringToNode` function is only used internally. Therefore, an error-free input is expected and no error messages are provided if the parsing fails.

The PostgreSQL executor relies on the planner to create error-free plans. There are several `Assert` statements implemented for each operator to support the development process, but they are disabled by default. Therefore, the executor is highly prone to errors in the plan tree structure.

It is possible to deserialize an arbitrary execution plan by using `stringToNode`, but it is extremely hard to develop or modify such a plan without assistance. A user would have to know and supply every field of any node in the correct ordering and format. It is very difficult to do this without making mistakes. Since we are about to assemble plans on our own, this places the burden on us to construct plans that are consistent and, in fact, executable at all. Below, we describe how to aid users in generating such correct plans.

```
{PLANNEDSTMT
    :commandType 1
    :parallelModeNeeded false
    :planTree
        {LIMIT
        :parallel_aware false
        :parallel_safe false
        :plan_node_id 0
        :targetlist [...]
        }
    :nParamExec 0
}
```

```
typedef struct PlannedStmt
{
  CmdType    commandType;
  ...
  bool   parallelModeNeeded;
  struct Plan *planTree;
  int    nParamExec;
} PlannedStmt;
```

Fig. 2: This shows a heavily truncated snippet of what the `nodeToString` function produces (left). On the right, we show how the corresponding data structure of a plan looks like.

2.2 Execution of Custom Plan Trees

PostgreSQL is designed to execute SQL queries. To our knowledge, there is no "side entrance" that allows the execution of plan trees directly. As mentioned, the stages before the executor are not required for the execution of a plan tree. PostgreSQL implements several methods to execute a query, using the SPI (Server Programming Interface) for example, but none of these functions bypass the steps prior to the executor.

Instead of writing a custom execution method from scratch, the extensive set of *hooks* provided by PostgreSQL can be used to inject a physical plan tree into the executor. Each hook consists of a global function pointer that allows to modify or replace the behavior of a

module. Recompilation of core code is not required. A hook can be enabled and disabled by setting a global variable at run time of PostgreSQL.

We use the `planner_hook` to implement the execution of arbitrary plan trees. The `planner_hook` allows to replace the `standard_planner` with another function (see Listing 1). Plan forcing can be implemented by using a planner that ignores any input and, instead, return an execution plan that we have assembled on our own. Using the `planner_hook` we can avoid to deal with low-level error- and memory management, because the same execution methods as usual can be used.

```
PlannedStmt *
planner(Query *parse, int cursorOptions, ParamListInfo boundParams)
{
     PlannedStmt *result;

     if (planner_hook)
         result = (*planner_hook) (parse, cursorOptions, boundParams);
     else
         result = standard_planner(parse, cursorOptions, boundParams);
     return result;
}
```

Listing 1: Entry point of the query optimizer [Gr18b, planner.c, 255-265]

The injection can now be done easily. After parsing an execution plan using the `stringToNode` function, SPI can be used to issue an arbitrary dummy query (e.g., `SELECT NULL`) while the `planner_hook` is enabled. PostgreSQL will parse, analyze and rewrite this query as usual. After the *analyze* module, the *query tree* is passed on to the planner. Usually, the `standard_planner` would now generate a plan tree for this query. But because the `planner_hook` is enabled, our custom planner is used instead. As explained earlier, this planner statically returns the previously parsed plan tree. This plan is evaluated by the executor accordingly. The injection is complete at this point. Figure 3 shows how the injection modifies the query pipeline.

3 Plan Forcing Applications

Now that we may lay "plan eggs into PostgreSQL's nest", a variety of interesting use cases open up. We sketch a few of these below.

3.1 Unnesting of Correlated Subqueries

Unnesting of *correlated subqueries* can greatly speed up query processing, but the majority of existing RDBMS cannot de-correlate queries in the general case. Thomas Neumann and Alfons Kemper provide a generic approach for unnesting arbitrary queries based on

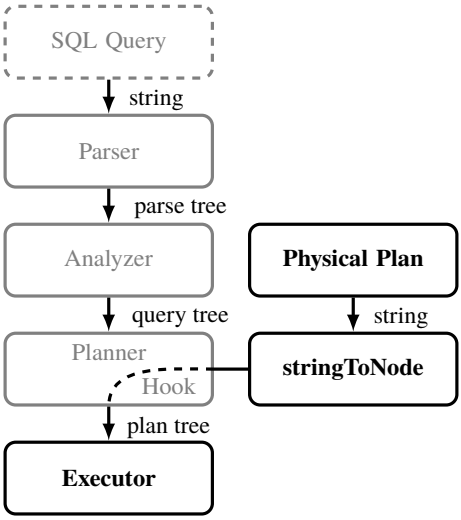


Fig. 3: Modified query pipeline that plan forcing implements. Modules shown in grey are active during normal SQL query processing but serve no function during plan forcing.

algebraic representations of a query with correlated subqueries [NK15]. As far as we know, this approach has not been implemented in any RDBMS besides HyPer [Ne11a].

The unnesting is performed using a rule system which implements algebraic rewriting of logical execution plans. The planner of the RDBMS HyPer uses their generic unnesting approach and is therefore able to unnest arbitrary queries whereas PostgreSQL in general is not. PostgreSQL's resulting performance disadvantage can be significant.

For PostgreSQL, there is no immediate way to use Neumann and Kemper's rule system. Ideally, the PostgreSQL developers would integrate this rule system directly with the optimization process of the planner. This would improve the performance of nested queries for all users. Because it is not clear if and when this planner feature will be implemented, we may instead use plan forcing to implement this. All that is left to do then, is the implementation of the rule system. This rewriting step can be implemented as a C language extension. In this case, the plan would stay inside the database system. Another option is to use the textual representation of a plan tree and rely on external tooling to implement the unnesting. Generally, such *externalization* of formerly strictly system-internal query processing steps is one of the most interesting aspects of the presented work.

Neumann and Kemper used two nested SQL queries Q_1 and Q_2 to compare the unnesting capabilities of various RDBMS, including PostgreSQL 9.1 and HyPer. They have been able to show that their rule system is able to produce unnested versions of these queries. The impact of their unnesting approach on PostgreSQL is not clear yet. However, instead of using a SQL query as input, we can now use plan forcing to execute the unnested plans of Q_1 and

Q_2 to determine the impact of unnesting for PostgreSQL. In effect, we use plan forcing to simulate a PostgreSQL kernel that is equipped with HyPer's sophisticated unnesting strategy. Among other lessons learned, this simulation can be used to decide whether far-reaching changes to PostgreSQL's current unnesting procedure are worth the effort.

Just as Neumann and Kemper have, we used $1,000$ `student` and $10,000$ `exam` tuples as test data. We then reproduced their experiment for Q_1 and Q_2 using PostgreSQL 10. The nested version of Q_1 had a run time of 3926.69 ms whereas the decorrelated formulation could be executed in just 47.77 ms. PostgreSQL 10 took $17,199.45$ ms to execute Q_2 without unnesting and $3,745.55$ ms with unnesting. The performance improvement matches those observed by Neumann and Kemper and therefore further supports the idea "that a system should be able to unnest arbitrary queries" [NK15].

3.2 Experimentation with Execution Plans

Plan forcing makes experimentation with execution plans possible because it can touch every aspect of a plan. This is interesting for research and educational purposes. Usually there is no way to locally control the behavior of the planner.

The plan tree data structure contains all information that the executor might need to evaluate a query. This makes these structures verbose and thus hard to construct from scratch. Normally, the user would not care about much of this information because the RDBMS automatically determines the required information based on the SQL query. We have to determine this information manually to construct a custom execution plan.

The complexity of the plan tree makes it extremely challenging to construct plan trees without assistance. We have developed a Haskell library that supports the creation of artificial execution plans. Based on two domain-specific-languages, we designed an *inference rule system* that uses the PostgreSQL catalog tables and the tree structure of the input to infer a multitude of gory plan details.

- PostgreSQL uses *object identifiers (OIDs)* as primary keys for various system tables. These OIDs are represented using an unsigned four-byte integer [Gr18a, 8.18 Object Identifier Types]. A plan tree consists of dozens of parameters that use OIDs to reference entries of the system tables. The inference rule system allows to use names such as "`sum`" or "`int4`" rather than the raw numeric value of the corresponding OID. This removes the necessity to memorize or lookup OIDs required for a plan. Also it adds a layer of abstraction to function and operator application, as they can be *overloaded*.
- Plan trees are a strongly typed data structure which means that every sub-expression must be annotated with a type. When provided with SQL queries, most of the required typing is achieved by the *analyze* module. The sophisticated *type system* allows to

skip explicit type annotations in SQL queries most of the time. This property is lost when an execution plan is constructed manually, however.
Using the same information from the system tables as the *analyze* module, our Haskell library implements a type system for the user input language. The type system serves multiple purposes in the inference rule system. First, it allows to infer type information almost completely. The only exception is the construction of constant values, which still requires explicit type annotations. Second, it is used to choose the correct alternative of possibly overloaded functions and operators, based on the number of arguments and their types.

These inferences ease plan construction notably, considering the fact that there are about 500 types, 800 operators, and 3000 functions in a fresh PostgreSQL 10.4 instance. Also semantical correctness can be guaranteed as it eliminates a potential source of error.

Another option we immediately get is to use plan trees as part of regular queries. Our PostgreSQL extension provides the table-valued SQL function `plan_execute` that expects an execution plan as input and returns the rows generated by this plan. Figure 4 shows how we can use this function to compose multiple plans into a single query. Queries that contain such rendered plans pass through the query pipeline as usual, but the *sub plan* encapsulated by the table-valued function is protected from any alterations by PostgreSQL.

```
SELECT a, b, a+b
FROM   plan_execute('{PLANNEDSTMT
                      :commandType 1 [...]
                      :planTree {SEQSCAN [...]}
                      :rtable ({RTE [...] :rtekind 0 :relid 90138 :relkind r [...])
                      [...]
                     }') AS tbl(a INT, b INT),
       plan_execute('{PLANNEDSTMT
                      :commandType 1 [...]
                      :planTree
                        {AGG [...]
                        :lefttree {SEQSCAN [...]}
                        :righttree <> [...]}
                      :rtable ({RTE [...] :rtekind 0 :relid 90138 :relkind r [...]})
                      [...]
                     }') AS agg(min INT)
WHERE a > min;
```

Fig. 4: Multiple occurrences of table-valued function `plan_execute` may be used to "stitch together" plan fragments as the users sees fit.

3.3 Plan Caching

We can turn a database system into its own plan cache, by storing the textual representation of execution plans in a table. This is an immediate result of this plan forcing approach. Instead of statically returning a plan tree, a different planner is required, that performs a lookup in a plan cache table. If this lookup returns a result, we can omit the `standard_planner`.

Otherwise, the caching planner has to call the `standard_planner` and add the generated plan into the cache table. Based on the mentioned Haskell library, we additionally have the option to prime the cache with execution plans that the original PostgreSQL query optimizer never would have considered on its own.

4 Related Work

Some RDBMSs such as Oracle, MariaDB and SQL Server natively support *optimizer hints* that can be used with SQL statements to alter execution plans. "Hints let you make decisions usually made by the optimizer. Hints provide a mechanism to instruct the optimizer to choose a certain query execution plan based on the specific criteria. For example, you might know that a certain index is more selective for certain queries. Based on this information, you might be able to choose a more efficient plan than the optimizer" [Or18, 16 Using Optimizer Hints].

Plan forcing is comparable to optimizer hints, but their expressive strength is noticeably weaker. While optimizer hints, for instance, allow to choose a certain join algorithm locally, they can not be used to create artificial plan trees. The planner and a SQL query is still required to generate a certain plan.

"Appropriate use of the right hint on the right query can improve query performance. The exact same hint used on another query can create more problems than it solves, radically slowing your query and leading to serve blocking and timeouts in your application [Ne11b, p. 177]. While query hints allow you to control the behavior of the optimizer, it doesn't mean your choices are necessarily better than the optimizer's choices. [. . .] Also, remember that a hint applied today may work well but, over time, as data and statistics shift, the hint may no longer work as expected [Ne11b, p. 181]."

SQL Server 2005 introduced the `USE PLAN` query hint. This allows to specify an entire execution plan as a target to be used to optimize a query. The `USE PLAN` hint can force most of the specified plan properties, including the tree structure, join order, join algorithms, aggregations, sorting, unions, and index operations like scans [Ne11b, pp. 248-250]. This approach is similar to ours, as it allows the execution of a serialized *plan tree*. However, plan forcing in SQL Server still requires a SQL query and it is required that the *plan tree* can be produced by the optimizer's normal search strategy [Pa05]. These restrictions prevent SQL Server from crashing when an invalid *plan tree* is encountered. This is not the case for PostgreSQL. Invalid *plan trees* can crash the database server which results in a restart of the service.

The `pg_plan_advsr` extension is another interesting use case of the `planner_hook`. This extension implements a feedback loop from the executor to the planner which is used to auto-tune plans. Normally, no run-time information of plans is fed back into the planner in order to self optimize. By repeatedly feeding back the information about the actual execution

of a plan, the error in estimation of row counts can be corrected. That affects the planning process and the resulting plan [Ya18].

5 Closing Thoughts

We were able to add plan forcing to PostgreSQL by using a C language extension and standard features such as the `planner_hook`. This demonstrates impressively how far the envelop of PostgreSQL extensibility can be pushed. Further investigations into the various application opportunities of plan forcing will be required. An important module we are currently working on is another Haskell library that abstracts physical into logical algebra plans. One application of this library will be algebraic rewriting. More specifically, we want to use it to implement Neumann and Kemper's unnesting rule system for PostgreSQL, as described in Section 3.2.

References

[Ch98] Chaudhuri, S.: An Overview of Query Optimization in Relational Systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS '98, ACM, Seattle, Washington, USA, pp. 34–43, 1998, ISBN: 0-89791-996-3, URL: http://doi.acm.org/10.1145/275487.275492.

[Co70] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks -. Freuburg i.B., 1970.

[DD03] Douglas, K.; Douglas, S.: PostgreSQL - A Comprehensive Guide to Building, Programming, and Administering PostgresSQL Databases. Sams Publishing, Indianapolis, Indiana, 2003, ISBN: 978-0-735-71257-7.

[Gr18a] Group, T. P. G. D.: PostgreSQL 10 Documentation, 2018, URL: http://www.postgresql.org/docs/10/static/index.html, visited on: 06/25/2018.

[Gr18b] Group, T. P. G. D.: PostgreSQL 10 Source Code, 2018, URL: https://www.postgresql.org/ftp/source/v10.0/, visited on: 06/25/2018.

[Le15] Leis, V.; Gubichev, A.; Mirchev, A.; Boncz, P.; Kemper, A.; Neumann, T.: How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9/3, pp. 204–215, Nov. 2015, ISSN: 2150-8097, URL: http://dx.doi.org/10.14778/2850583.2850594.

[Ne11a] Neumann, T.: Efficiently Compiling Efficient Query Plans for Modern Hardware. Proc. VLDB Endow. 4/9, pp. 539–550, June 2011, ISSN: 2150-8097, URL: http://dx.doi.org/10.14778/2002938.2002940.

[Ne11b] Nevarez, B.: Inside the SQL Server Query Optimizer -. Red Gate Books, 2011, ISBN: 978-1-906-43460-1.

[NK15] Neumann, T.; Kemper, A.: Unnesting Arbitrary Queries. In: BTW. Vol. 241. LNI, GI, pp. 383–402, 2015.

[Or18] Oracle: Oracle Database Online Documentation, 2018, URL: https://docs.oracle.com/cd/B19306_01/server.102/b14211/hintsref.htm#i8327, visited on: 06/25/2018.

[Pa05] Patel, B. A.: Forcing Query Plans, 2005, URL: https://technet.microsoft.com/en-us/library/cc917694.aspx, visited on: 06/25/2018.

[Ya18] Yamada, T.: Auto Plan Tuning using Feedback Loop, 2018, URL: https://www.postgresql.eu/events/pgconfeu2018/schedule/session/2132-auto-plan-tuning-using-feedback-loop/, visited on: 10/31/2018.

Policy-based Authentication and Authorization based on the Layered Privacy Language

Sebastian Wilhelm[1], Armin Gerl[2]

Abstract: In 2018 the *General Data Protection Regulation (GDPR)* has been enforced providing a new legal framework with rules and regulations for processing *personal data*. The requirement for distinguishing between purposes has been introduced, leading to the necessity of adapting existing authentication and authorization processes. We introduce a detailed authentication and authorization extension, which is able to verify requests on personal data based on the *Layered Privacy Language (LPL)*. This extension is evaluated in the form of a benchmark, utilizing the *Policy-based De-identification*, to demonstrating its efficiency and suitability for data-warehouses.

Keywords: Access Control, GDPR, Privacy, Privacy Language

1 Introduction

The *General Data Protection Regulation (GDPR)* has been enforced in Europe since May 25th 2018 constituting *Privacy by Design* and *Privacy by Default* for all technical systems [Co16, Art. 25]. Hereby, processing of personal data, which also includes storage of personal data in databases and data-warehouses, requires either a legal basis or consent [Co16, Art. 6]. Furthermore, several conditions for consent have to be fulfilled, including the necessity of the *Controller* demonstrating that the *Data Subject* consented. Consent has to be given freely, or consent has to be differentiated according to the purposes of processing [Co16, Art. 7, Recital 32]. Therefore, common authentication and authorization processes have to be adapted and extended to allow a purpose-based processing of personal data to comply with the GDPR. The Layered Privacy Language (LPL) in combination with its overarching privacy framework intends to comply with the requirements of the GDPR to enforce privacy policies 'from consent to processing' [Ge18b].

The focus of this work lies in the detailing of the extension of authentication and authorization by introducing purpose- and data-based authorization based on LPL as well as its evaluation. The evaluation considers the scalability of this extension utilizing a benchmark.

[1] Deggendorf Institute of Technology, Technology Campus Grafenau, Hauptstraße 3, D-94481 Grafenau sebastian.wilhelm@th-deg.de
[2] University of Passau, Chair of Distributed Information Systems, Innstraße 41, D-94032 Passau, armin.gerl@uni-passau.de

doi:10.18420/btw2019-ws-25

In the following, we will give an introduction of relevant elements of LPL in section 2. Section 3 details the authentication and authorization processes of LPL introducing purpose- and data-based authorization, which will be evaluated in section 4. Related work is presented in section 5. Lastly, the work is concluded and future work is outlined in section 6.

2 Layered Privacy Language (LPL)

The Layered Privacy Language (LPL) is a domain specific language, designed to model legal privacy policies [Ge18b]. In the following, we will detail only the relevant elements for the scope of this paper. We also want to point out, that we will not consider the *UI Extension* [Ge18a] or *Art. 12-14 GDPR Extension* [GP18].

The root element of LPL is the *LayeredPrivacyPolicy*-element *lpp*, which has a set of *Purpose*-elements *p*, a *DataSource*-element *ds*, as well as several attributes which are not relevant for the scope of this work. Hereby, the *DataSource*-element denotes either the *Data Subject* or another legal entity (e.g. a *Controller*) providing the data. A *Purpose*-element further describes a set of *DataRecipient*-elements *dr* as well as a set of *Data*-elements *d*, further elements and attributes are omitted for the sake of this work.

Based on LPL an overarching privacy framework is developed, which is intended to provide both a user interface to inform the user on the privacy policy as well as the enforcement of the policy. In general, the life-cycle of LPL consists of six steps. It starts with the *Creation* step, in which the responsible data protection officer creates the LPL privacy policy. This policy is then presented to the *Data Subject* during the *Negotiation* step, in which the *Data Subject* can personalize it and eventually decides to accept or give consent to the personalized privacy policy or not. If the privacy policy is accepted or consented to, then the privacy policy will be validated and information about the *Data Subject* and its personal data will be added during the *Pre-Processing* before it is stored along with the regular data during *Storage*. During the *Transfer* step, data may be transferred to third parties under a negotiated policy, to which then the original privacy policy will be added. Lastly, the stored data will be processed during the *Usage* step, which will be the main focus in this work.

Hereby, the *Policy-based De-identification* process will be applied on each request for data processing, whereas data will be de-identified as necessary, when the requesting entity is authenticated and authorized to process the data for the specified purposes. The processes for authentication and authorization are the focus of this work and will be detailed in the following in the context of the *Policy-based De-identification*.

3 Policy-based De-Identification

A request on the *Policy-based De-identification* generally takes the form of the following tuple:

$$request = (userIdentifier, credential, \widehat{P}, \widehat{D}, \widehat{DS})$$

Where *userIdentifier* is the unique identifier of the requesting user (e.g. username), *credential* the credential of the user for authentication (e.g. password), $\widehat{P}$ the set of all requested *Purposes*, $\widehat{D}$ the set of all requested *Data* and $\widehat{DS}$ the set of all requested *DataSources*.

LPL now defines four steps for this process of inspecting which *Data* the *Data Recipient* is allowed to request:

1. **Entity-Authentication**. Verify the authenticity of the requesting entity.
2. **Purpose-Authorization**. Get all *Child-Purposes* $\widehat{P_{child}}$ of the requested *Purposes* $\widehat{P_{requested}}$ and consider which *Purposes* are relevant.
3. **Entity-Authorization**. Verify if the requesting entity (*Data Recipient*) is authorized to request the *Purposes* $\widehat{P_{requested}}$ (or $\widehat{P_{child}}$).
4. **Data-Authorization**. Verify whether it is allowed to request the requested *Data* $\widehat{D}$ of the requested *Data Sources* $\widehat{DS}$ with the authorized *Purposes* $\widehat{P_{authorized}}$.

The result of the de-identification process can be described as follows:

$$resultset = (\widehat{SD}); \qquad SD = (dataSource, \widehat{DP}); \qquad DP = (data, \widehat{P_{result}})$$

Where *dataSource* is a *DataSource*-element, *data* a requested and authorized *Data*-element, and $\widehat{P_{result}}$ is the set of all relevant *Purposes* for a specific *Data*-element for a specific *Data Source*-element which are authorized. *DP* maps all relevant *Purposes* to a *Data*-element and *SD* maps this *Data-Purpose Maps* to a specific *Data Source*-element. The four steps of the *Policy-based De-identification* will be detailed in the following.

3.1 Entity-Authentication

The *Entity-Authentication* process has the task of verifying the identity of a *Data Recipient*. It will be inspected if the requesting entity is known[3]. Then the entity authentication will be conducted, based on the configured authentication method. A request on the *Entity-Authentication* has generally the form of the following tuple:

$$request = (userIdentifier, credential)$$

The module verifies if the handed *userIdentifier* and the *credential* belongs together. Some different authentication methods are possible (e.g. hashed passwords or OAuth). The result of *Entity-Authentication* is the authenticated *DataRecipient*-element dr_{auth}.

[3] Basically, an entity is stored in the data-storage (e.g. database). But there can be some special cases when entities are known but not stored in the data-storage (e.g. for some API-keys)

3.2 Purpose-Authorization

The *Purpose-Authorization* process has the task of gathering all authorized *Purposes* for a specific request. Each *Purpose p* can have *Child-Purposes* $\widehat{P_{child}}$. Contrary to the original definition in [Ge18b], it will be allowed that a *Child-Purpose* inherits from two (or more) *Parent-Purposes*. This change in definition should provide the framework with more flexibility. It is important, that no cycles in the definition of the *Purposes* exists.

A *Purpose p* is relevant for a *DataSource*-element *ds* if it (or one of its *Parent-Purposes*) is part of the privacy policy of the *Data Source*. If a *Purpose* is mentioned in at least one of the requested *DataSource*-elements *DS*, it is relevant for the current request. A request on *Purpose-Authorization* has generally the form of the following tuple:

$$request = (\widehat{P_{requested}}, \widehat{DS_{requested}})$$

Where $\widehat{P_{requested}}$ is the set of all requested *Purpose*-elements and $\widehat{DS_{requested}}$ is the set of all requested *DataSource*-elements. The module initially receives all inherited child *Purpose*-elements $\widehat{P_{child}}$ of the requested *Purpose*-elements $\widehat{P_{requested}}$ and then maps to each of the inherited child *Purpose*-elements P_{child} all *DataSource*-elements for which this specific *Purpose*-element is relevant. Finally, all elements from $\widehat{P_{child}}$, which do not have mapped any *DataSource*-elements, will be removed. The result of the *Purpose-Authorization* can be described as follows:

$$resultset = (\widehat{PD}); \qquad PD = (p, \widehat{E})$$

Where p is the *Purpose*-element and $\widehat{E}$ is the set of *DataSource*-elements for which the corresponding *Purpose*-element is relevant.

3.3 Entity-Authorization

The *Entity-Authorization* process verifies whether a *DataRecipient dr* is authorized for requesting a set of *Purposes* $\widehat{P}$. Each *Entity e* can have *Child-Entities* $\widehat{E_{child}}$. So, the *Entity e* can inherit the rights for requesting *Purposes* from its *Child-Entities* $\widehat{E_{child}}$. It is important, that no cycles in the definition of the *Entities* exists.

A *DataRecipient dr* is allowed to request a specific *Purpose p* if the *DataRecipient dr* or at least one of its children $\widehat{DR_{child}}$ is authorized for using the *Purpose p*. A request on *Entity-Authorization* has generally the form of the following tuple:

$$request = (dr, \widehat{P_{relevant}}, \widehat{P_{requested}})$$

Where *dr* is the *DataRecipient*, $\widehat{P_{relevant}}$ is the set of all relevant *Purposes* and $\widehat{P_{requested}}$[4] is the set of all requested *Purposes*. The result of *Entity-Authorization* is a set of all relevant *Purposes* for which the *Data Recipient* is authorized.

3.4 Data-Authorization

Data-Authorization verifies whether it is allowed to request the *Data* $\widehat{D}$ of the *DataSource*-elements $\widehat{DS}$ with the authorized *Purposes* $\widehat{P_{aut}}$. For each *Purpose p* it is defined which *Data* elements $\widehat{D}$ are allowed to be requested for the *Purpose p* and for each *Data Source ds Purposes* are defined to be used. The result of *Purpose-Authorization* has already mapped each requested *DataSource*-element $\widehat{DS}$ to its relevant *Purposes*. The *Data-Authorization* now has to reorder the map and get for each requested *DataSource ds* and each requested *Data* element *d* a set of possible *Purposes*. A request on *Data-Authorization* has generally the form of the following tuple:

$$request = (\widehat{D}, \widehat{DS}, \widehat{P_{aut}})$$

Where $\widehat{D}$ is the set of all requested *Data*, $\widehat{DS}$ is the set of all requested *DataSources* and $\widehat{P_{aut}}$ ist the set of all authorized *Purposes*. Basically, the module reorder the data $\widehat{D}$, $\widehat{DS}$ and $\widehat{P_{aut}}$ so that they are in the form of the resultset. Some different configurations of *Data-Authorization* are possible (e.g. return only data which are permitted for all requested *Data Source* elements). The result of the *Data-Authorization* can be described as follows:

$$resultset = (\widehat{SDP}); \qquad SDP = (ds, \widehat{DP}); \qquad DP = (d, \widehat{P})$$

Where *p* is the *Purpose*-element, *ds* the *DataSource*-element, *d* the *Data*-element and $\widehat{E}$ the set of entities for which the corresponding *Purpose*-element is relevant.

4 Benchmark-Evaluation

We implemented the steps *Entity-Authentication*, *Purpose-Authorization*, *Entity-Authorization* and *Data-Authorization* on the basis of *section 3* in *JAVA* and verified the functionality with unit tests, to measure the execution times of the individual steps depending on the input variables (e.g. number of purposes) to determine the efficiency of the *Policy-based De-identification*. It is not the goal to be able to calculate the detailed execution times of specific requests on a dataset with specific parameters rather it is important for using the *Layered Privacy Language* on data-warehouses to ensure that the concept is scaleable. In this section, execution times are analyzed, time intensive processes and method are to be found and if necessary and possible, improved or possible improvements found.

[4] $\widehat{P_{requested}}$ will be needed for a specific system configuration where will be considered if the permission on at least on of the requested *Purposes* $\widehat{P_{requested}}$ is missing.

The execution time of a request basically depends on the *system-configuration* (e.g. type of entity authentication), the amount and dependencies of the *data in the data-storage* (e.g. number of possible Data Subjects) and the amount of the requested *Data Sources*, *Data* and *Purposes*. The goal is now to measure and compare the execution times of the individual steps while varying the individual parameters. For this purpose, it is necessary to implement a *Benchmark Test Suite* which tests all possible system configurations by entering the parameters for the data store and the requested data and then saves the individual execution times. For this the *Mockito* framework is used, to mock the interface to the persistence layer. Specifically, we determine the parameters in *Tab. 1*.

PurposeAmount:	Amount of different *Purposes*, which are predetermined.
PurposeBranchning:	Degree of branching of the *Purposes* with each other.
***DataSourceNumber*:**	Amount of different entities to which *Data* are stored.
***PurposePerLpp*:**	Amount of *Purposes* for which a single entity has consented to.
***DataNumber*:**	Value determining the amount of *Data* elements.
***DataPerPurpose*:**	Value determining the amount of *Data* elements which can be requested with a single, specific *Purpose*.
***DataRecipientAmount*:**	Amount of entities which are authorized to request data.
***RecipientBranching*:**	Degree of branching of the *Data Recipients*.
***PurposePerRecipient*:**	Amount of *Purposes* for which a single *Data Recipient* is authorized.
***RequestedPurposes*:**	Amount of requested *Purposes*.
***RequestedData*:**	Amount of different requested *Data* elements.
***RequestedDataSource*:**	Amount of requested *Data Sources*.

Tab. 1: Configuration parameters for benchmark evaluation.

The *Benchmark Test Suite* first mocks the complete data store[5] of the system depending on the configuration parameters. From this mocked objects, data (*Purposes*, *Data*, *Data Sources*) are determined, again depending on the configuration parameters, which should be used for a request to the system. This request then will be executed for every possible system configuration, measuring how much time each step will take. Finally, all measurements are stored in a separate, external database. All tests were executed on a *MacBook Pro (13 inch, Mid 2012)* with the operating system *macOS High Sierra (10.13.3)*, a *2,5 GHz Intel Core i5* processor and *16 GB 1600 MHz DDR3* random access memory.

The measurements were analyzed by comparing the change of the execution times by varying one single configuration parameter at a time. *Tab. 2* shows an example of such values. With this (and further) measurements it was possible to approximate algorithms to determine a dependency on the varied configuration parameter.

It can be observed that *Entity-Authentication* is affected by *DataRecipientAmount*, *Purpose-Authorization* is affected by *PurposeAmount*, *DataSourceNumber*, *PurposePerLPP*, *RequestedPurposes* and *RequestedDataSource*, *Entity-Authorization* is affected by *PurposeAmount*, *DataSourceNumber*, *PurposePerLPP* and *DataRecipientAmount*, *Data-Authorization* is af-

[5] The persistence-layer of the system, holing the relevant user and meta data for the *Layered Privacy Language.*

RequestedDataSource	1	100	250	500	1000
Entity Authentication	0.38	0.38	0.5	0.41	0.37
Purpose Authorization	110.76	251.21	494.89	878.67	1675.47
Entity Authorization	1,59	1.84	1.82	1.89	1.88
Data Authorization	0.77	0.72	0.85	0.73	0.68
Total	113.6	254.25	498.16	881.84	1678.51

Tab. 2: Measured execution time in *ms* of the individual steps on varying the *RequestedDataSource*.

	Entity-Authentication	Purpose-Authorization	Entity-Authorization	Data-Authorization
PurposeAmount		✓	✓	
PurposeBranching				
DataSourceNumber		✓	✓	
PurposePerLpp		✓	✓	
DataNumber				✓
DataPerPurpose				✓
DataRecipientAmount	✓		✓	
RecipientBranching				
RequestedPurposes		✓		✓
RequestedData				✓
RequestedDataSource		✓		

Tab. 3: Relevance of the individual benchmark test parameters to the different steps.

fected by *DataNumber*, *DataPerPurpose*, *RequestedPurposes* and *RequestedData* (see*Tab. 3*). In summary, it could be observed that each of the benchmark parameters has at a maximum a linear effect on the execution time of the steps. This ensures scalability.

However, the parameters *DataSourceNumber*, *RequestedPurposes* and *RequestedDataSources* are noteworthy since they already have an execution time *>100ms* in some tested scenarios. Which is unacceptable for a system that is intended to work in real-time, therefore further investigation research was executed.

In the following, the parameter *RequestedDataSources* is exemplary further analyzed.The measured values of the benchmark evaluation indicate the *RequestedDataSource* parameter has a direct, linear effect on the execution time of *Purpose Authorization*.

$$t_{purposeAuthorization} = 1,5708ms * RequestedDataSources + 100.69ms$$

The value *RequestedDataSources* determines the number of *DataSource*-elements from which data should be queried. A request of *10.000* elements and even bigger numbers are normal in data-warehouse scenario (e.g. for generating a statistic). With the given formula, we already get the execution times for a *RequestedDataSource* of *10.000*.

$$t_{purposeAuthorization} = 15808.69ms$$

These execution times for the step of *PurposeAuthorization* is not acceptable for a real-time system. In order to optimize the software design, additional measurements were taken and analyzed.

Put all requested and inherited *Purposes* to the key-set	7 ms
of which executing *getInheritedPurposes*	7 ms
Put the *Data Sources* to the relevant *Purposes*	1289 ms
of which executing *getLppRelevantPurposes*	1278 ms
Getting all not relevant *Purposes* of the key-set	<1 ms
Remove all not relevant *Purposes* from the result map	<1 ms
Total execution time	**1296 ms**

Tab. 4: Detailed measured execution times of *Purpose-Authorization* for the benchmark test parameter *RequestedDataSource*.

From the *1296 ms* execution time of the step *Purpose-Authorization*, are *1286 ms* waiting for methods of other modules (*PurposeRequestManager* - the mocked persistence layer). The pure computing time required by this step is approximately *10 ms*.

Thus, we note that the high measured execution times of the *RequestedDataSource* parameter are due only to the time Mockito takes to process the *getInheritedPurposes* and *getLppRelevantPurposes* methods. Further optimization of *Purpose-Authorization* does not seem necessary to solve the original problem of long execution times for a high amount of *RequestedDataSource*. By connecting the implementation to a high-performance persistence layer, the execution time of the steps promise to be within an acceptable range. This

determination can analogously be transferred to the parameters *DataSourceNumber* and *RequestedPurposes*.

In summary, it can thus be stated that the measured execution times are very much influenced *Mockito*, and that queries can be processed even faster by a high-performance persistence layer. The aim of this *section* 4, however, was merely to show that the implementation can also be used in large systems and is scalable. This could be demonstrated.

5 Related Work

Data protection and purpose-based storage and processing of personal data is not a new field of research. Thereby, some privacy languages were already proposed, each with their own distinct application [Ge18b]. In the context of authentication and authorization amongst other the privacy languages *AIR* [Kh10], *DORIS* [BB88], *E-P3P* [As02], *P2U* [IV14], *P3P* [Cr06], *Ponder* [SLL01], *Primelife policy language* [Ar09], *Rei* [Ka02], *SecPAL* [BFG10], *SPECIAL* [Ki18], *XACL* [BCF04], are worth mentioning.

But also in the context of database systems, purpose-based authorization has been introduced in *Hippocratic Databases* [Ag02], which we will detail in the following. In contrast to the *Layered Privacy Language*, the *Hippocratic Database* design is not to be considered as fixed technology or framework [vTJ11], but as a vision of a database system that implements the principles of *Purpose Specification*, *Consent*, *Limited Collection*, *Limited Use*, *Limited Disclosure*, *Limited Retention*, *Accuracy*, *Safety*, *Openness* and *Compliance*. In the concept of *Hippocratic Databases*, analogues to the *Layered Privacy Language*, *Purposes* are the central component. For the stored data, the *Purposes* is attached, which is relevant for this record, for example we look at the schema of *Tab. 5*. In contrast to the *Layered Privacy*

purpose	customer id	name	email

Tab. 5: Database Schema for *Hippocratic Database*.

Language, the *Entity-Authentication* is not part of the *Hippocratic Database* concept. Furthermore, the *Layered Privacy Language* promises a more flexible *Data-Authorization* with different configuration parameters (e.g. return only consented data from *Data Subjects*) and so a more flexible usage of the framework.

6 Conclusion

The *Layered Privacy Language* defines six steps to privacy-preserving process data - *Entity-Authentication*, *Purpose-Authorization*, *Entity-Authorization*, *Data-Authorization*, *Minimum Anonymization* and *Application of Privacy Model*. The first four steps are detailed and evaluated in this work. The benchmark evaluation shows that the implemented modules

are scalable and capable of processing complex requests (many requested data, extensive data storage) efficient, which depends on an efficient backend.

For future works the remaining steps of the *Policy-based De-identification* process will be evaluated, showing the introduced overhead of the policy processing against the de-identification solely. Furthermore, we plan to introduce pseudonymization capabilities utilitzing tokenization methodology in both LPL as well as the *Policy-based De-identification* process. Another concern is the possibility of privacy inference that has to be detected and prevented due to consecutive requests. Lastly, different scenarii, concerning relevant privacy use cases, will be defined and used to evaluate the complete *Policy-based De-identification* process based on LPL.

References

[Ag02] Agrawal, Rakesh; Kiernan, Jerry; Srikant, Ramakrishnan; Xu, Yirong: Hippocratic Databases. In: Proceedings of the 28th International Conference on Very Large Data Bases. VLDB '02. VLDB Endowment, pp. 143–154, 2002.

[Ar09] Ardagna, Claudio A; Bussard, Laurent; De Capitani di Vimercati, Sabrina; Neven, Gregory; Pedrini, E; Paraboschi, S; Preiss, F; Samarati, P; Trabelsi, S; Verdicchio, M: Primelife policy language. In: W3C Workshop on Access Control Application Scenarios. W3C, 2009.

[As02] Ashley, Paul; Hada, Satoshi; Karjoth, Günter; Schunter, Matthias: E-P3P privacy policies and privacy authorization. In: Proceeding of the ACM workshop on Privacy in the Electronic Society - WPES '02. ACM Press, 2002.

[BB88] Biskup, Joachim; Brüggeman, Hans Hermann: The personal model of data:. Computers & Security, 7(6):575–597, dec 1988.

[BCF04] Bertino, Elisa; Carminati, Barbara; Ferrari, Elena: Access control for XML documents and data. Information Security Technical Report, 9(3):19–34, jul 2004.

[BFG10] Becker, Moritz Y.; Fournet, Cédric; Gordon, Andrew D.: SecPAL: Design and Semantics of a Decentralized Authorization Language. J. Comput. Secur., 18(4):619–665, December 2010.

[Co16] Council of the European Union: , General Data Protection Regulation, April 2016. Regulation (EU) 2016 of the European Parliament and of the Council of on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC.

[Cr06] Cranor, Lorrie; Dobbs, Brooks; Egelman, Serge; Hogben, Giles; Humphrey, Jack; Langheinrich, Marc; Marchiori, Massimo; Presler-Marshall, Martin; Reagle, Joseph M.; Schunter, Matthias; Stampley, David A.; Wenning, Rigo: , The Platform for Privacy Preferences 1.1 (P3P1.1) Specification. World Wide Web Consortium, Note NOTE-P3P11-20061113, November 2006.

[Ge18a] Gerl, Armin: Extending Layered Privacy Language to Support Privacy Icons for a Personal Privacy Policy User Interface. In: Proceedings of British HCI 2018. BCS Learning and Development Ltd., Belfast, UK, p. 5, 2018.

[Ge18b] Gerl, Armin; Bennani, Nadia; Kosch, Harald; Brunie, Lionel: LPL, Towards a GDPR-Compliant Privacy Language: Formal Definition and Usage. volume Transactions on Large-Scale Databases and Knowledge-Centered Systems (TLDKS) of Lecture Notes in Computer Science (LNCS) 10940. Springer-Verlag GmbH Germany, part of Springer Nature 2018, chapter 2, pp. 1–40, 2018.

[GP18] Gerl, Armin; Pohl, Dirk: Critical Analysis of LPL according to Articles 12 - 14 of the GDPR. In: Proceedings of International Conference on Availability, Reliability and Security. ARES 2018, Hamburg, Germany, p. 9, August 2018.

[IV14] Iyilade, Johnson; Vassileva, Julita: P2U: A Privacy Policy Specification Language for Secondary Data Sharing and Usage. In: Proceedings of the 2014 IEEE Security and Privacy Workshops. SPW '14, IEEE Computer Society, Washington, DC, USA, pp. 18–22, 2014.

[Ka02] Kagal, Lalana: Rei: A Policy Language for the Me-Centric Project. Technical report, HP Laboratories Palo Alto, 2002.

[Kh10] Khandelwal, Ankesh; Bao, Jie; Kagal, Lalana; Jacobi, Ian; Ding, Li; Hendler, James: Analyzing the AIR Language: A Semantic Web (Production) Rule Language. In (Hitzler, Pascal; Lukasiewicz, Thomas, eds): Web Reasoning and Rule Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 58–72, 2010.

[Ki18] Kirrane, Sabrina; Fernández, Javier D.; Dullaert, Wouter; Milosevic, Uros; Polleres, Axel; Bonatti, Piero A.; Wenning, Rigo; Drozd, Olha; Raschke, Philip: A Scalable Consent, Transparency and Compliance Architecture. In: Lecture Notes in Computer Science, pp. 131–136. Springer International Publishing, 2018.

[SLL01] Sloman, Morris; Lobo, Jorge; Lupu, Emil, eds. POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, London, UK, UK, 2001. Springer-Verlag.

[vTJ11] van Tilborg, Henk C. A.; Jajodia, Sushil: Optimal Extension Fields (OEFs). In: Encyclopedia of Cryptography and Security. Springer US, Boston, MA, pp. 888–890, 2011.

Tutorienprogramm

Tutorial: Data Analytics with Graph Algorithms — A Hands-on Tutorial with Neo4J

Lena Wiese[1]

Abstract: This tutorial presents perspectives for advanced graph data analytics and covers the background of graph data management in modern data stores. It provides an overview of several well-established graph algorithms. The three categories covered are path-based algorithms, community detection and centrality scores. A deeper understanding of graph algorithms is a major precondition to efficiently analyze graph-structured data. The tutorial hence enables participants to achieve an informed decision about what kind of algorithm is appropriate for which use case.

1 Introduction

Many scientists as well as practitioners work with graph-structured data. Such data often occur in many modern data science applications. Graph theory offers different variants of graph algorithms as well as many optimizations for them. Often it is however difficult to assess which algorithms will be applicable to specific use cases.

2 Outline

The tutorial will proceed in the following steps.

Background Graph data are predominant in many applications like medicine and biology, social networks, the internet and the semantic web. We present several of these applications to highlight the importance of graph data management and analytics.

Graph theory This topic explains some basics of graph theory. Having presented several choices for graph data structures (from adjacency matrix to incidence list), it describes the predominant data model for graph databases: the property graph model.

[1] University of Goettingen, Institute of Computer Science , Goldschmidtstraße 7, 37077 Göttingen, wiese@cs.uni-goettingen.de

doi:10.18420/btw2019-ws-26

The graph database The Neo4J graph database is a world-leading open source graph database. Its declarative query language *Cypher* makes graph data management convenient even for novice users. The database offers built-in visualization and implementations of all covered graph algorithms. In order to gain first practical experiences with the database, Neo4J offers an online sandbox [Sa] including the graph algorithm library.

Graph algorithms The following categories of graph algorithms will be presented:

- **Path-based** algorithms find optimal traversals of the graph. We will cover Minimum Weight Spanning Tree as well as Shortest Path.
- **Community Detection** algorithms identify clusters of nodes in the graph and assess the quality of these clusters. We will cover Label Propagation, Louvain, Weakly and Strongly Connected Components and Triangle Count.
- **Centralities** are algorithms that assign a score to each node in the graph. These scores help identify those nodes most important for certain applications. We will cover Page Rank, Betweenness Centrality as well as Closeness Centrality.

All presented algorithms will be visualized with appropriate examples in the Neo4J database.

3 Related Work

Theoretical exposition of the graph algorithms will be based on Chapter 5 and Chapter 10 of [LRU14]. Hands-on examples with the Neo4J database are based on [Co] and [NH19]. Background on graph data structures and graph databases will be based on [Wi15].

4 Speaker's Biography

Dr. Lena Wiese is head of the research group Knowledge Engineering and lecturer at the Georg August University Göttingen. She has been teaching advanced courses on data management and database technology for several years at both graduate and undergraduate level. The Neo4J database is used in these courses as a convenient test environment. She is author of the book "Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases" (DeGruyter/Oldenbourg, 2015) [Wi15]. She holds a PhD from the University of Dortmund and worked as a postdoctoral researcher at the Japanese National Institute of Informatics in Tokyo. She acts as a reviewer for international conferences and journals on a regular basis.

References

[LRU14] Leskovec, Jure; Rajaraman, Anand; Ullman, Jeffrey David: Mining of massive datasets. Cambridge university press, 2014.

[Nea] Neo4J Committers: , Neo4j Graph Algorithms User Guide. https://neo4j.com/docs/graph-algorithms/current/.

[Neb] Neo4J Sandbox: . https://neo4j.com/sandbox-v2/.

[NH19] Needham, Mark; Hodler, Amy E.: Graph Algorithms: Practical Examples in Apache Spark and Neo4j. O'Reilly, 2019.

[Wi15] Wiese, Lena: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases. DeGruyter, 2015.

StaRAI or StaRDB?

A Tutorial on Statistical Relational AI

Tanya Braun[1]

Abstract: This tutorial aims at connecting databases and statical relational AI (StaRAI), demonstrating how database systems can benefit from methods developed within StaRAI, e.g., for implementing efficient systems combining databases and StaRAI. Thus, the goal of this tutorial is two-fold: (i) Present an overview of methods within StaRAI. (ii) Provide a forum to members of both communities for exchanging ideas.

Keywords: Statistical Relational AI, Probabilistic Relational Models, Probabilistic Inference

1 Introduction

In recent years, a need for compact representations of large relational databases became apparent, e.g., in natural language understanding or decision making. Using inductive logic programming (ILP), one can build a model of a database, allowing for a crisp reproduction of data. Another idea is to build a so called factor graph model of data and introduce a probability distribution to reproduce data approximately. Such a model defines an intensional representation of a probabilistic database. A factor graph model uses parameterised variables similarly to the variables in ILP to compactly represent relations and objects. Grounding such a model incurs an exponential blowup and makes inference infeasible. Instead of grounding out a model, one can answer queries on the model directly and in a scalable way.

This tutorial aims at connecting databases and statical relational AI (StaRAI), demonstrating how database systems can benefit from methods developed within StaRAI with the goal of implementing efficient systems for probabilistic inference. Thus, the goal of this tutorial is two-fold: (i) Present an overview of methods within StaRAI. (ii) Provide a forum to members of both communities for exchanging ideas.

This tutorial provides an overview of the approaches towards probabilistic relational modelling, looking at applications, semantics, and inference problems. The main part dives into algorithms developed to scale inference in probabilistic relational models, highlighting where database systems connect in our quest for efficient implementations for large data sets. Slides are available at `https://www.ifis.uni-luebeck.de/index.php?id=597`.

[1] Universität zu Lübeck, Institut für Informationssysteme, Ratzeburger Allee 160, 23562 Lübeck, braun@ifis.uni-luebeck.de

doi:10.18420/btw2019-ws-27

2 Probabilistic Relational Modelling

Applications of probabilistic relational modelling range from information retrieval to predicting network attacks, often requiring some form of query answering (QA). As early as 1995, Fuhr presents Probabilistic Datalog for information retrieval [Fu95]. In 2017, Muñoz-González et al. use Bayesian networks for network analysis [Mu17]. While the range of application areas is wide, the areas have all in common that QA requires proper probabilistic reasoning which brings along scalability issues, especially under grounding semantics [Sa95]. In a complementary approach, one is interested to lift ground instances into a first-order or template representation and answer queries exploiting the first-order structure during calculations [Po03]. Lifted representations allow for modelling relations between objects under uncertainty.

Of course, there is not only the question of how to model a scenario but what semantics to link with a model. The above mentioned grounding semantics defines a discrete joint distribution based on factors. Various frameworks, such as the above mentioned Probabilistic Datalog or ProbLog [RKT07] use grounding semantics, including lifted approaches [Po03, RD06]. For continuous domains, probabilistic soft logic defines a density function using a log-linear model [BMG10]. Maximum-entropy semantics allow for partially specifying discrete joints, which are then completed uniformly [Th10].

The inference problems full into two categories, static or dynamic, where dynamic refers to modelling a sequential or temporal process. In the static case, the inference tasks consist of (i) projection (margins), (ii) most-probable explanation (MPE), and (iii) maximum a posteriori (MAP). In the dynamic case, the inference problems are (i) filtering (present), (ii) prediction (future), (iii) hindsight (past), and (iv) MPE/MAP (temporal sequence). The main part, regarding scalability through lifting, focuses on solving static and dynamic inference tasks within the parfactor modelling framework first introduced by Poole [Po03].

3 Scalability by Lifting

Lifting for inference has lead to the development of a range of representation formalisms such as parfactor models [Po03] and Markov logic networks [RD06], lifting propositional representations, e.g., Bayesian networks, factor graphs, or Markov networks. There exist approaches for learning lifted representations, a prominent one being the colouring algorithm by Ahmadi et al. [Ah13]. Learning lifted representations is a subject deserving of a tutorial of its own. This tutorial concentrates on efficient inference algorithms for lifted representations.

Inference algorithms take a lifted representation as input and answer queries efficiently by exploiting the first-order structures in the representations. Similar to lifting propositional models, researchers have lifted algorithms that work for propositional models to work for lifted models. From a more logical inference perspective, there exist lifted versions [VMD14] of knowledge compilation [DM02] or theorem proving [GD11], all based on

weighted model counting. Approximate lifted inference include lifted versions of belief propagation [SD08, Ah13] as well as lifted sampling approaches [GD11, FV18].

From the area of probabilistic exact inference, variable elimination (VE) is a standard algorithm for answering queries on static models [ZP94]. The junction tree algorithm (JT) builds a helper structure, called a junction tree, to efficiently answer a set of queries on a static model, incorporating VE as a subroutine [LS88]. The interface algorithm (IA) uses the junction tree idea to formalise an efficient algorithm for answering a set of queries on a dynamic model [Mu02]. For each algorithm, lifted counterparts exist, allowing for runtimes no longer depending exponentially on domain sizes, i.e., number of objects in a model. Lifted VE (LVE) avoids duplicate calculations by performing a calculation once for a representative instance and then incorporating the isomorphic instances [Po03, dSBAR05, Mi08, Ta13]. The lifted junction tree algorithm (LJT) builds a lifted junction tree representation, which enables LJT to use LVE as a subroutine [BM16]. The lifted dynamic junction tree algorithm is based on IA and LJT, combining the idea behind the interface algorithm and lifted junction trees [GBM18].

In this tutorial, we take a look at the given formalisms and algorithms through examples, investigating possible links to database systems in a quest for an efficient implementation of StaRAI algorithms in database systems.

References

[Ah13] Ahmadi, Babak; Kersting, Kristian; Mladenov, Martin; Natarajan, Sriraam: Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. Machine Learning, 92(1):91–132, 2013.

[BM16] Braun, Tanya; Möller, Ralf: Lifted Junction Tree Algorithm. In: Proc. of KI 2016: Advances in Artificial Intelligence. Springer, pp. 30–42, 2016.

[BMG10] Bröcheler, Matthias; Mihalkova, Lilyana; Getoor, Lise: Probabilistic Similarity Logic. In: UAI-10 Proc. of the 26th Conf. on Uncertainty in Artificial Intelligence. pp. 73–82, 2010.

[DM02] Darwiche, Adnan; Marquis, Pierre: A Knowledge Compilation Map. Journal of Artificial Intelligence Research, 17(1):229–264, 2002.

[dSBAR05] de Salvo Braz, Rodrigo; Amir, Eyal; Roth, Dan: Lifted First-order Probabilistic Inference. In: IJCAI-05 Proc. of the 19th International Joint Conf. on Artificial Intelligence. IJCAI Organization, pp. 1319–1325, 2005.

[Fu95] Fuhr, Norbert: Probabilistic Datalog - A Logic for Powerful Retrieval Methods. In: SIGIR-95 Proc. of the 18th Annual International ACM SIGIR Conf. on Research and Development in Information Retrieval. pp. 282–290, 1995.

[FV18] Friedman, Tal; Van den Broeck, Guy: Approximate Knowledge Compilation by Online Collapsed Importance Sampling. In: NIPS-18 Advances in Neural Information Processing Systems 31. Curran Associates, Inc., pp. 8034–8044, 2018.

[GBM18] Gehrke, Marcel; Braun, Tanya; Möller, Ralf: Lifted Dynamic Junction Tree Algorithm. In: Proc. of the International Conf. on Conceptual Structures. Springer, pp. 55–69, 2018.

[GD11] Gogate, Vibhav; Domingos, Pedro: Probabilistic Theorem Proving. In: UAI-11 Proc. of the 27th Conf. on Uncertainty in Artificial Intelligence. pp. 256–265, 2011.

[LS88] Lauritzen, Steffen L.; Spiegelhalter, David J.: Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. Journal of the Royal Statistical Society. Series B: Methodological, 50:157–224, 1988.

[Mi08] Milch, Brian; Zettelmoyer, Luke S.; Kersting, Kristian; Haimes, Michael; Kaelbling, Leslie Pack: Lifted Probabilistic Inference with Counting Formulas. In: AAAI-08 Proc. of the 23rd AAAI Conf. on Artificial Intelligence. AAAI Press, pp. 1062–1068, 2008.

[Mu02] Murphy, Kevin Patrick: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, University of California, Berkeley, 2002.

[Mu17] Muñoz-González, Luis; Sgandurra, Daniele; Barrère, Martín; Lupu, Emil C.: Exact Inference Techniques for the Analysis of Bayesian Attack Graphs. IEEE Transactions on Dependable and Secure Computing, PP(99):1–14, 2017.

[Po03] Poole, David: First-order Probabilistic Inference. In: IJCAI-03 Proc. of the 18th International Joint Conf. on Artificial Intelligence. IJCAI Organization, pp. 985–991, 2003.

[RD06] Richardson, Matthew; Domingos, Pedro: Markov Logic Networks. Machine Learning, 62(1-2):107–136, 2006.

[RKT07] Raedt, Luc De; Kimmig, Angelika; Toivonen, Hannu: ProbLog: A Probabilistic Prolog and its Application in Link Discovery. In: IJCAI-07 Proc. of 20th International Joint Conf. on Artificial Intelligence. IJCAI Organization, pp. 2062–2467, 2007.

[Sa95] Sato, Taisuke: A Statistical Learning Method for Logic Programs with Distribution Semantics. In: Proc. of the 12th International Conf. on Logic Programming. MIT Press, pp. 715–729, 1995.

[SD08] Singla, Parag; Domingos, Pedro: Lifted First-order Belief Propagation. In: AAAI-08 Proc. of the 23rd AAAI Conf. on Artificial Intelligence. AAAI Press, pp. 1094–1099, 2008.

[Ta13] Taghipour, Nima; Fierens, Daan; Davis, Jesse; Blockeel, Hendrik: Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. Journal of Artificial Intelligence Research, 47(1):393–439, 2013.

[Th10] Thimm, Matthias; Finthammer, Marc; Loh, Sebastian; Kern-Isberner, Gebriele; Beierle, Christoph: A System for Relational Probabilistic Reasoning on Maximum Entropy. In: FLAIRS-10 Proc. of the 23rd International Florida Artificial Intelligence Research Society Conf. pp. 116–121, 2010.

[VMD14] Van den Broeck, Guy; Meert, Wannes; Darwiche, Adnan: Skolemisation for Weighted First-order Model Counting. In: KR-14 Proc. of the 14th International Conf. on Principles of Knowledge Representation and Reasoning. AAAI Press, pp. 111–120, 2014.

[ZP94] Zhang, Nevin L.; Poole, David: A Simple Approach to Bayesian Network Computations. In: Proc. of the 10th Canadian Conf. on Artificial Intelligence. Springer, pp. 171–178, 1994.

NoSQL & Real-Time Data Management in Research & Practice

Wolfram Wingerath[1], Felix Gessert[2], Norbert Ritter[3]

Abstract: Users have come to expect reactivity from mobile and web applications, i.e. they assume that changes made by other users become visible immediately. However, developers are challenged with building reactive applications on top of traditional pull-oriented databases, because they are ill-equipped to push new information to the client. Systems for data stream management and processing, on the other hand, are natively push-oriented and thus facilitate reactive behavior, but they do not follow the same collection-based semantics as traditional databases: Instead of database collections, stream-oriented systems are based on a notion of potentially unbounded sequences of data items. In this tutorial, we survey and categorize the system space between pull-oriented databases and push-oriented stream management systems, using their respectively facilitated means of data retrieval as a reference point. We start with an in-depth survey of the most relevant NoSQL databases to provide a comparative classification and highlight open challenges. To this end, we analyze the approach of each system to derive its scalability, availability, consistency, data modeling, and querying characteristics. We present how each system's design is governed by a central set of trade-offs over irreconcilable system properties. We then cover recent research results in distributed data management to illustrate that some shortcomings of NoSQL systems could already be solved in practice, whereas other NoSQL data management problems pose interesting and unsolved research challenges. A particular emphasis lies on the novel system class of real-time databases which combine the push-based access paradigm of stream-oriented systems with the collection-based query semantics of traditional databases. We explore why real-time databases deserve distinction in a separate system class and dissect their different architectures to highlight issues, derive open challenges, and discuss avenues for addressing them.

Keywords: Real-Time Databases, NoSQL, Scalability, Distributed Systems, High Availability, Polyglot Persistence, Stream Processing, Cloud Data Management, Big Data, Push-Based Data Access

1 Introduction

The design of any data management system reflects a bias towards either pull-based or push-based data access: A *pull-based* query assembles data from a bounded data repository and completes by returning data once, whereas a *push-based* query processes a conceptually unbounded stream of information to generate incremental output over time. For example, traditional databases are clearly geared towards efficiency for pull-based data retrieval, even

[1] Baqend, Stresemannstraße 23, 22769 Hamburg, ww@baqend.com
[2] Baqend, Stresemannstraße 23, 22769 Hamburg, fg@baqend.com
[3] Universität Hamburg, DBIS, Vogt-Kölln-Straße 30, 22527 Hamburg, ritter@informatik.uni-hamburg.de

doi:10.18420/btw2019-ws-28

though they do support push-based access to a certain degree (e.g. through triggers). Figure 1 illustrates how the different classes of data management systems can be classified by the way they facilitate access to data.

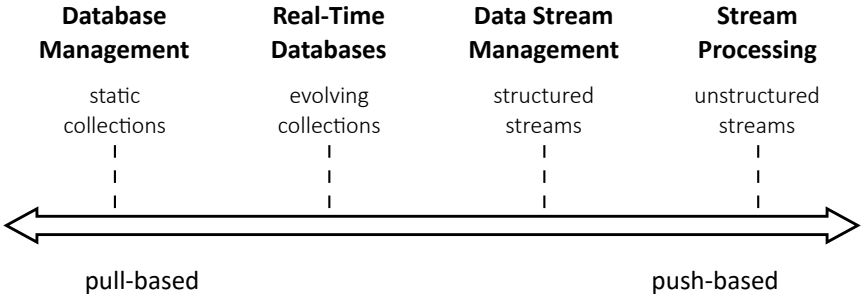


Fig. 1: Different classes of data management systems and the access patterns they support.

At the one extreme, there are **traditional databases** which represent snapshots of domain knowledge that are the basis of all queries. At the other extreme, there are general-purpose **stream processing** engines which are designed to generate output from conceptually unbounded and arbitrarily structured ephemeral data streams. Real-time databases and data stream management systems both stand in the middle, but adhere to different semantics: **Real-time databases** work on evolving collections that are distinguished from their static counterparts (i.e. from typical database collections) through continuous integration of updates over time, enabling push-based real-time queries. **Data stream management** systems provide APIs to query data streams, for example, by applying filters to incoming data or by computing rolling aggregations and joins over configurable time windows.

2 Tutorial Outline

With this tutorial, we intend to provide an overview over the entire system spectrum. Our tutorial is divided into three parts as follows.

In the first part, we cover **pull-based** data management systems. To this end, we first recall the basics of distributed data management (e.g. partitioning, replication, eventual consistency, NoSQL data models) and present the most important impossibility results (e.g. the CAP theorem). We then provide an in-depth survey of the NoSQL landscape by discussing the individual architectures and classifying each system according to functional and non-functional properties.

In the second part, we turn to **push-based** systems for real-time data management. After a short historical recap of push-based mechanisms in data management, we dissect the current state of the art in real-time databases and stream processing frameworks with respect to their capabilities in storing, querying, and analyzing data with low latency.

In the final part of our tutorial, we review polyglot persistence environments that bring together many of the discussed systems, unfold open practical and research challenges in the field of data management, and discuss possible venues for addressing them.

3 Intended Audience & Relationship to Prior Tutorials

We expect the tutorial to appeal to a large portion of the BTW community, specifically anybody interested in a comparative overview of the current data management landscape and a discussion of open challenges. Our target audience thus includes students who are looking for novel research topics and orientation, experienced researchers in the fields of database systems, cloud computing, and distributed systems, as well as industry practitioners tackling data management problems who are looking for a survey and classification of existing systems and their respective sweet spots.

This tutorial includes revised content from earlier tutorials given at EDBT 2018 [Wi18], BTW 2017 [GWR17], and ICDE 2016 [GR16].

4 Presenters

Wolfram Wingerath is a distributed systems engineer at the Backend-as-a-Service company Baqend[4] where he is responsible for all things related to real-time query processing. During his PhD studies at the University of Hamburg, Wolfram conceived the scalable design behind Baqend's real-time query engine and thereby also developed a strong background in real-time databases and related technology such as scalable stream processing, NoSQL database systems, cloud computing, and Big Data analytics. Eager to connect with others and share his experiences, Wolfram regularly speaks at developer and research conferences.

Felix Gessert is the CEO and co-founder of Baqend[4]. During his PhD studies at the University of Hamburg, he developed the core technology behind Baqend's web performance service. Felix is passionate about making the web faster by bringing research to practice. He frequently talks at conferences about exciting technology trends in data management and web performance.

Norbert Ritter is a full professor of computer science at the University of Hamburg, where he heads the databases and information systems group. He received his PhD from the University of Kaiserslautern in 1997. His research interests include distributed and federated database systems, transaction processing, caching, cloud data management, information integration, and autonomous database systems. He has been teaching NoSQL topics in various courses for several years. Seeing the many open challenges for NoSQL systems, he and Felix Gessert have been organizing the annual Scalable Cloud Data Management Workshop[5] to promote research in this area.

References

[GR16] Gessert, Felix; Ritter, Norbert: Scalable Data Management: NoSQL Data Stores in Research and Practice. In: 32nd IEEE International Conference on Data Engineering, ICDE 2016. 2016.

[4] Baqend: `https://www.baqend.com/`.

[5] Annual Scalable Cloud Data Management Workshop: `www.scdm.cloud`.

[GWR17] Gessert, Felix; Wingerath, Wolfram; Ritter, Norbert: Scalable Data Management: An In-Depth Tutorial on NoSQL Data Stores. In: Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband, 2.-3. März 2017, Stuttgart, Germany. volume P-266 of LNI. GI, pp. 399–402, 2017.

[Wi18] Wingerath, Wolfram; Gessert, Felix; Witt, Erik; Friedrich, Steffen; Ritter, Norbert: Real-Time Data Management for Big Data. In: Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018. OpenProceedings.org, 2018.

Vorstellung
DFG-Schwerpunktprogramm 2037

DFG Priority Program SPP 2037: Scalable Data Management for Future Hardware

Kai-Uwe Sattler[1] Alfons Kemper[2] Thomas Neumann[2] Jens Teubner[3]

Abstract: The priority program 2037 "Scalable Data Management for Future Hardware" is funded by the DFG and comprises 10 projects from german universities. The program is based on the observation that the currently used database concepts and systems are not well prepared to support emerging application domains. At the same time current and future hardware trends provide new opportunities. In the following we give an overview of the overall goals and the projects funded within the program.

1 Program Goals and Structure

Over the past thirty years, database management systems have been established as one of the most successful software concepts. In today's business environment they constitute the centerpiece of almost all critical IT systems. The reasons for this success are manyfold. On the one hand, such systems provide abstractions hiding the details of underlying hardware or operating systems layers. This covers the existence of a memory hierarchy, memory organization, data representation and efficient data access for multiple users or application developers. On the other hand, database management systems are ACID compliant, which enables them to represent an accurate picture of a real world scenario, and ensures correctness of the managed data even in extreme cases (e.g., a high number of concurrent database operations or possible system failures). Hence, there is a wide acceptance of architectural patterns for database systems which are based on assumptions of classic hardware setups.

Today, the application of database systems has moved beyond pure transaction-oriented scenarios. Instead they are more and more utilized as data integration platforms to realize a unified access model (perhaps limited to read operations) to heterogeneous or even distributed data. In addition, database technology in a broader sense is exploited in pure analytical applications (e.g., building models for data mining algorithms such as classification, clustering, recommendation, etc.). These analyses are based on clickstream data or experimental results in the scientific environment (e.g., protein analyses in micro biology, or galaxy detection in astro-physical research projects). For such applications, the rigid transaction-oriented architecture of classical database systems often proves to be too

[1] TU Ilmenau, kus@tu-ilmenau.de
[2] TU Munich, *lastname*@in.tum.de
[3] TU Dortmund, jens.teubner@cs.tu-dortmund.de

doi:10.18420/btw2019-ws-29

rigid, inflexible and not scalable to the required extent. During the consequently emerging diversification of data management solutions some of the well established functionalities of classical database systems fell by the wayside: For instance, consistency in eventually consistent system has to be realized at the application level (e.g. using versioning). At the same time, current and future hardware trends provide new opportunities such as:

- many-core CPUs: Next-generation CPUs will provide hundreds of compute cores already in the commodity range. In order to allow high degrees of parallelism some architectures already provide hardware support for the necessary synchronization, e.g. transactional memory. Utilizing this parallelism for database processing is still an open issue.
- co-processors like GPUs and FPGAs: Special-purpose computing units such as GPUs and FPGAs allow for parallelism at much higher degrees accelerating compute-intensive tasks significantly – even for database tasks. Moreover, heterogeneous hardware designs such as coupled CPU-FPGA and CPU-GPU architectures represent a trend of close integration between classic hardware and emerging hardware which could be beneficial in data management.
- novel storage technologies like NVRAM and SSD: Modern in-memory database system solutions still rely mostly on block-based media for ensuring persistence of data. Emerging memory technologies such as non-volatile memory (NVRAM) promise byte-addressable persistence with latencies close to DRAM requiring to revisit memory and storage hierarchies in data management systems.
- high-speed networks: Both, in scale-up and scale-out scenarios efficient interconnects play a crucial role. Today, some network technologies based on Gbit Ethernet or InfiniBand already support Remote DMA, i.e., direct access to memory of a remote node. But, to utilize this technology in database systems requires new concepts.

The goals of the DFG priority program on Scalable Data Management for Future Hardware are based on the observation that data management architectures will undergo a radical shift in the next years. This is driven by the fact that on the one hand, the range of applications requiring to handle large sets of data has significantly broadened, in particular those based on analytics/machine learning, and on the other hand, new trends in hardware as well as at operating system level offer great opportunities for rethinking current system architectures. Thus, the leitmotif of the first phase of the priority program can be formulated as “Scalability beyond current limits”.

The program is coordinated by Kai-Uwe Sattler (TU Ilmenau), Jens Teubner (TU Dortmund), Alfons Kemper (TU Munich), and Thomas Neumann (TU Munich). The coordinators are supported by an advisory board with highly experienced experts both from industry and academia: Peter Boncz (CWI/VU Amsterdam), Franz Färber (SAP SE), Goetz Graefe (Google, Madison), Theo Härder (TU Kaiserslautern), and Wolfgang Lehner (TU Dresden).

2 Funded Projects

The first phase of the priority program started in summer 2017 with a kickoff meeting at the VLDB in Munich. During this phase, 10 projects in the areas **Networking and RDMA**, **FPGA & Many-core CPUs**, **Memory and Storage**, and **SGX** are funded involving 22 researchers and 18 PIs. In detail, the following projects are part of the priority program.

Scalable Data Management in the Presence of High-Speed Networks (TU Darmstadt). The goal of this project is to develop abstractions for remote direct memory access (RDMA) in distributed databases. These abstractions shall support a wide range of different workloads ranging from traditional applications (OLAP and OLTP) to more complex workloads (e.g., machine learning). The experimental evaluation will address large-scale deployments with up to 100 nodes.

Distributed, fault-tolerant in-place consensus sequence on innovative hardware as a building block for data management (ZIB Berlin). This project deals also with RDMA technology in combination with NVRAM to manage distributed shared states. The main goal of this project is to extend the Paxos protocol to support a sequence of consensus decisions in-place.

Interactive Big Data Exploration on Modern Hardware (TU Munich). This project aims to provide interactive response times for database queries even on big data. This is achieved by deeply integrating big data exploration functionality into the core of the system. In particular, latest and emerging hardware trends to scale database systems as well as techniques such as query compilation and micro adaptivity are exploited.

Query Compilation for the Heterogeneous Many Core Age (TU Berlin). The goal of this project is to allow database systems to adapt themselves automatically to heterogeneous, previously unknown processors, and in this way, avoiding manual per-processor tuning. This is achieved by introducing variations to database operators (e.g., code optimizations, data structures and parallelization strategies), which allows to generate custom implementations of database operators for each processor.

ReProVide: Query Optimisation and Near-Data Processing on Reconfigurable SoCs for Big Data Analysis (University Erlangen-Nuremberg). In this project, a novel FPGA-based System-on-Chip (SoC) architecture called ReProVide (Reconfigurable Data ProVider) for near-data processing will be designed and developed. The goal is to provide query-specific accelerator datapaths and filter functions on-demand by exploiting the fact that the hardware of an FPGA may be dynamically reconfigured.

Adaptive Data Management in Evolving Heterogeneous Hardware/Software Systems (Uni Magdeburg). This project aims to develop integration concepts for diverse operators and heterogeneous hardware devices in adaptive database systems. In particular, optimization strategies for exploiting individual device-specific features but also the inherent cross-device parallelism in multi-device systems are investigated. The complexity of the query optimization design space incurred by the parallelism is handled by a distributed optimization approach as well as a set on cross-layer optimizations strategies incorporating learning-based techniques.

MxKernel: A Bare-Metal Runtime System for Database Operations on Heterogeneous Many-Core Hardware (TU Dortmund). As part of this projecz a bare-metal runtime system called MxKernel is developed. MxKernel provides very lightweight resource management for database system. For this purpose, heterogeneity and parallelism become first-class citizens. This is achieved by an abstraction for work items called MxTask which represents a unit of work for which atomic execution is guaranteed.

High-Performance Event Processing on Modern Hardware (Uni Marburg). This project deals with low latency requirements of Complex Event Processing (CEP) and high-throughput analysis of event data. The main goals are to develop new indexing techniques for CEP and analysis of historical event analysis exploiting modern storage technologies such as SSDs. In particular, new loading strategies for multiversion indexes and storage layouts for processing queries like pattern matching are considered.

Transactional Stream Processing on Non-Volatile Memory (TU Ilmenau). The project addresses the challenges of transactional stream processing, i.e. a combination of data stream processing with transactional guarantees such as ACID, exactly-once and ordered execution, by exploiting opportunities of modern hardware technology – in particular NVRAM. For this purpose, data structures for managing operator states taking into account the specific properties of NVRAM are developed and evaluated.

Scalable Hardware-Aided Trusted Data Management (TU Braunschweig/University of Applied Sciences Harz). The goal of this project is to exploit recent hardware security technologies, in particular Intel Software Guard Extensions (SGX), for scalable data management. In particular, the project aims to deriving an architecture model for document-based DBMSs with functionality tailored to hardware encryption support (e.g., confidentiality and integrity protection) and security awareness.

3 Activities and Outlook

In addition to the ordinary program meetings and workshops, the members of the priority program have organized several scientific activities, e.g., a Dagstuhl seminar on “Database Architectures for Modern Hardware (Seminar 18251)” [BGH+18] as well as a special issue of the german database jourrnal Datenbank-Spektrum [SKH18].

The first funding period ends in summer 2020. The call for the second phase will be published in September 2019.

References

[BGH+18] Peter A. Boncz, Goetz Graefe, Bingsheng He, Kai-Uwe Sattler: Database Architectures for Modern Hardware (Dagstuhl Seminar 18251). Dagstuhl Reports 8(6): 63-76 (2018).

[SKH18] Kai-Uwe Sattler, Alfons Kemper, Theo Härder: Editorial. Datenbank-Spektrum 18(3): Special Issue on Data Management on New Hardware (2018).

Data Science Challenge 2019

Vorwort

Die Data Science Challenge auf der BTW 2019 in Rostock

Hannes Grunert[1], Holger Meyer[2]

Abstract: Zum zweiten Mal — nach der BTW 2017 in Stuttgart [Wa17] — findet auf der BTW-Konferenzreihe die Data Science Challenge statt. Die Teilnehmer der Challenge hatten die Möglichkeit, ihren eigenen Ansatz zur cloud-basierten Datenanalyse zu entwickeln und damit im direkten Vergleich gegen andere Teilnehmer anzutreten.

Keywords: Data Science Challenge; Big Data Analytics; Feinstaub

1 Preise und Bewertungskriterien

Nach der Bewerbungsrunde präsentieren die Teilnehmer in Rostock auf der BTW 2019 ihre Ergebnisse vor einer Fachjury, bestehend aus Vertretern von Forschung und Industrie. Der erste bis dritte Platz wird mit einem Preisgeld gewürdigt. Da das Votum der Jury zum Zeitpunkt der Drucklegung noch nicht feststand, finden Sie in diesem Workshopband die Beiträge aus der Bewerbungsphase. Die ersten drei Plätze werden mit einem Preisgeld gewürdigt:

- **Erster Platz:** 500 Euro
- **Zweiter Platz:** 300 Euro
- **Dritter Platz:** 200 Euro

Die Bewertung und Auswahl der Gewinner umfasst die folgenden Kriterien:

- Neuheit und Umsetzbarkeit der Ergebnisse,
- Vollständigkeit / Umfang der Ergebnisse,
- Gesellschaftliche Relevanz,
- Datenvisualisierung und

[1] Universität Rostock, Lehrstuhl für Datenbank- und Informationssysteme, Albert-Einstein-Straße 22, 18059 Rostock, hg@informatik.uni-rostock.de
[2] Universität Rostock, Lehrstuhl für Datenbank- und Informationssysteme, Albert-Einstein-Straße 22, 18059 Rostock, hme@informatik.uni-rostock.de

doi:10.18420/btw2019-ws-30

- Live-Präsentation am 05.03.2019 auf der BTW 2019

Bewertet wurden die Beiträge durch das Preiskomitee:
Leitung: Holger Meyer, Universität Rostock

- Stefan Goers, TÜV Nord (Umweltservices)
- Daniela Nicklas, Universität Bamberg
- Kai-Uwe Sattler, TU Ilmenau
- Holger Schwarz, Universität Stuttgart
- Tim Waizenegger, IBM Böblingen
- Rajko Zschiegner, OKLab Stuttgart

2 Vorgehen

Zeitgleich mit der Ausschreibung wurden Beispieldatenquellen sowie dazu passende Beispielaufgaben bekanntgegeben. Diese Datenquellen und Aufgaben wurden von den Teilnehmern genutzt, um darauf aufbauend Ihr initiales Konzept zu entwickeln und einen ersten Prototypen zu entwickeln. Anschließend bewarben sich die Teilnehmer mit einer zweiseitigen Beschreibung.

Nach Bekanntgabe der zugelassenen Teilnehmer wurden einen Monat vor der BTW 2019 die für die Challenge zu verwendeten Datenquellen und Aufgaben bekanntgegeben. Diese sind so gewählt, dass die wesentlichen Aspekte der entwickelten Konzepte weiter verwendet werden konnten; jedoch mussten innerhalb eines Monates die Konzepte an die veränderte Aufgabenstellung angepasst werden.

Die Teilnehmer hatten eine freie Auswahl in Bezug auf die verwendeten Cloud-Dienste und -Technologien. Die zu untersuchenden Daten wurden auch in der IBM Cloud über ein IBM Watson Studio Notebook bereitgestellt.

3 Problemstellung

Sowohl die Bewerbungsaufgabe als auch die Aufgabe für das Finale drehten sich rund um das Thema Feinstaub. Als Ausgangsbasis diente der archivierte Datenbestand von `archive.luftdaten.info`. Das Citizen Science Projekt `luftdaten.info` hat zum Ziel, die Feinstaubmessung im großflächig zu realisieren. Ausgehend von dem Datenbestand wurden den potentiellen Teilnehmern eine Reihe von Beispielaufgaben gestellt:

- Bereinigen Sie die Daten bzw. stellen Sie Datenqualität sicher.

- Finden Sie interessante Sachverhalte und Muster. Visualisieren Sie Ihre Ergebnisse.
- Versuchen Sie vorherzusagen, wie die Feinstaubbelastung in der nahen Zukunft aussehen könnte.
- Entdecken Sie mögliche größere, zusammenhängende "No goAreale, an denen die Feinstaubbelastung zu groß ist bzw. zu groß sein wird.

Zur Sicherstellung der Diversität der Ergebnisse mussten die Teilnehmer die Feinstaubdaten Ihrer Universitätsstadt analysieren. Die Teilnehmer verwendeten verschiedenste Technologien, wie beispielsweise IBM Cloud SQL Queries in Kombination mit Jupyter Notebooks, Apache Spark und vorgefertigten Deep-Learning-Bibliotheken. Die Analysen reichen von der Datenbereinigung und -integration über Anwendung neu entwickelter statistischer Funktionen, neuronaler Netze und Zeitreihenanalysen bis hin zu Visualisierungen.

4 Data Science Challenge @ BTW 2019

Dem Beitragsaufruf folgten fünf Teams mit folgenden Beiträgen:

- Dresden: *Assessing the Impact of Driving Bans with Data Analysis*
- Leipzig: *Deep Learning zur Vorhersage von Feinstaubbelastung*
- Berlin: *Explanation of Air Pollution Using External Data Sources*
- Ilmenau: *Peaks and the Influence of Wheather, Traffic, and Events on Particulate Pollution*
- Stuttgart: *Prediction of air pollution with machine learning*

Nach Prüfung der Beiträge wurden alle Teilnehmer zur Präsentation Ihrer Lösung der Finalaufgabe eingeladen. Auf Basis der aktuellen politischen und gesellschaftlichen Diskussion hinsichtlich des Für und Wider von Feinstaubmessungen wurden die Teilnehmer mit neuen Fragestellungen konfrontiert:

- Wie hat sich das Verkehrsaufkommen in den Städten mit Fahrverboten verändert?
- Ist eine Korrelation zwischen der Lebenserwartung und der Feinstaubbelastung erkennbar? Sind die bisherigen Studien vom Helmholtz-Institut bestätigbar?
- Welche weiteren Faktoren, neben dem Verkehr, spielen eine Rolle bzgl. der Feinstaubbelastung? Finden sie öffentliche Events, die zu einer kurzzeitigen Steigerung der Feinstaubbelastung führen.
- Wie kann durch die Integration von mehreren Messdaten ein bundesweit einheitlicheres Bild der Feinstaubbelastung erstellt werden?

Im Rahmen des Workshopprogramms der BTW am 05.03.2019 wurden die einzelnen Beiträge präsentiert. Die Gewinner werden im Rahmen des Data Science Panels am Mittwoch, den 06.03.19, bekanntgegeben.

Die Gewinner werden im Anschluss an die BTW 2019 eingeladen, an einem Sonderbeitrag für das Datenbankspektrum mitzuwirken. Dort werden die Aufgabe aus der Finalrunde, sowie die prämierten Lösungsansätze im Detail vorgestellt.

5 Organsiation

- Stefan Goers, TÜV Nord (Umweltservices)
- Hannes Grunert, Universität Rostock
- Holger Meyer, Universität Rostock
- Ute Schuerfeld, IBM Böblingen
- Rajko Zschiegner, OKLab Stuttgart

Literatur

[Wa17] Waizenegger, T.: BTW 2017 Data Science Challenge (SDSC17). In (Mitschang, B.; Ritter, N.; Schwarz, H.; Klettke, M.; Thor, A.; Kopp, O.; Wieland, M., Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme"(DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband. Bd. P-266. LNI, GI, S. 405–406, 2017, ISBN: 978-3-88579-660-2, URL: https://dl.gi.de/20.500.12116/938.

Teilnehmer der Challenge

Assessing the Impact of Driving Bans with Data Analysis

Lucas Woltmann[1], Claudio Hartmann[1], Wolfgang Lehner[1]

1 Introduction

Suspended particulate matter (SPM) is a significant problem discussed in current environmental research with an impact on the every-day life of many people. Our goal for the BTW 2019 Data Science Challenge (DSC) is to leverage information from available sensor data about SPM and assess the benefits and disadvantages of driving bans. Our application builds upon data of 57 sensors in the city of Dresden and 338 sensors in the city of Stuttgart. Each sensor tracks particle concentration, temperature, and humidity. Stuttgart has a particular interesting situation because of the driving ban for outdated diesel engines on roads in the inner city introduced in January 2019. This gives us the possibility to compare the effectiveness of driving bans not only over time but also between two cities. While we only analyze two cities exemplary in this report, we see high potential of applying our tools to other cities and scenarios. We think, this universality of our approach is an important factor in knowledge transfer. The applications are not limited to SPM analyses but can be extended for example to weather and climate research.

The following sections address the tasks and sketch our approaches. Section 2 discusses data cleaning and preparation. This includes a minute-wise aggregation and a linear interpolation of the data. Then, we show visualization techniques used to identify patterns for an assessment of driving bans (Section 3). Furthermore, we predict the future development of SPM with a technique called Cross-sectional AutoRegession (CSAR) [Ha19], developed in our group (Section 4). In Section 5, we condense the results of previous analyses and define no-go areas with the highest particle concentrations and also identify the reasons for particular no-go areas. We close our work with a short conclusion in Section 6.

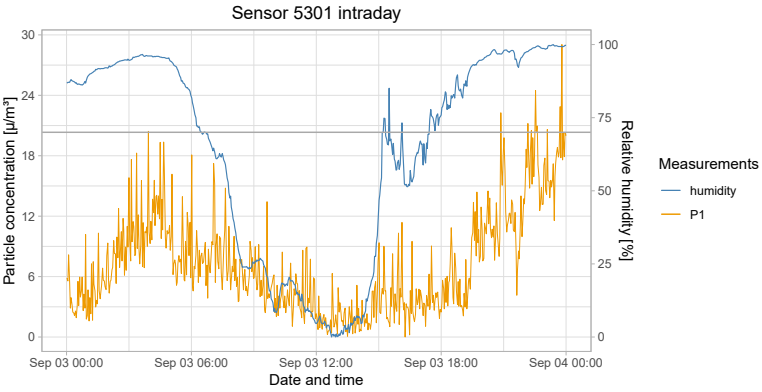
2 Data Cleaning and Preparation

First, the data of all sensors in Dresden and Stuttgart is collected. We use a nearest neighbor search around the city center of both cities with a radius of 10km to retrieve all sensors in both city areas.

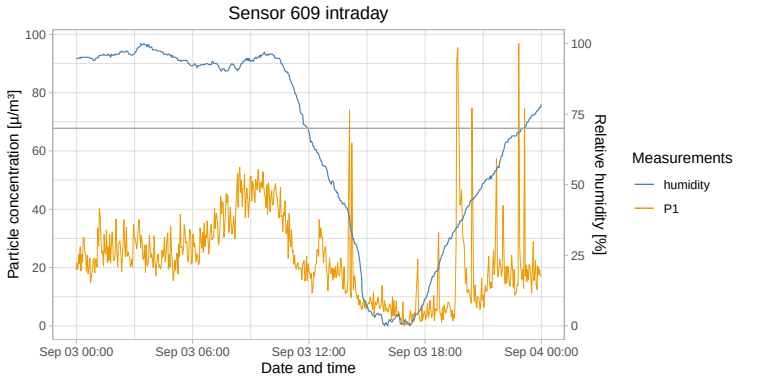
A general problem is the division of sensor types into SPM sensors and temperature/humidity sensors. To construct a common base for the analysis, we need to integrate the SPM sensors

[1] Technische Universität Dresden, Database Systems Group, 01062 Dresden, Germany
<firstname.lastname>@tu-dresden.de

doi:10.18420/btw2019-ws-31



(a) Comparison Dresden



(b) Comparison Stuttgart

Fig. 1: Particle concentration P1 compared to humidity.

and the humidity/temperature sensors into one data set. Therefore, we aggregate the data for each sensor type to a minute-wise time scale by averaging all measurement within every minute for each sensor. This provides a common minute granularity for all sensors. The next step is to merge the temperature and humidity data with the particle concentration data by sensor location and time. Note that this is only possible in a standardized minute granularity in both sensor type data.

From the resulting table, we remove all particle concentrations where the humidity is larger than 70% because above this value the particle sensors do not provide reliable reads [No15]. This leads to sparse data for both Dresden and Stuttgart because both cities have a rather humid river climate. Additionally, both cities are located in a valley. This problem is shown

in Figure 1 which compares the particle concentration P1 to the humidity for one sensor in each city close to the river. The 70% limit for the humidity is marked with a black line.

Finally, we impute all missing values with linear interpolation. This has proven to be a very robust approach in the field of time series analysis [Mo15].

3 Visualization and Patterns

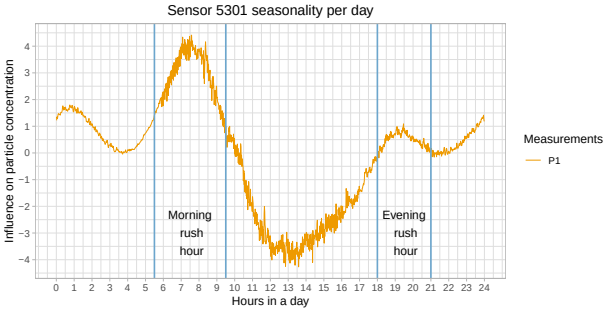
Visualization is a common approach to find interesting patterns. We present two different kinds of plots helping with the identification of structure in the data. First, all time series get split into trend, season, and residual components with time series decomposition [Cl90]. The seasonal component details peaks and valleys in the time series which occur in a regular pattern. This component shows the recurrent influence of each point in time on the time series, as shown on the y-axis.

For example, we can clearly show the morning and evening rush hours as two peaks in the seasonal component of the sensor data in Figure 2. Both sensors are located in the respective center of the given city. Whereas the sensor in Dresden in Figure 2a has clearly visible peaks for these two events, the sensor in Stuttgart details the evening rush hour only in a noisy pattern for 2018. We can now compare the regular pattern of the rush hours before and after the introduction of the driving ban in Figures 2b and 2c. The last plot has no characteristic structure for neither morning nor evening rush hour. There are no recurring rush hour patterns in the data for January 2019. This can be ascribed to the driving ban because less cars in the city center also means less traffic during rush hours or even no rush hours at all. The sheer absence of peek concentrations can be seen as an improvement in air quality. But if we have a look at Figure 3, where we took a sensor near the border of the driving ban area, one can see a negative impact of the restriction. Following the introduction of the ban, the evening rush hour has a much larger impact on the sensor's surrounding area. Figure 3b details an influence value for the evening rush hour (y-axis) 10 times higher than in Figure 3a. This can be an indication that the driving ban does in fact make the inner city cleaner but only at the expense of flooding the surrounding areas with more cars and pollution. This negative effect can be observed especially during rush hours.

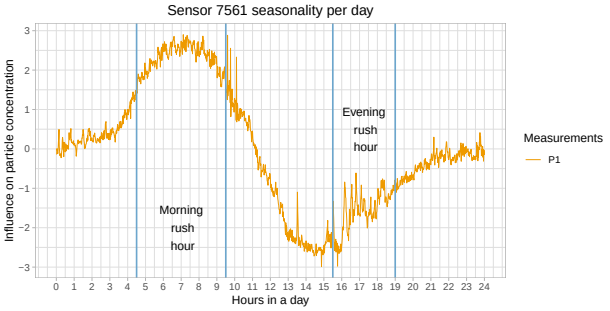
As a second visualization, we will map the distribution of particles over both cities. The plot in Figure 4 highlights particle hotspots in the cities. The darker the shade of an area, the higher the particle concentration. For visualization, the concentration is smoothed using cubic splines [Pe84]. The black points on the map are the locations of the SPM sensors. Furthermore, we will use this plot as one of the tools to identify no-go areas of high particle concentration in Section 5.

For now, we can compare the distribution of particles over Stuttgart before and after the ban. Figure 4c compared to Figure 4b gives the same intuition as our last visualization. The general pollution got less with the ban. This can be seen via the lighter hue of the complete

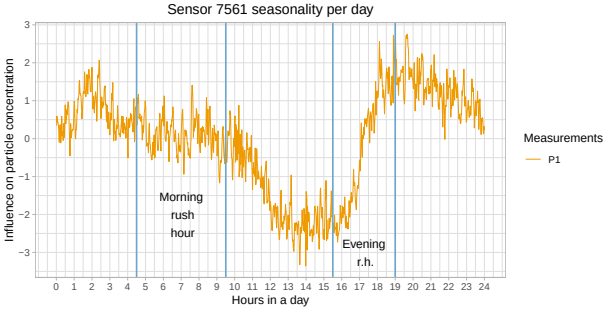
city area and the smaller areas with dark hue. Whereas the particle pollution in the inner city got better, the surrounding areas suffer from heavier pollution. The hue of the concentration coloring is darker on the edges of the city of Stuttgart. As mentioned before, the reason



(a) Rush hours in Dresden



(b) Rush hours in Stuttgart in 2018

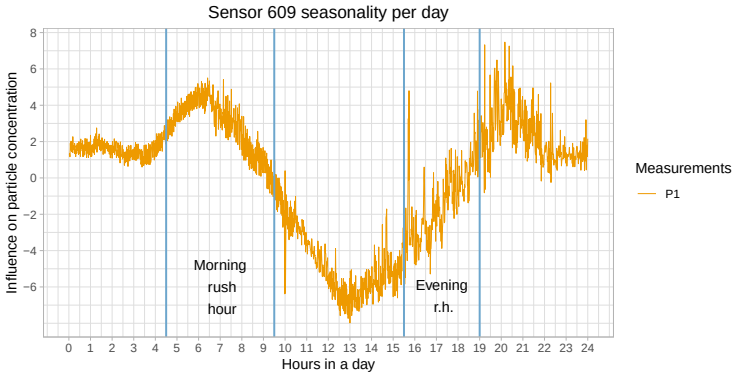


(c) Rush hours in Stuttgart in January 2019

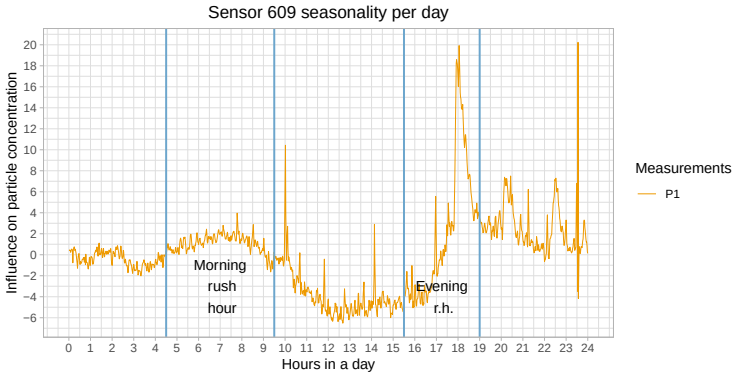
Fig. 2: Rush hour patterns for both cities.

for that might be the diverted traffic now filling the streets around the city creating more pollution there.

If we compare Dresden, a city without a driving ban, to Stuttgart, we only need to take a look at the data for 2018 because there is no active change in pollution in Dresden. Dresden has a lighter coloring and therefore less pollution because of having less traffic in the city area than Stuttgart [IN17]. The data in Stuttgart is less sparse because there are more sensors available. This leads to a finer gradient in color for the visualization and a more detailed view on pollution compared to Dresden. Nevertheless, the similar topology of both cities cause the same problems. Both cities have large areas of congestion on their access roads to the Autobahn (see Section 5). The valley position of both cities can dictate long access road to the Autobahn because building interstate roads near or in the valley might be either very expensive or very polluting. Long access roads mean more potential for heavy traffic.

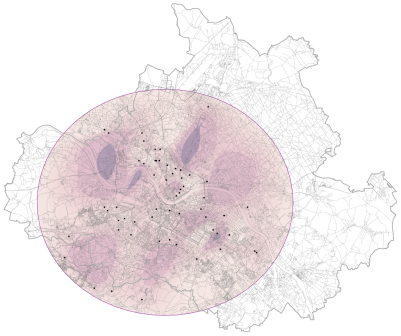


(a) Rush hours in 2018

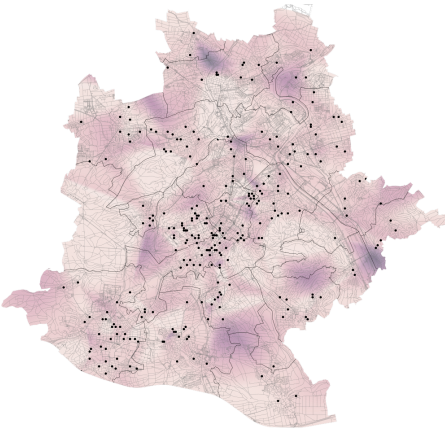


(b) Rush hours in January 2019

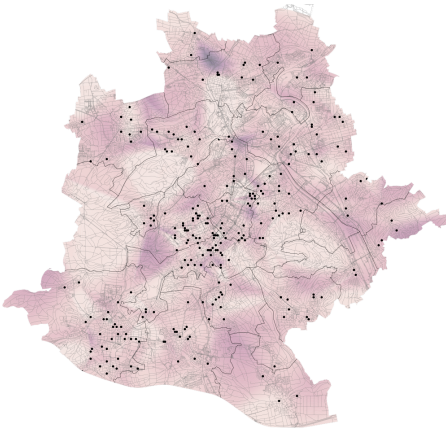
Fig. 3: Rush hour patterns for sensor outside the driving ban area.



(a) Particle distribution over Dresden in 2018.



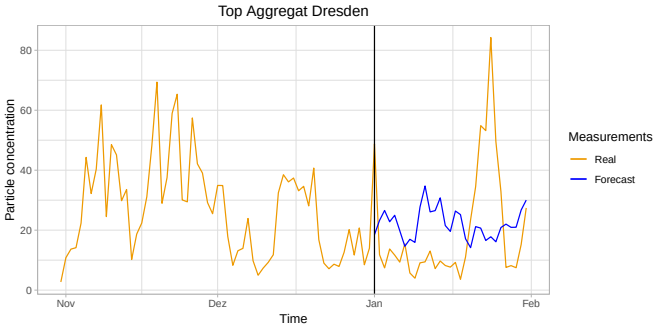
(b) Particle distribution over Stuttgart in December 2018.



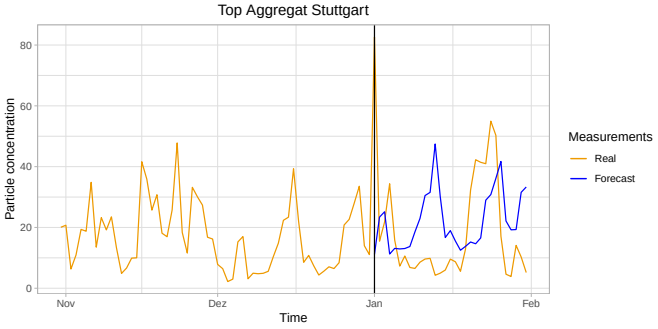
(c) Particle distribution over Stuttgart in January 2019.

Fig. 4: Particle distribution over the cities.

Even though the valley structure of both cities is hardly visible in the data visualization, one could argue that in Stuttgart one can see the southwest to northeast valley around the Nesenbach and the Neckar valley east of the city center in a darker hue. This assumption does not hold for Dresden, mainly due to the sparsity of sensors.



(a) Forecast January 2019 Dresden



(b) Forecast January 2019 Stuttgart

Fig. 5: Comparison of forecasts and actual sensor readings.

Given our analysis results, we can show that visualization is a powerful tool for assessing particle concentration and its impact on the environment. We see high potential for our approach to be applied to other cities and use cases. This could also help domain experts making decisions and drawing conclusions with a more profound background in environmental research than us.

4 Time Series Forecasting

Time series forecasting is a technique that allows computation of expected values for the future behavior of time-dependent measure values. In our previous work we designed a forecast technique called CSAR [Ha19] that focuses on the prediction of many time series that originate from the same domain, i.e. SPM sensor readings. Thorough predictions enable the comparison of predicted sensor readings without a driving ban and the actual measured values after the driving ban takes effect.

(a) No-go areas in Dresden in 2018. (b) No-go areas in Stuttgart in 2018.

Fig. 6: No-go areas in both cities.

We predict the readings for all sensors in both cities using a dedicated CSAR model for Dresden and one for Stuttgart. The results of our experiment are shown in Figure 5. For the forecast, we aggregate the sensor readings to a daily granularity. Otherwise, the prediction of the entire January would end up with a forecast horizon too long to calculate reliably. For both cities, we see that the actual SPM levels (orange line) are slightly lower than the corresponding forecast values (blue line). For Stuttgart (Figure 5b) this might be attributed to the driving ban that took effect in January. The results for Dresden (Figure 5a), show that other external factor might influence the SPM levels too, such that there is a visible deviation.

However, the comparison of only two cities is too little data to draw a reliable conclusion from. More data and an in-depth analysis are required to assess the actual role of driving bans. Furthermore, other external influences that affect the SPM levels have to be identified and taken into account during the forecasting process. This would lead to more accurate forecast results and enable more reliable decisions on the effectiveness of driving bans.

5 No-go Areas

As mentioned earlier, we will identify no-go areas with the highest particle concentrations using the generated maps. We calculate the centroids for all areas with a dark hue. The centroid gives us the possibility to analyze nearby points of interest which could generate such a high concentration. Currently, we already identified six of the top concentration accumulations in Dresden in Figure 6a and the top six in Stuttgart in Figure 6b.

In Dresden, the accumulation in the north and south are the access roads from the surrounding areas to the city center. This includes two long tunnels on the Autobahn producing a high concentration at their entrances. The northwestern area is an Autobahn exit which has a lot of traffic backup from a large shopping center located right next to the Autobahn. The one

cluster near the city center is the river harbor. Ships are well-known for not having clean engines [Sm15].

The Neckar harbor in the eastern part of Stuttgart has a similar influence. A high particle concentration can be spotted there. Near the harbor, there is an industrial area which is visible in the concentration as well. The city center has also a high particle concentration due to a lot of traffic going through. A particular interesting point is the ridge around the Birkenkopf hill southwest of the city center (marked with the mountain symbol). The curvature of the ridge forms a barrier in the particle concentration. The reason for this phenomenon could be the agglomeration of particles in front of the ridge if the wind flows perpendicular to the ridge. Last, one can see two darker areas for the access roads to the western Autobahn in the north and northwest of Stuttgart.

6 Conclusion

We have shown a multi-tool workbench for assessing the impact of SPM and driving bans in city areas. We use visualization and forecasting to find interesting patterns. This is done completely with a pure data-driven approach. Our argumentation closely follows the findings from the data. So, we rely on the correctness and completeness of the data. Data analysis only can be as good as the data fulfilling these two criteria. In our case the sparsity of the measurements due to the high humidity in some areas and the sparsity of the sensor network itself are negative factors for the analysis. The sparsity introduces uncertainty which can not be modeled correctly or can not be modeled at all. Complete data would require a organized network of sensors in our use case. Whereas the idea of open data and citizen science to collect relevant data is a good one, we see potential in getting a more organized structure into the project. Given the severity of the SPM topic, this could extend into a government-organized data collection scheme.

We think our work can be extended to different cities but also to a different set of problems. Any data which can be represented both as a concentration distribution and a time series will be assessable with our framework. Topics closely aligned would be weather and climate research, market research, and energy research. We see our approach as an important tool for knowledge transfer between people from different areas of research. The main goal would be to give domain experts a robust analysis platform for their decision making.

References

[Cl90] Cleveland, Robert B; Cleveland, William S; McRae, Jean E; Terpenning, Irma: STL: A Seasonal-Trend Decomposition. Journal of Official Statistics, 6(1):3–73, 1990.

[Ha19] Hartmann, Claudio; Ressel, Franziska; Hahmann, Martin; Habich, Dirk; Lehner, Wolfgang: CSAR: the cross-sectional autoregression model for short and long-range forecasting. International Journal of Data Science and Analytics, jan 2019.

[IN17] INRIX Global Traffic Scoreboard, http://inrix.com/scorecard/, Accessed on 08-02-2019.

[Mo15] Moritz, S.; Sardá, A.; Bartz-Beielstein, T.; Zaefferer, M.; Stork, J.: Comparison of different Methods for Univariate Time Series Imputation in R. ArXiv e-prints, October 2015.

[No15] SDS011 Laser PM2.5 Sensor specification, http://ecksteinimg.de/Datasheet/SDS011 laser PM2.5 sensor specification-V1.3.pdf, Accessed on 08-02-2019.

[Pe84] Peter Alfeld: A trivariate Clough-Tocher scheme for tetrahedral data. Computer Aided Geometric Design, 1(2):169–181, 1984.

[Sm15] Smith, T. W. P. et al.: Third IMO Greenhouse Gas Study 2014. International Maritime Organization, United Kingdom, 2015.

Explanation of Air Pollution Using External Data Sources

Mahdi Esmailoghli[1], Sergey Redyuk[2], Ricardo Martinez[3,4],
Ziawasch Abedjan[1,3], Tilmann Rabl[2,3], Volker Markl[2,3]

High concentrations of fine-grained particles in the air can adversely affect human health[5]. To control it, the European Union has undertaken several strategies, such as the introduction of certain particle concentration thresholds allowed in populated areas, or limitations for vehicle access [Ra15]. However, many cities in Germany are unable to follow this legislation[6] and control the particle emission because it is hard to attribute the pollution to a clear source. Therefore, it is important to understand the dynamic process of the fine-grained particles distribution and the reasons the emission occurs. In this project, we aim at designing a system that provides the human analyst with descriptions about polluted areas within a city, and potential causes.

During the phase of exploratory data analysis on the sensor dataset, which is provided by BTW specifically for the data science challenge[7], we selected all sensors located within a 10-kilometer radius from Berlin city center (255 sensors with 3GB of data), and provided a common data schema that fitted all the sensor types. We found that a particular subset of sensors (lat. 52.556) shows consistently higher degree of pollution. Since the original data was not enough to explain potential causes of this anomaly, we integrated the dataset with external air traffic data. Then, we established that the location of Tegel (TXL) airport airways correlated with the sensors that recorded higher pollution. We also observed seasonal fluctuations in pollution, and considered inversion during the winter as a potential cause. However, the seasonal trend near TXL turned out to be different. Air pollution is increased drastically during the summer, more likely, due to the higher number of flights to or from the airport. During the exploration phase, we discovered that air pollution might be caused by numerous local events organized in the city. For instance, we observed an instant increase in pollution ratio near the Berlin TV Tower during the New Year's Eve. Checking external web sources revealed the news feed about the New Year celebration fireworks.

[1] Technische Universität Berlin, lastname@tu-berlin.de
[2] Technische Universität Berlin, firstname.lastname@tu-berlin.de
[3] Deutsches Forschungszentrum für Künstliche Intelligenz, Berlin,
[4] ricardo_ernesto.martinez_ramirez@dfki.de
[5] https://www.pca.state.mn.us/air/fine-particles-and-human-health
[6] http://ec.europa.eu/environment/air/quality/standards.htm
[7] https://archive.luftdaten.info/

doi:10.18420/btw2019-ws-32

As demonstrated, external data can provide human analyst with comprehensible understanding of causes for the observed pollution levels. Therefore, we formulate the goal for the project as *finding explanations for air pollution through integration of external data sources, and building a new tool that provides human analysts with the explanation of potential sources of pollution*. In the project, we face several challenges: (i) streaming scenario and fast-changing data that current outlier explanation tools do not handle well [ZDM17] (e.g., MAD algorithm for univariate outlier detection is not applicable for streaming scenarios; solutions that adapt MAD for data streams, process batches instead of true streaming); (ii) heterogeneity of sensory data that leads to multiple schemata and makes data integration harder; and (iii) malfunctioning sensors that create erroneous and incomplete data [Ab16]. We address the aforementioned challenges in our project.

Progress Report and Outlook
We choose Berlin as the target region, and take two additional data sources for data integration - weather and air traffic data (airports TXL and SXF). As the data contains temporal information, we propose an event-based simulation model for our prototype that "replays" historical information as if the events are happening now, thus supporting stream processing to fit the fast-changing real-world scenario [Gr18] (Challenge (i)). In order to accommodate different schemata for external data sources, we provide a common schema that fits all external sources, and use data integration techniques [DHI12] for merging (Challenge (ii)). We utilize hexagon binning [Le11] and clustering methods to group the data spatially and integrate the readings from different neighboring sensors. Assuming that the sensors close to one another record similar data, we can fuse these data points into a single record, improving the data quality. This approach can also be used for cross-validation, in order to handle anomalies that are generated by malfunctioning sensors (Challenge (iii)). For interactive data analysis, we propose to use visualization tools, such as Thingsboard[8], and Plotly Dash[9]. To find the reason of pollution observed by aforementioned sensors, we use state-of-the-art outlier explanation systems such as Macrobase [Ba17], and integrate the correlated features with external sources, to provide reasonable interpretation of feature-wise causal relationships for interesting points [Mi13].

In the first phase of this project, we apply MAD on pollution data. We choose MAD as outlier detection algorithm because (i) pollution ratios are correlated and outlier in P1 means an outlier in P2, and vice versa; so we can use MAD which is a univariate outlier detection technique, and (ii) MAD is used in many state-of-the-art systems such as Macrobase. We introduce an online version of MAD that can treat the data as stream. Then, we acquire and prepare both weather and flight data for further integration into the prototype (fusing by the compound timestamp-location key). After data integration, we apply ranking metrics to select external data features that "explain" potential causes of anomalous pollution levels.

As fine-grained particles have many potential sources (factories, transport, cultural events, power stations, agriculture, plants' pollen, forest fires etc.), in the future we aim to generalize

[8] https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/aggregate-latest-data
[9] https://plot.ly/products/dash/

our solution and add more external sources. We also aim to provide a solution that selects external information automatically, by integrating web tables and web forms with the detected features [Ab15].

Relevant Experience. Coming from the DIMA and BIGDAMA research groups at TU Berlin, we cover the necessary expertise in data management, distributed computing [Al14], data integration [De17], and machine learning [Mo17]. Our previous applied projects included analysis of sensory data for the metal industry (production line optimization, hot rolling mills [St18]), urban development (traffic analysis), graph-based fraud detection in healthcare, and outlier explanation.

Acknowledgements. We thank Felix Neutatz, Batuhan Tüter, Felipe Gutierrez and Dimitrios Giouroukis for their constructive comments and help.

References

[Ab15] Abedjan, Z.; Morcos, J.; Gubanov, M. N.; Ilyas, I. F.; Stonebraker, M.; Papotti, P.; Ouzzani, M.: Dataxformer: Leveraging the Web for Semantic Transformations. In: CIDR. 2015.

[Ab16] Abedjan, Z.; Chu, X.; Deng, D.; Fernandez, R. C.; Ilyas, I. F.; Ouzzani, M.; Papotti, P.; Stonebraker, M.; Tang, N.: Detecting data errors: Where are we and what needs to be done? VLDB 9/12, pp. 993–1004, 2016.

[Al14] Alexandrov, A.; Bergmann, R.; Ewen, S.; Freytag, J.-C.; Hueske, F.; Heise, A.; Kao, O.; Leich, M.; Leser, U.; Markl, V., et al.: The stratosphere platform for big data analytics. VLDB 23/6, pp. 939–964, 2014.

[Ba17] Bailis, P.; Gan, E.; Madden, S.; Narayanan, D.; Rong, K.; Suri, S.: MacroBase: Prioritizing Attention in Fast Data. In: SIGMOD. Pp. 541–556, 2017.

[De17] Deng, D.; Fernandez, R. C.; Abedjan, Z.; Wang, S.; Stonebraker, M.; Elmagarmid, A. K.; Ilyas, I. F.; Ouzzani, S. M. M.; Tang, N.: The Data Civilizer System. In: CIDR. 2017.

[DHI12] Doan, A.; Halevy, A. Y.; Ives, Z. G.: Principles of Data Integration. Morgan Kaufmann, 2012, ISBN: 978-0-12-416044-6.

[Gr18] Grulich, P. M.; Saitenmacher, R.; Traub, J.; Breß, S.; Rabl, T.; Markl, V.: Scalable Detection of Concept Drifts on Data Streams with Parallel Adaptive Windowing. In: EDBT. Pp. 477–480, 2018.

[Le11] Lewin-Koh, N.: Hexagon binning. Online: `http://cran.r-project.org/web/packages/hexbin/vignettes/hexagon_binning.pdf/`, 2011.

[Mi13] Micenková, B.; Ng, R. T.; Dang, X.-H.; Assent, I.: Explaining outliers by subspace separability. In: ICDM. Pp. 518–527, 2013.

[Mo17] Monte, B. D.; Karimov, J.; Mahdiraji, A. R.; Rabl, T.; Markl, V.: PROTEUS: Scalable Online Machine Learning for Predictive Analytics and Real-Time Interactive Visualization. In: EDBT/ICDT 2017 Joint Conference. 2017.

[Ra15] Rausch, A.; Werhahn, O.; Witzel, O.; Ebert, V.; Vuelban, E. M.; Gersl, J.; Kvernmo, G.; Korsman, J.; Coleman, M.; Gardiner, T., et al.: Metrology to underpin future regulation of industrial emissions. In: 17th International Congress of Metrology. EDP Sciences, p. 07008, 2015.

[St18] Ståhl, N.; Falkman, G.; Mathiason, G.; Karlsson, A.: A Self-Organizing Ensemble of Deep Neural Networks for the Classification of Data from Complex Processes. In: IPMU. Pp. 248–259, 2018.

[ZDM17] Zhang, H.; Diao, Y.; Meliou, A.: EXstream: Explaining Anomalies in Event Stream Monitoring. In: EDBT. 2017.

Peaks and the Influence of Weather, Traffic, and Events on Particulate Pollution

Stefan Hagedorn[1] Kai-Uwe Sattler[1]

The task of the Data Science Challenge as part of the BTW 2019 conference is to analyze air quality data collected by the *luftdaten*[2] project. This project provides sensor measurements recorded from volunteers around the world. With do-it-yourself setups people can deploy their own sensors and report various environmental values to the project's servers, where they are made available as open data for further analyses. Thus, data is available only in regions where volunteers decided to participate in the project. Since in our city, Ilmenau, as well as in the state Thuringia only very few sensors are present, we decided to shift our focus to a broader area around Thuringia.

1 Frameworks, Technology & Preparation

Used Frameworks & Technology To investigate the schemata and contents of the many measurement files, we preferred a notebook system that let us easily explore the files. We use Apache Spark (and SparkSQL) in combination with our STARK[3] framework for spatial and temporal analyses, preprocessing as well as visualization. We additionally utilize Spark ML, e. g. for finding rules in the measurements. As programming languages we chose Scala, Python, and R.

Data Cleaning First analyses of downloaded data reveiled that alone for a single day, 2018-02-01, six different schemata exist. Some sensors report temperature and humidity only (`DHT22`), while others measure the particulate matter (`SDS011`). In addition to these two we found further six sensor types that measure e. g. air pressure, altitude or another level of particle concentration. We used the per-sensor measurement files provided in the *luftdaten* archive and integrated them using our Piglet[4] as well as Python scripts. In addition to the integration, a challenging task is to find invalid measurements, e. g. due to extreme weather condiditons, missing values, etc.

[1] Technische Universität Ilmenau, Databases & Information Systems, Ilmenau, first.last@tu-ilmenau.de
[2] `https://luftdaten.info/`
[3] `https://github.com/dbis-ilm/stark/`
[4] `https://github.com/dbis-ilm/piglet`

doi:10.18420/btw2019-ws-33

2 Analyses Goals

Particle Concentration Peaks Our first goal is to find peaks in the particle measurements for regions. Peaks are measurements exceeding the average particle value for a short period of time. Such peaks occur e. g. on New Year's Eve or during commuting hours. This is achieved by performing a spatial join with the sensors and a data set containing the regions of interest using STARK. Then, we calculate the average value per such region and subsequently filter the input measurements for values greater than that average. The result is visualized on a map, showing a timeline per region when and by how much the average particle concentration was exceeded. This information can be used to learn how long such extreme air pollution persists. Since this also depends on the weather conditions, we use values from weather data sets provided by the National Oceanic and Athmospheric Administration[5].

Weather Influence The weather conditions have an impact on the particle concentration measured by the sensors. Our hypothesis is that with rainy weather more people use their car instead of walking or taking the bike. However, since the rain will clean the air, we expect not many peaks. In order to support this hypothesis, we plan to additionally integrate open traffic data that can be found on various open data portals[6][7]. Besides the case for rainy weather, we also look at the opposite weather condition: Do people use the car when it gets too hot?

The overall idea is to find weather conditions when many people (or more people than usual) will use their private vehicles and thus produce more particulates. The results are temperature and precipitation values (lowest and highest) where the particle concentration rises above average.

Particulate Matter Polution Prediction & Correlation We employ a frequent itemset mining approach to find rules to predict the particle concentration and limit exceedings from the weather conditions and current volume of traffic. For this, we use the traffic data as well as "historic" weather data sets to categorize the particulate pollution. An additional correlation is to be tested between "events" (such as the begin of New Year's Eve, begin/end of school vacations, or concerts, football matches etc.) in cities and the air pollution. Since the *luftdaten* project continuously collects new values, the set of existing values is used as training data and future values are used for validation.

The result of this analysis is a set of rules that can be used to predict the range in which the particle concentration will be under forecasted weather conditions and traffic volume.

[5] https://www.ncdc.noaa.gov/cdo-web/datasets#GHCND
[6] http://opentraffic.io/
[7] http://govdata.de

Prediction of air pollution with machine learning

Christian Schmitz[1] Dhiren Devinder Serai[2] Tatiane Escobar Gava[3]

Abstract: Cities worldwide are facing air quality issues, leading to bans of vehicles and lower quality of life for inhabitants. We forecast the air quality for Stuttgart based on expected weather condition. For that purpose, we extract, cleanse, and integrate the DHT22 and SDS11 sensors' data to feed two different machine learning models for predicting the particulate matter values for the near future.

1 Introduction

According to the World Health Organization (WHO) [WH16], urban air pollution increased by more than 8% between 2008 and 2013, despite all efforts on improving air quality in many countries around the globe. Urban air pollution may lead to a number of diseases, including reduced lung function, respiratory infections, and aggravated asthma. We live in Stuttgart, the city with the highest air pollution in Germany. Thus, we are directly affected. However, we do not want to risk our health more than necessary. Therefore, we propose a solution to predict the air quality situation for the next hours and days in the city center. This information may support various applications of value to society, e.g., four route planning or adapting daily habits.

2 Approach

Our initial solution predicts air pollution by integrating two data sources. The first source is sensor data from [OK15]. In Stuttgart's downtown area, 25 sensors collect data on temperature, humidity, and particle matter. The second data source provides data on weather forecast OpenWeather [Op18], and is used as input for the machine learning models in order to predict the air quality for that given scenario (weather forecast, date, and time).

From the air pollution sensor dataset, we collect all DHT22 sensors measurements for temperature and humidity information and SDS011 sensors' measurements for air quality indices (particle matter values, to be precise). We filter the dataset, so that it only contains values from Stuttgart's downtown area. We integrate the weather information from the DHT22 sensors with the air quality indices from the SDS011 sensors based on the location and time. We notice that the dataset contains many implausible sensor values. That is why we remove those data items which were measured under unsupported weather conditions. In the end, we have an integrated dataset containing records with particle matter values and

[1] Universität Stuttgart, IPVS, Universitätsstr. 38, 70569 Stuttgart, st160269@stud.uni-stuttgart.de
[2] Universität Stuttgart, IPVS, Universitätsstr. 38, 70569 Stuttgart, st161906@stud.uni-stuttgart.de
[3] Universität Stuttgart, IPVS, Universitätsstr. 38, 70569 Stuttgart, st160427@stud.uni-stuttgart.de

doi:10.18420/btw2019-ws-34

weather information. We use this dataset to train, test, and fine-tune our machine learning models. We for instance use recurrent neural network regression techniques such as Long Short-Term Memory (LSTM).

Further, we obtain relevant weather forecast data such as the hourly humidity and temperature and feed our trained model with the forecast values, so that we finally obtain an hourly particle matter forecast for the next days. We present the final result using a visualization adapted to a chosen application, e.g., a plot to visualize the forecast for the next hours.

3 Cloud Services

Our solution makes use of multiple cloud services. For data cleaning and integration, we use Google's Cloud Dataproc service, which allows us to run Spark jobs. For machine learning, we employ Google's Cloud Machine Learning Engine. Further, we extract relevant weather forecast information from [Op18].

4 Insights

Based on our initial data cleaning, integration, and visualization, we get credible insights on relevant features for our machine learning model. In Fig. 1, we display the hourly pollution for a Sunday and Monday in spring and summer. Multiple factors affecting the air pollution are visible here, e.g., the seasonal (weather) effect, traffic during rush hour, or variation depending on day of week.

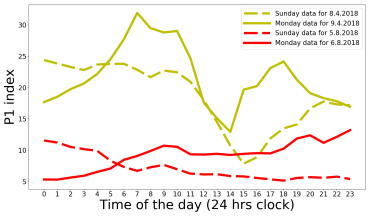


Fig. 1: Hourly air particle matter in summer and spring for a Monday and Sunday.

5 Next Steps

We currently have a cleaned and integrated dataset. We have also collected a list of important features for our machine learning approach. Nest, we will evaluate different recurrent neural network approaches such as LSTM and Gated Recurrent Unit (GRU) in order to find the most promising approach. The model for Stuttgart will then be used to implement a predictive application targeted to improve the quality of life of urban population.

Acknowledgements. We thank our advisors Ralf Diestelkämper and Melanie Herschel for their support on this project.

References

[OK15] OK Lab Stuttgart: Luftdaten Info, https://archive.luftdaten.info/csv_per_month/, Stand: 28.11.2018, 2015.

[Op18] OpenWeather: Weather API - OpenWeatherMap, https://openweathermap.org/api, Stand: 28.11.2018, 2018.

[WH16] WHO: Air pollution levels rising in many of the world's poorest cities, http://www.who.int/en/news-room/detail/12-05-2016-air-pollution-levels-rising-in-many-of-the-world-s-poorest-cities, Stand: 24.11.2018, 2016.

Deep Learning zur Vorhersage von Feinstaubbelastung

Georges Alkhouri[1], Moritz Wilke[2]

1 Einleitung

Feinstaubbelastung steht seit einiger Zeit in der öffentlichen Debatte und stellt mir hoher Wahrscheinlichkeit ein großes Gesundheitsrisiko dar. Laut WHO [Or06] kann die Reduzierung von Feinstaub zur Senkung verschiedener Krankheiten, wie bspw. Herzinfarkten, Lungenkrebs und asmathischen Erkankungen dienen. Deswegen werden von der Organisation Tagesgrenzwerte von 25 $\mu g/m^3$ für Partikel um 2,5 μm (PM2,5) und 50 $\mu g/m^3$ für Partikel um 10 μm (PM10) empfohlen. In diesem Beitrag zur Data Science Challenge soll gezeigt werden, wie die vorhandenen Feinstaubsensoren in der Stadt Leipzig genutzt werden können, um zukünftige Werte vorherzusagen.[3] Eine solche Vorhersage könnte nicht nur zur Warnung dienen, sondern auch Grundlage für kurzfristige Gegenmaßnahmen (bspw. den Wechsel auf ÖPNV) bilden.

2 Vorverarbeitung der Sensordaten

Die vorhandenen Daten in Leipzig bringen zwei Herausforderungen mit sich. Zum Einen braucht es für eine zuverlässige Verwendung der Feinstaubmesswerte aus einem Sensor vom Typ SDS011 auch ein Luftfeuchtigkeitswert zum gleichen Zeitpunkt, da die Sensoren nur zwischen einer relativen Luftfeuchtigkeit von 20% – 50% zuverlässig messen. [Bu18] Diese Messungen sind nicht für jeden Ort vorhanden. Zum Anderen benötigt es für eine Vorhersage eine Messreihe von mindesten einem Jahr, wenn (vermutete) zyklische Einflussfaktoren, wie Jahreszeit, berücksichtigt werden sollen.

Abb. 1 zeigt alle Standorte mit Sensoren vom Typ SDS011 und DHT22 (Temperatur, Luftfeuchtigkeit) in Leipzig. Da deren Verteilung keine genauere örtliche Unterteilung innerhalb der Stadt ermöglicht und die Sensoren z.T. erst seit Anfang 2018 in Betrieb

[1] Competence Center for Scalable Data Services and Solutions (ScaDS), Ritterstrasse 9-13, 04109 Leipzig, alkhouri@informatik.uni-leipzig.de
[2] Competence Center for Scalable Data Services and Solutions (ScaDS), wilke@informatik.uni-leipzig.de
[3] Der Quellcode der beschriebenen Analysen und weitere Arbeiten finden sich unter https://github.com/GeorgesAlkhouri/golddust.

doi:10.18420/btw2019-ws-35

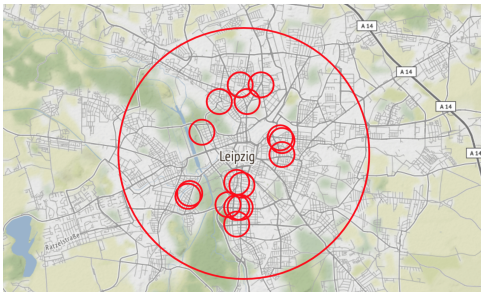


Abb. 1: Sensoren in Leipzig

sind, wurden die validen Messwerte bereinigt und die mittleren Werte für Temperatur, Luftfeuchtigkeit und Feinstaubbelastung von allen Sensoren in Leipzig zu einer Zeitreihe zusammengefasst. Hierbei wurden stündliche Intervalle aggregiert.

3 Zeitreihenanalyse und Vorhersage

Die extrahierte Zeitreihe wurde verwendet um ein neuronales Netz zu trainieren. Zur Vorhersage der Zielvariable PM10 wurden verschiedene Merkmale aus dem Datum und der angegebenen Zeit des Sensors genutzt und zudem weitere Merkmale gebildet. Prototypisch wurden die Merkmale Stunde, Tag des Monats und Monat genutzt und die Merkmale Wochentag, Wochennummer und Jahreszeit konstruiert. Des Weiteren fließen die durchschnittliche Temperatur und Luftfeuchtigkeit, in einer stündlichen Auflösung, in die Vorhersage ein.

Der verwendete neuronale Netz-Protoypen, besteht aus einer LSTM-Schicht [HS97] mit 48 Neuronen. Um einem Overfitting vorzubeugen wurde eine regulierende Dropoutschicht hinzugefügt und anschließend die Vorhersage von einem linearen Neuron durchgeführt. Hierbei konnte mit diesem simplen neuronalen Netz und den weiter oben aufgeführten Merkmalen eine mittlere quadratische Abweichung von $32,232 \mu g/m^3$ erreicht und ein erster Eindruck der Vorhersageleistung gewonnen werden, welcher in Abb. 2 dargestellt ist.

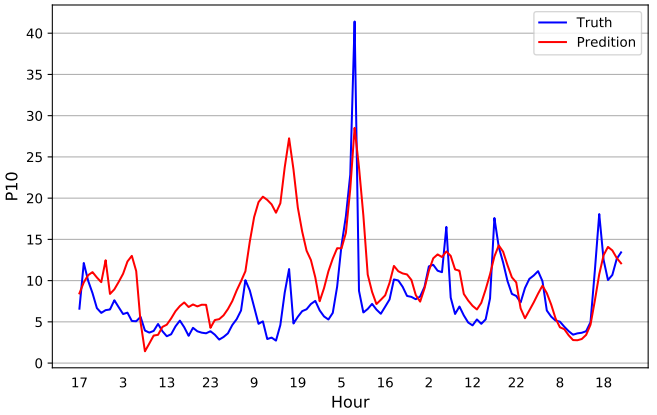


Abb. 2: Erste Validierung der Vorhersage mit einem MSE von $32.232 \mu g/m^3$ durch ein LSTM Netz.

Obwohl der illustrierte Zeitabschnitt, in Abb. 2, dem neuronalen Netz zum Trainingszeitpunkt nicht bekannt war, ist ein Vorhersagbarkeit tendenziell zu erkennen.

4 Weitere Arbeiten

Um die Aussagekraft der extrahierten Zeitreihe zu erhöhen, soll diese in ein Walk-Forward-Modell transformiert werden. Dies bedeutet, dass der Erwartungswert PM10 nach jeder Vorhersage als Merkmal für die nächste Vorhersage zur Verfügung steht. Formal soll dies folgendes Beispiel illustrieren:

$$(x_0, \ldots, x_n)_t \rightarrow PM10_t$$
$$(x_0, \ldots, x_n, PM10_t)_{t+1} \rightarrow P; 10_{t+1}$$

, wobei $x_0, \ldots, x_n$ Merkmale zur Vorhersage von $PM10$ darstellen und t den Zeitschritt definiert. Weiter gilt es, die Zeitreihe unter dem Erwartungswert PM2,5 zu untersuchen und auszuwerten.
Da exemplarisch nur ein Standort berechnet wurde, bietet sich die Möglichkeit, die Vorhersage auf mehrere Sensoren auszudehnen. Hierbei könnten dann weitere Faktoren wie Verkehrsdaten, Bevölkerungsdichte und Kategorisierung der Umgebung (Wohn-, Industrie- oder Naherholungsgebiet) in die Vorhersage einfließen. Hierbei bietet sich auch an zu prüfen, ob und wie entsprechende Daten aus öffentlichen Quellen generiert werden können. So gibt es in vielen Städten Webcams zur Betrachtung der aktuellen Verkehrslage die auswerten ließen. Abschließend ist eine Validierung der Qualität der PM10 Messwerte und Vorhersagen mittels der Daten des Bundesumweltamtes[4] notwendig.

[4] https://www.umweltbundesamt.de/daten/luft/feinstaub-belastung

5 Verwendete Technologien

Zur Verarbeitung der Rohdaten wurden IBM Cloud SQL Queries in Kombination mit Jupyter Notebooks verwendet. Die Vorhersage basiert auf der Deep-Learning Bibliothek Keras [Ch15]. Es wurde prototypisch nur ein (kombinierter) Sensor untersucht. Für die parallele Echtzeit-Verarbeitung von mehreren Sensoren bietet sich ein cloud-basiertes Streaming Framework, bspw. Apache Flink[5] oder Spark[6] an.

Literaturverzeichnis

[Bu18] Budde, Matthias; Müller, Thomas; Laquai, Bernd; Streibl, Norbert; Schwarz, Almuth; Schindler, Gregor; Riedel, Till; Beigl, Michael; Dittler, Achim: Suitability of the Low-Cost SDS011 Particle Sensor for Urban PM-Monitoring. In: 3rd International Conference on Atmospheric Dust. 2018.

[Ch15] Chollet, François et al.: , Keras. `https://keras.io`, 2015.

[HS97] Hochreiter, Sepp; Schmidhuber, Jürgen: Long Short-term Memory. Neural computation, 9, 1997.

[Or06] Organization, World Health: Air quality guidelines: global update 2005: particulate matter, ozone, nitrogen dioxide, and sulfur dioxide. World Health Organization, 2006.

[5] https://flink.apache.org/
[6] https://spark.apache.org/

Autorenverzeichnis

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT:
Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik –
Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^{2}CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf INFOS 2011
14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landbewirtschaftung
Fokus Forstwirtschaft

P-195 Neeraj Suri, Michael Waidner (Hrsg.)
Sicherheit 2012
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)

P-196 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2012
Proceedings of the 11th International Conference of the Biometrics Special Interest Group

P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)
Software Engineering 2012
Fachtagung des GI-Fachbereichs Softwaretechnik

P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)
Software Engineering 2012
Workshopband

P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)
ARCS 2012 Workshops

P-201 Elmar J. Sinz Andy Schürr (Hrsg.)
Modellierung 2012

P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)
MMS 2012:Mobile und Ubiquitäre Informationssysteme

P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
5. DFN-Forum Kommunikations-technologien
Beiträge der Fachtagung

P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I2CS 2012)

P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-206 Stefanie Rinderle-Ma, Mathias Weske (Hrsg.)
EMISA 2012
Der Mensch im Zentrum der Modellierung

P-207 Jörg Desel, Jörg M. Haake, Christian Spannagel (Hrsg.)
DeLFI 2012: Die 10. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.
24.–26. September 2012

P-208 Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert Matthies, Wolf-Tilo Balke, Lars Wolf (Hrsg.) INFORMATIK 2012

P-209 Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.) Nachhaltiges Software Management

P-210 Erhard Plödereder, Peter Dencker, Herbert Klenk, Hubert B. Keller, Silke Spitzer (Hrsg.) Automotive – Safety & Security 2012 Sicherheit und Zuverlässigkeit für automobile Informationstechnik

P-211 M. Clasen, K. C. Kersebaum, A. Meyer-Aurich, B. Theuvsen (Hrsg.) Massendatenmanagement in der Agrar- und Ernährungswirtschaft Erhebung - Verarbeitung - Nutzung Referate der 33. GIL-Jahrestagung 20. – 21. Februar 2013, Potsdam

P-212 Arslan Brömme, Christoph Busch (Eds.) BIOSIG 2013 Proceedings of the 12th International Conference of the Biometrics Special Interest Group 04.–06. September 2013 Darmstadt, Germany

P-213 Stefan Kowalewski, Bernhard Rumpe (Hrsg.) Software Engineering 2013 Fachtagung des GI-Fachbereichs Softwaretechnik

P-214 Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mit schang, Theo Härder, Veit Köppen (Hrsg.) Datenbanksysteme für Business, Technologie und Web (BTW) 2013 13. – 15. März 2013, Magdeburg

P-215 Stefan Wagner, Horst Lichter (Hrsg.) Software Engineering 2013 Workshopband (inkl. Doktorandensymposium) 26. Februar – 1. März 2013, Aachen

P-216 Gunter Saake, Andreas Henrich, Wolfgang Lehner, Thomas Neumann, Veit Köppen (Hrsg.) Datenbanksysteme für Business, Technologie und Web (BTW) 2013 – Workshopband 11. – 12. März 2013, Magdeburg

P-217 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.) 6. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung 03.–04. Juni 2013, Erlangen

P-218 Andreas Breiter, Christoph Rensing (Hrsg.) DeLFI 2013: Die 11 e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI) 8. – 11. September 2013, Bremen

P-219 Norbert Breier, Peer Stechert, Thomas Wilke (Hrsg.) Informatik erweitert Horizonte INFOS 2013 15. GI-Fachtagung Informatik und Schule 26. – 28. September 2013

P-220 Matthias Horbach (Hrsg.) INFORMATIK 2013 Informatik angepasst an Mensch, Organisation und Umwelt 16. – 20. September 2013, Koblenz

P-221 Maria A. Wimmer, Marijn Janssen, Ann Macintosh, Hans Jochen Scholl, Efthimios Tambouris (Eds.) Electronic Government and Electronic Participation Joint Proceedings of Ongoing Research of IFIP EGOV and IFIP ePart 2013 16. – 19. September 2013, Koblenz

P-222 Reinhard Jung, Manfred Reichert (Eds.) Enterprise Modelling and Information Systems Architectures (EMISA 2013) St. Gallen, Switzerland September 5. – 6. 2013

P-223 Detlef Hühnlein, Heiko Roßnagel (Hrsg.) Open Identity Summit 2013 10. – 11. September 2013 Kloster Banz, Germany

P-224 Eckhart Hanser, Martin Mikusz, Masud Fazal-Baqaie (Hrsg.) Vorgehensmodelle 2013 Vorgehensmodelle – Anspruch und Wirklichkeit 20. Tagung der Fachgruppe Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik (WI-VM) der Gesellschaft für Informatik e.V. Lörrach, 2013

P-225 Hans-Georg Fill, Dimitris Karagiannis, Ulrich Reimer (Hrsg.) Modellierung 2014 19. – 21. März 2014, Wien

P-226 M. Clasen, M. Hamer, S. Lehnert, B. Petersen, B. Theuvsen (Hrsg.) IT-Standards in der Agrar- und Ernährungswirtschaft Fokus: Risiko- und Krisenmanagement Referate der 34. GIL-Jahrestagung 24. – 25. Februar 2014, Bonn

P-227 Wilhelm Hasselbring,
Nils Christian Ehmke (Hrsg.)
Software Engineering 2014
Fachtagung des GI-Fachbereichs Softwaretechnik
25. – 28. Februar 2014
Kiel, Deutschland

P-228 Stefan Katzenbeisser, Volkmar Lotz, Edgar Weippl (Hrsg.)
Sicherheit 2014
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
19. – 21. März 2014, Wien

P-229 Dagmar Lück-Schneider, Thomas Gordon, Siegfried Kaiser, Jörn von Lucke,Erich Schweighofer, Maria A.Wimmer, Martin G. Löhe (Hrsg.)
Gemeinsam Electronic Government ziel(gruppen)gerecht gestalten und organisieren
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI)
2014, 20.-21. März 2014 in Berlin

P-230 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2014
Proceedings of the 13th International Conference of the Biometrics Special Interest Group
10. – 12. September 2014 in Darmstadt, Germany

P-231 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
7. DFN-Forum Kommunikationstechnologien
16. – 17. Juni 2014
Fulda

P-232 E. Plödereder, L. Grunske, E. Schneider, D. Ull (Hrsg.)
INFORMATIK 2014
Big Data – Komplexität meistern
22. – 26. September 2014
Stuttgart

P-233 Stephan Trahasch, Rolf Plötzner, Gerhard Schneider, Claudia Gayer, Daniel Sassiat, Nicole Wöhrle (Hrsg.)
DeLFI 2014 – Die 12. e-Learning Fachtagung Informatik
der Gesellschaft für Informatik e.V.
15. – 17. September 2014
Freiburg

P-234 Fernand Feltz, Bela Mutschler, Benoît Otjacques (Eds.)
Enterprise Modelling and Information Systems Architectures
(EMISA 2014)
Luxembourg, September 25-26, 2014

P-235 Robert Giegerich, Ralf Hofestädt, Tim W. Nattkemper (Eds.)
German Conference on Bioinformatics 2014
September 28 – October 1
Bielefeld, Germany

P-236 Martin Engstler, Eckhart Hanser, Martin Mikusz, Georg Herzwurm (Hrsg.)
Projektmanagement und Vorgehensmodelle 2014
Soziale Aspekte und Standardisierung
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Stuttgart 2014

P-237 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2014
4.–6. November 2014
Stuttgart, Germany

P-238 Arno Ruckelshausen, Hans-Peter Schwarz, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Referate der 35. GIL-Jahrestagung
23. – 24. Februar 2015, Geisenheim

P-239 Uwe Aßmann, Birgit Demuth, Thorsten Spitta, Georg Püschel, Ronny Kaiser (Hrsg.)
Software Engineering & Management 2015
17.-20. März 2015, Dresden

P-240 Herbert Klenk, Hubert B. Keller, Erhard Plödereder, Peter Dencker (Hrsg.)
Automotive – Safety & Security 2015
Sicherheit und Zuverlässigkeit für automobile Informationstechnik
21.–22. April 2015, Stuttgart

P-241 Thomas Seidl, Norbert Ritter, Harald Schöning, Kai-Uwe Sattler, Theo Härder, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015)
04. – 06. März 2015, Hamburg

P-242 Norbert Ritter, Andreas Henrich, Wolfgang Lehner, Andreas Thor, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015) – Workshopband
02. – 03. März 2015, Hamburg

P-243 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
8. DFN-Forum Kommunikationstechnologien
06.–09. Juni 2015, Lübeck

P-244 Alfred Zimmermann, Alexander Rossmann (Eds.)
Digital Enterprise Computing (DEC 2015)
Böblingen, Germany June 25-26, 2015

P-245 Arslan Brömme, Christoph Busch , Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2015
Proceedings of the 14th International Conference of the Biometrics Special Interest Group
09.–11. September 2015
Darmstadt, Germany

P-246 Douglas W. Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.)
INFORMATIK 2015
28.9.-2.10. 2015, Cottbus

P-247 Hans Pongratz, Reinhard Keil (Hrsg.)
DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)
1.–4. September 2015
München

P-248 Jens Kolb, Henrik Leopold, Jan Mendling (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 6th Int. Workshop on Enterprise Modelling and Information Systems Architectures, Innsbruck, Austria
September 3-4, 2015

P-249 Jens Gallenbacher (Hrsg.)
Informatik allgemeinbildend begreifen
INFOS 2015 16. GI-Fachtagung Informatik und Schule
20.–23. September 2015

P-250 Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Martin Mikusz, Alexander Volland (Hrsg.)
Projektmanagement und Vorgehensmodelle 2015
Hybride Projektstrukturen erfolgreich umsetzen
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Elmshorn 2015

P-251 Detlef Hühnlein, Heiko Roßnagel, Raik Kuhlisch, Jan Ziesing (Eds.)
Open Identity Summit 2015
10.–11. November 2015
Berlin, Germany

P-252 Jens Knoop, Uwe Zdun (Hrsg.)
Software Engineering 2016
Fachtagung des GI-Fachbereichs Softwaretechnik
23.–26. Februar 2016, Wien

P-253 A. Ruckelshausen, A. Meyer-Aurich, T. Rath, G. Recke, B. Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Fokus: Intelligente Systeme – Stand der Technik und neue Möglichkeiten
Referate der 36. GIL-Jahrestagung
22.-23. Februar 2016, Osnabrück

P-254 Andreas Oberweis, Ralf Reussner (Hrsg.)
Modellierung 2016
2.–4. März 2016, Karlsruhe

P-255 Stefanie Betz, Ulrich Reimer (Hrsg.)
Modellierung 2016 Workshopband
2.–4. März 2016, Karlsruhe

P-256 Michael Meier, Delphine Reinhardt, Steffen Wendzel (Hrsg.)
Sicherheit 2016
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
5.–7. April 2016, Bonn

P-257 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
9. DFN-Forum Kommunikationstechnologien
31. Mai – 01. Juni 2016, Rostock

P-258 Dieter Hertweck, Christian Decker (Eds.)
Digital Enterprise Computing (DEC 2016)
14.–15. Juni 2016, Böblingen

P-259 Heinrich C. Mayr, Martin Pinzger (Hrsg.)
INFORMATIK 2016
26.–30. September 2016, Klagenfurt

P-260 Arslan Brömme, Christoph Busch, Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2016
Proceedings of the 15th International Conference of the Biometrics Special Interest Group
21.–23. September 2016, Darmstadt

P-261 Detlef Rätz, Michael Breidung, Dagmar Lück-Schneider, Siegfried Kaiser, Erich Schweighofer (Hrsg.)
Digitale Transformation: Methoden, Kompetenzen und Technologien für die Verwaltung
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2016
22.–23. September 2016, Dresden

P-262 Ulrike Lucke, Andreas Schwill, Raphael Zender (Hrsg.)
DeLFI 2016 – Die 14. E-Learning Fachtagung Informatik
der Gesellschaft für Informatik e.V. (GI)
11.–14. September 2016, Potsdam

P-263 Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Oliver Linssen, Martin Mikusz, Alexander Volland (Hrsg.)
Projektmanagement und Vorgehensmodelle 2016
Arbeiten in hybriden Projekten: Das Sowohl-als-auch von Stabilität und Dynamik
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik
der Gesellschaft für Informatik e.V., Paderborn 2016

P-264 Detlef Hühnlein, Heiko Roßnagel, Christian H. Schunck, Maurizio Talamo (Eds.)
Open Identity Summit 2016
der Gesellschaft für Informatik e.V. (GI)
13.–14. October 2016, Rome, Italy

P-265 Bernhard Mitschang, Daniela Nicklas,Frank Leymann, Harald Schöning, Melanie Herschel, Jens Teubner, Theo Härder, Oliver Kopp, Matthias Wieland (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2017)
6.–10. März 2017, Stuttgart

P-266 Bernhard Mitschang, Norbert Ritter, Holger Schwarz, Meike Klettke, Andreas Thor, Oliver Kopp, Matthias Wieland (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2017)
Workshopband
6.–7. März 2017, Stuttgart

P-267 Jan Jürjens, Kurt Schneider (Hrsg.)
Software Engineering 2017
21.–24. Februar 2017, Hannover

P-268 A. Ruckelshausen, A. Meyer-Aurich, W. Lentz, B. Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Fokus: Digitale Transformation – Wege in eine zukunftsfähige Landwirtschaft
Referate der 37. GIL-Jahrestagung
06.–07. März 2017, Dresden

P-269 Peter Dencker, Herbert Klenk, Hubert Keller, Erhard Plödereder (Hrsg.)
Automotive – Safety & Security 2017
30.–31. Mai 2017, Stuttgart

P-270 Arslan Brömme, Christoph Busch, Antitza Dantcheva, Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2017
20.–22. September 2017, Darmstadt

P-271 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
10. DFN-Forum Kommunikations-technologien
30. – 31. Mai 2017, Berlin

P-272 Alexander Rossmann, Alfred Zimmermann (eds.)
Digital Enterprise Computing (DEC 2017)
11.–12. Juli 2017, Böblingen

P-273 Christoph Igel, Carsten Ullrich, Martin Wessner (Hrsg.)
BILDUNGSRÄUME
DeLFI 2017
Die 15. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)
5. bis 8. September 2017, Chemnitz

P-274 Ira Diethelm (Hrsg.)
Informatische Bildung zum Verstehen und Gestalten der digitalen Welt
13.–15. September 2017, Oldenburg

P-275 Maximilian Eibl, Martin Gaedke (Hrsg.)
INFORMATIK 2017
25.–29. September 2017, Chemnitz

P276 Alexander Volland, Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Oliver Linssen, Martin Mikusz (Hrsg.)
Projektmanagement und Vorgehensmodelle 2017
Die Spannung zwischen dem Prozess und den Menschen im Projekt
Gemeinsame Tagung der Fachgruppen Projektmanagement und Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V. in Kooperation mit der Fachgruppe IT-Projektmanagement der GPM e.V., Darmstadt 2017

P-277 Lothar Fritsch, Heiko Roßnagel, Detlef Hühnlein (Hrsg.)
Open Identity Summit 2017
5.–6. October 2017, Karlstad, Sweden

P-278 Arno Ruckelshausen, Andreas Meyer-Aurich, Karsten Borchard, Constanze Hofacker, Jens-Peter Loy, Rolf Schwerdtfeger, Hans-Hennig Sundermeier, Helga Floto, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Referate der 38. GIL-Jahrestagung
26.–27. Februar 2018, Kiel

P-279 Matthias Tichy, Eric Bodden, Marco Kuhrmann, Stefan Wagner, Jan-Philipp Steghöfer (Hrsg.)
Software Engineering und Software Management 2018
5.–9. März 2018, Ulm

P-280 Ina Schaefer, Dimitris Karagiannis, Andreas Vogelsang, Daniel Méndez, Christoph Seidl (Hrsg.)
Modellierung 2018
21.–23. Februar 2018, Braunschweig

P-281 Hanno Langweg, Michael Meier, Bernhard C. Witt, Delphine Reinhardt (Hrsg.)
Sicherheit 2018
Sicherheit, Schutz und Zuverlässigkeit
25.–27. April 2018, Konstanz

P-282 Arslan Brömme, Christoph Busch, Antitza Dantcheva, Christian Rathgeb, Andreas Uhl (Eds.)
BIOSIG 2018
Proceedings of the 17th International Conference of the Biometrics Special Interest Group
26.–28. September 2018
Darmstadt, Germany

P-283 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
11. DFN-Forum Kommunikationstechnologien
27.–28. Juni 2018, Günzburg

P-284 Detlef Krömker, Ulrik Schroeder (Hrsg.)
DeLFI 2018 – Die 16. E-Learning Fachtagung Informatik
10.–12. September 2018, Frankfurt a. M.

P-285 Christian Czarnecki, Carsten Brockmann, Eldar Sultanow, Agnes Koschmider, Annika Selzer (Hrsg.)
Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit
26.–27. September 2018, Berlin

P-286 Martin Mikusz, Alexander Volland, Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Oliver Linssen (Hrsg.)
Projektmanagement und Vorgehensmodelle 2018
Der Einfluss der Digitalisierung auf Projektmanagementmethoden und Entwicklungsprozesse
Düsseldorf 2018

P-287 A. Meyer-Aurich, M. Gandorfer, N. Barta, A. Gronauer, J. Kantelhardt, H. Floto (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Fokus: Digitalisierung für landwirtschaftliche Betriebe in kleinstrukturierten Regionen – ein Widerspruch in sich?
Referate der 39. GIL-Jahrestagung
18.–19. Februar 2019, Wien

P-289 Torsten Grust, Felix Naumann, Alexander Böhm, Wolfgang Lehner, Jens Teubner, Meike Klettke, Theo Härder, Erhard Rahm, Andreas Heuer, Holger Meyer (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2019)
4.–8. März 2019 in Rostock

P-290 Holger Meyer, Norbert Ritter, Andreas Thor, Daniela Nicklas, Andreas Heuer, Meike Klettke (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2019)
Workshopband
4.–8. März 2019 in Rostock

P-291 Michael Räckers, Sebastian Halsbenning, Detlef Rätz, David Richter, Erich Schweighofer (Hrsg.)
Digitalisierung von Staat und Verwaltung
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2019
6.–7. März 2019 in Münster

P-292 Steffen Becker, Ivan Bogicevic,Georg Herzwurm, Stefan Wagner (Hrsg.)
Software Engineering and Software Management 2019
18.–22. Februar 2019 in Stuttgart

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-684-8

"BTW 2019" is the 18th event in a conference series focusing on a broad range of database topics from a variety of perspectives. With its emphasis on lively discussions and cross-fertilization of academia and industry, it provides a valuable platform to further the state of the art in database foundations and techniques for high-performance query processing and optimization, cloud data management, text and graph processing, and big data analytics, among others. This volume contains contributions from the refereed workshop program, the refereed student program, the tutorials, and the data science challenge.