43rd International Conference on Very Large Data Bases

Peter Christen, Bettina Kemme, Erhard Rahm (eds.):

Proceedings of the VLDB 2017 PhD Workshop

August 28, 2017

Munich, Germany

Preface

This volume contains the proceedings of the VLDB 2017 PhD Workshop, co-located with the 43nd International Conference on Very Large Database (VLDB 2017) and held on August 28, 2017, in Munich, Germany.

The established VLDB PhD Workshop series is a unique opportunity for graduate students to present and discuss their research in the context of a premier international conference. The workshop provides a forum that facilitates interactions among PhD students and stimulates feedback from more experienced researchers—not only in terms of diligent paper reviews but also at the conference site.

Following the call for papers, 27 papers were submitted for presentation at the workshop. Of these, the program committee carefully selected a total of 14 papers, with 10 selected for long and 4 for short presentations, which are contained in this volume.

Long presentation papers:

- Efficient Migration of Very Large Distributed State for Scalable Stream Processing.
  Bonaventura Del Monte, DFKI Berlin, Germany
- Spatio-temporal Locality in Hash Tables.
  Matthew A Pugh, University of Edingburgh, UK
- Graph Pattern Mining for Business Decision Support.
  André Petermann, University of Leipzig, Germany
- Efficient Clustering for Large□Scale, Sparse, Discrete Data with Low Fundamental Resolution.
  Veronika Strnadova□Neeley, UC Santa Barbara, USA
- Query Processing Based on Compressed Intermediates.
  Patrick Damme, Technical University Dresden, Germany
- A Hardware□Oblivious Optimizer for Data Stream Processing.
  Constantin Pohl, Technical University Ilmenau, Germany
- Generalizing Matching Knowledge using Active Learning.
  Anna Primpeli, University of Mannheim, Germany
- Comparing entities in RDF graphs.
  Alina Petrova, University of Oxford, UK
- Scalable Linkage across Location Enhanced Services.
  Fuat Basik, Bilkent University, Turkey
- Distributed Similarity Joins on Big Textual Data: Toward a Robust Cost□Based Framework.
  Fabian Fier, Humboldt University Berlin, Germany

Short presentation papers:

- Facilitating User Interaction With Data.
  Zainab Zolaktaf, University of British Columbia Vancouver, Canada
- Processing Moving Object Data Streams with Data Stream Management Systems.
  Tobias L Brandt, University of Oldenburg, Germany
- Symmetric and Asymmetric Aggregate Function in Massively Parallel Computing.
  Chao Zhang, Blaise Pascale University Aubière, France
- Practical Verification of Hierarchical Artifact Systems.
  Yuliang Li, UC San Diego, USA

The workshop program also featured a keynote talk by Prof. Dr. Anastasia Ailamaki, (EPFL, Switzerland) who talked about the past and future of database transaction processing engines. The workshop ended with a panel consisting of leading database experts including Michael Carey (University of California, Irvine, USA), Wolfgang Lehner (Technische Universität Dresden, Germany), and Themis Palpanas (Paris Descartes University, France) who discussed the “life with a PhD” and provided valuable tips and suggestions to the PhDs who attended the workshop.

We like to thank all authors who submitted papers to the VLDB 2017 PhD workshop, to Anastasia for her keynote presentation, and to the panel members for interesting discussions.

Our sincere thanks also go to all members of the program committee for their insightful and valuable reviews. All of you have helped to make the VLDB 2017 PhD Workshop a successful event.

Peter Christen (The Australian National University, Australia)
Bettina Kemme (McGill, Montreal, Canada)
Erhard Rahm (Univ. of Leipzig, Germany)

PC Co-Chairs
August 2017

PhD Workshop Program Committee

Angela Bonifati (University of Lyon)
Stéphane Bressan (National University of Singapore)
George Fletcher (TU Eindhoven)
Lukasz Golab (University of Waterloo)
Carson Leung (University of Manitoba)
Sebastian Link (University of Auckland)
Rachel Pottinger (University of British Columbia)
Louiqa Raschid (University of Maryland)
Donatello Santoro (Università della Basilicata)
Timos Sellis (Swinburne University of Technology)
Andreas Thor (HFT Leipzig)
Wei Wang (University of New South Wales)
Qing Wang (The Australian National University)

Keynote by Prof. Dr. Anastasia Ailamaki (EPFL, Switzerland):

Title: **The Next 700 Transaction Processing Engines**

Abstract:

For over four decades, throughput has been the target metric of choice for Online Transaction Processing engines. Around mid-2000s, however, Dennard scaling came to a crushing halt and now multicore processors provide explicit thread-level parallelism as an alternative to frequency scaling for increasing throughput. Thus, OLTP research focuses on developing scalable synchronization techniques for exploiting parallelism provided by multicore processors. In the late 2000s, DRAM price free-fall made it possible to fit a single server with Terabytes of memory, and to fit most operational databases, with the exception of a few rare cases, entirely in memory. This led to a flurry of research on the design of scalable main-memory OLTP engines that adopt radically different designs compared to their disk-based counterparts.

Today, state-of-the-art main-memory OLTP engines can handle millions of transactions per second and provide near-linear scalability under most workloads. However, three recent trends indicate an impending change in OLTP engine design once again: 1) changes in application workloads, 2) shifting hardware landscape, and 3) new target metrics. In this talk, we will discuss the implications of these trends on the design of next-generation transactional engines, and explore new designs with the twin goal of meeting changing application demands and optimizing for the new metrics by exploiting emerging hardware.

Biography:

Anastasia Ailamaki is a Professor of Computer and Communication Sciences at the Ecole Polytechnique Federale de Lausanne (EPFL) in Switzerland. Her research interests are in data-intensive systems and applications, and in particular (a) in strengthening the interaction between the database software and emerging hardware and I/O devices, and (b) in automating data management to support computationally-demanding, data-intensive scientific applications. She has received an ERC Consolidator Award (2013), a Finmeccanica endowed chair from the Computer Science Department at Carnegie Mellon (2007), a European Young Investigator Award from the European Science Foundation (2007), an Alfred P. Sloan Research Fellowship (2005), eight best-paper awards in database, storage, and computer architecture conferences, and an NSF CAREER award (2002). She holds a Ph.D. in Computer Science from the University of Wisconsin-Madison in 2000. She is an ACM fellow and the vice chair of the ACM SIGMOD community, a senior member of the IEEE, and an elected member of the Swiss National Research Council. She has served as a CRA-W mentor, is a member of the Expert Network of the World Economic Forum.

Efficient Migration of Very Large Distributed State for Scalable Stream Processing

Bonaventura Del Monte
supervised by Prof. Volker Markl
DFKI GmbH
bonaventura.delmonte@dfki.de

ABSTRACT

Any scalable stream data processing engine must handle the dynamic nature of data streams and it must quickly react to every fluctuation in the data rate. Many systems successfully address data rate spikes through resource elasticity and dynamic load balancing. The main challenge is the presence of stateful operators because their internal, mutable state must be scaled out while assuring fault-tolerance and continuous stream processing. Both rescaling, load balancing, and recovering demand state movement among work units. Therefore, how to guarantee those features in the presence of large distributed state with minimal impact on the performance is still an open issue. We propose an incremental migration mechanism for fine-grained state shards through periodic *incremental checkpoints* and *replica groups*. This enables moving large state with minimal impact on stream processing. Finally, we present a low-latency hand-over protocol that smoothly migrates tuples processing among work units.

1. INTRODUCTION

Existing scalable *Stream Data Processing Engines* (SPEs) offer fast stateful processing of data streams with low latency and high throughput despite fluctuations in the data rate. To this end, stateful processing benefits from on-demand resource elasticity, load balancing, and fault tolerance. Currently, both research [5, 6, 17, 13] and industry [1, 4, 18] address scaling up stateful operators while assuring fault tolerance *in case of partitioned or partially distributed large state*. Here, large state means hundreds of gigabytes.

A motivating example. Many streaming applications require stateful processing. Examples of such applications are the data analytics stacks behind popular multimedia services, online marketplaces, and mobile games. These stacks perform complex event processing on live streams. Multimedia services and online marketplaces recommend new contents or items to their users through collaborative filtering [16]. Producers of mobile games track in-game behaviour of players to promote the best in-app purchase and to detect frauds. The size of the state in these applications scales with the number of users and their interactions with the application (e.g., rated items, purchases, actions of a player) and can grow to terabyte sizes. State in the size of terabytes introduces a multifaceted challenge. The SPE must optimally manage cluster resources respecting the size of the state. This does not only apply to intra-cluster instances but also inter-cluster ones, e.g., migrating the SPE among operational environments or to cheaper "pay-as-you-go" instances. Besides, parallel analytic algorithms need global state. Parallel instances of an operator work on their state partition and then update global state, e.g., machine learning models. Therefore, these analytics result in *very large distributed state*.

Research goal. Motivated by industrial needs, our goal is to achieve stream processing with low latency and high throughput when operators handle very large state. To this end, we focus on management techniques that enable fault-tolerance, on-demand resource scaling, and load balancing in the presence of very large distributed state.

Problem statement. Handling operators with very large distributed state is cumbersome. Guaranteeing fault-tolerance, resource elasticity, and dynamic load balancing for these operators (i) require state transfer, (ii) must not undermine the consistency of distributed state shards, and (iii) demand robust query processing performance. State transfer introduces latency proportional to its size. Exactly-once stream processing requires consistent state, i.e., results must be as accurate as if no failure happened or the SPE did not perform any rescaling or rebalancing operation on the state. Besides, a SPE must continuously process stream tuples despite any of those operations.

Current approaches. To the best of our knowledge, there is no system that fully features efficient state management when distributed very large state is involved. Many authors investigated this problem by constraining their scope to partitioned or partially distributed state [5, 4, 18] and to smaller size [7, 6, 17].

Proposed solution. Our solution is a low-latency incremental migration mechanism that moves fine-grained state shards by using periodic *incremental checkpoints* and *replica groups*. An incremental checkpoint is a periodic snapshot of a state shards that involves only modified values. A replica group is a set of computing instances holding a copy of a portion of the state. Our migration mechanism moves large operator states with low impact on the system performance and without stopping the streaming topology. Although incremental migration reduces the transfer overhead, we also provide a placement scheme for primary state shards and replica groups that minimizes transfer cost. Our solutions are as follows:

1. a communication-efficient replication protocol that keeps a replica group consistent with the changes in the state of the primary operator
2. an optimal primary state shards and replica groups placement for decreasing migration cost
3. a hand-over protocol that migrates the processing between two work units with minimal latency.

We point out that this thesis is at an early stage, hence, we do not have any experimental validation yet.

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

2. RELATED WORK

Castro et al. address the problem of scaling up and recovering stateful operators in a cloud environment through a set of primitives for state management that enables scaling up and recovery of stateful operators [5]. Their experiments include operators with small states and they confirmed that larger state has a higher recovery time. In a second work, the same authors propose a new abstraction over large mutable state, called stateful dataflow graph, which manages partitioned or partial distributed state [6]. Our aim is to fill the gap in this area by providing a mechanism that both scales out and recovers a long-running system with very large distributed state. ChronoStream is a system that seamlessly migrates and executes tasks [17], whose authors believe to have achieved costless migration thank to a locality-sensitive data placement scheme, delta checkpointing, and a lightweight transactional migration protocol. Although their experiments look promising, we argue transactional migration may be avoided by using two different protocols (one for state migration and one for the hand-over) and delta checkpoints adds synchronization issues.

Ding et al. deal with finding the optimal task assignment that minimizes the costs for state migration and satisfies load balancing constraints [7]. To this end, they introduce a live and progressive migration mechanism with negligible and controllable delay. They come to a different conclusion w.r.t. ChronoStream, because they also argue that synchronization issues may affect results correctness while performing a migration. The solution of Ding et al. performs multiple mini-migrations progressively: each mini-migration migrates a number of tasks smaller than a given threshold [7]. On the other hand, their experiments do not cover large state migration and it is unclear how the system could perform in such task. Furthermore, both ChronoStream and Ding et al. consider partitioned state.

Nasir et al. present *partial key grouping* as a solution to handle load imbalance caused by skewness in the keys distribution of input streams [14, 15]. The main idea is to keep track of the number of items in each parallel instance of an operator and route a new item to the instance with smaller load. Items with the same key are routed to different parallel instances of the same operator. An improvement to the solution is to determine the "hottest" keys in the stream and assign more workers to those keys. However, they assumed the operator state has the associative property, thus merging intermediate partitioned substates is possible with an extra aggregate operation. Splitting the state of a given key, indeed, mitigates its growth on one working unit, yet aggregating large state will require some potentially expensive network transfers. Our aim is to propose a load balancing approach that avoids such partial aggregations.

Gedik et al. propose transparent auto-parallelization for stream processing through a migration mechanism [8]. However, we argue that their approach does not consider distributed large state and it is totally decoupled from fault-tolerance.

Many SPEs have effectively implemented state management techniques (e.g., Apache Flink [1, 4], Apache Spark [18], SEEP [6], Naiad [13]). In particular, Apache Flink features a technique that asynchronously checkpoints the global states to minimize the latency of a snapshot [3].

3. RESEARCH ISSUES

Our goal is to move large operator states with minimal impact on the performance of query processing. Migrating large states between operator instances in one shot is expensive due to network transfer, especially if the system is already overloaded during its regular operation. Our key idea is to incrementally maintain a replica group for each fine-grained state unit over different work units. Each replica is updated through *incremental checkpoints* generated on the primary operator. In addition, intrinsic issues of migration pose new challenges, e.g., data consistency, tuples rerouting, physical shards handling, and network transfer cost. To better explain our key idea, we first define our data and system models. Then we provide an analysis of our research goals.

3.1 System Model

Data Model. Let S be a stream of tuples, for each tuple $q \in S$, we define k_q as the value of the partitioning key and t_q as its monotonically generated time-stamp.

Stream processing. Our system is made of p work units running on z physical nodes (each of them can run a variable number of work units). Our system executes jobs/queries expressed as a dataflow graph. Each operator of the graph runs on maximum p parallel instances. An operator takes n streams and outputs m streams. Every parallel instance receives tuples (sent from upstream operators) w.r.t. a distribution function that computes the assignments through k_q.

State model. The global state of all the operators in the streaming topology is a distributed logically partitionable data store (e.g., a distributed K-V store). Partitions of this data store contain a single state entry, e.g., window content of an operator, user-defined counters. Each logical partition is made of physical shards. Every parallel instance of an operator holds its own shard. Besides, each shard is made of fine grained data items. Each key of the input stream owns few data item in every logical partition of the state and each shard holds a range of keys. Each range of keys can be further partitioned and optionally split. Distributed state demands some consistency guarantee in case (i) a key needs to be stored in multiple shards, and (ii) tuple processing might trigger changes in more than one shard. The distribution function determines the content of the shards kept by stateful instances. Thus, each parallel instance of an operator does not only process tuples with specific keys but it also holds the data items of state for those keys.

3.2 Incremental Checkpoints

A prerequisite for our set of protocols is an incremental checkpoint protocol based on the approach of Carbone et al. [3]. Instead of taking a snapshot of the whole state, we asynchronously checkpoint the modified state values between the previous checkpoint and the current one. An asynchronous checkpoint executed at time time t will not contain updates happened later than t.

3.3 Replication Protocol

We design a replication protocol to keep the global state of a streaming topology replicated and consistent. This protocol replicates every primary state of each operator instance on a given number of work unit, i.e., each sub-range of keys has its own replica group. The purpose of a replica group is to keep a copy of different sub-ranges of keys for each operator. A primary operators sends incremental checkpoints for a given range of keys to its replica through the network.

3.4 Hand-Over Protocol

The hand-over protocol moves the processing of a given keys range (k_s,k_e) between two parallel instance of a target stateful operator. The system triggers this protocol when it detects the need of either rescaling an operator, balancing the load over parallel instances of an operator, or recovering an operator. Main ideas behind this protocol are the usage of replica groups, incremental checkpoints and the embedding of the protocol itself in the dataflow paradigm. Moving the processing of any key involves tuple rerouting and migration of the state for that key. This operation is lightweight if the destination instance is in

the replica group of the moved key. Indeed, the replica group misses at most the last incremental checkpoint. Let *upstream* be all the operators that send some input tuples to a target downstream operator, the steps of the protocol are:

1. The system decides to migrate that tuples marked with keys in range (k_s,k_e), from downstream instance o_s to o_t, which is in the replica group of (k_s,k_e)
2. Upstream injects a *key move* event in the data flow for keys k_s,...,k_e involving operators o_s and o_t
3. Upstream sends its outgoing tuples marked with keys k_s,...,k_e to o_t, which processes them creating new states s'_e,...,s'_t
4. o_s generates an incremental checkpoint that contains its current states s_e,...,s_t for keys k_s,...,k_e and sends it to o_t
5. As soon as o_t gets the incremental checkpoint, it updates its current states s_e,...,s_t with the received checkpoint
6. Then o_t asynchronously merges them with s'_e,...,s'_t. If new tuples arrive in o_t, it generates new states and subsequently merges them.

As a result, the handover protocol guarantees eventual consistency on every migrated primary state after merging. Moreover, we assume that user-defined state has *update* and *merge* policies; the former updates state by processing a stream tuple, whereas the latter merges two partial states for the same key. If merging of partial state is not semantically possible, then the target instance buffers incoming tuples and updates the state upon its full receiving.

3.5 Optimal Placement of Replica Groups

Each keys replica group is composed of q physical nodes, as we aim to minimize continuous migration cost, the replica group has to be optimally placed over the streaming topology. Indeed, transferring an incremental checkpoint from a node a to b could potentially have a different cost than shipping the same checkpoint to node c. This problem can be mapped as a bipartite graph matching problem whose classic solution is well-know as the Hungarian or Kuhn-Munkres algorithm [11, 12]. Nevertheless, our scenario is not static as we need to deal with resource scaling and failing nodes. Therefore, a dynamic approach to the assignment problem [10] is the best fit for our needs, since we look for an optimal assignment of the state items to an elastic set of physical nodes. Our optimization problem is formulated as follows: given l sub-ranges of keys and z physical nodes, find a placement for each sub-range of keys over q out of z nodes that minimizes the migration cost. We evaluate the cost of shipping an incremental checkpoint between two nodes by considering their workloads and the number of network switches involved.

4. RESEARCH PLAN

In this thesis, we intend to investigate above research issues w.r.t. our goal: transparently providing fault-tolerance, resource elasticity, and load balancing in the presence of very large distributed state. Our focus is to investigate the trade-offs behind our proposed solution. First, the hand-over protocol presents several challenges, e.g., the granularity of the keys ranges, the concurrent execution of the protocol, and the triggering policy of the protocol (through either consensus, common knowledge or centralized entity). Secondly, we plan to investigate the usage of log-less replication (similarly to Bizur [9]) by using shared registers [2]. Besides, log-less replication implies no log compaction overhead. As network is the main bottleneck, we plan to research orthogonal optimizations to reduce network overhead, e.g, remote direct memory access, data compression, and approximation. Lastly, the placement scheme of replica groups may require further investigation as our initial definition of migration might neglect significant hidden cost.

4.1 Achievement Plan

We have a clear idea about the achievement of our goal, which we define in the following sections.

Fault-tolerance. Our replication protocol guarantees that each replica group holds a copy of some ranges of keys for different operators. Since each key is replicated in $q+1$ physical units, the system can sustain up to q failing instances of an operator by resuming the computation on one unit in its replica group. The system may need to replay some tuple unless the group has the latest state checkpoint and the failing unit did not process any newer tuple.

Load balancing. Relying on a load balancing policy (e.g., shard size or ingested tuples count above a given threshold), the system triggers the hand-over protocol. Then, the hand-over protocol seamlessly moves the processing of some keys ranges from a primary work unit to another in their replica groups. Determining the placement of ranges of keys for the primary state is another orthogonal challenge that we plan to overcome.

Resource elasticity. Regardless of the chosen elasticity policy, we need to efficiently rescale the state of every range of keys along with its replicas minimizing the transfer cost. Rescaling possibly involves deleting some replicas, whereas state transfer can be still done incrementally by using above protocols. As we consider primary state and replica as one entity, we reassign them to parallel instances as described in Section 3.5. This procedure could benefit from current IaaS platforms where multiple VMs or containers share physical hardware. Indeed, we may provision new resources on either an already used physical node or a new node. The last scenario is more challenging as the system must migrate entire shards of the state to the new node.

4.2 The system in action

In Figure 1, we show a toy example of our system while it seamlessly performs resource scaling, state recovery, and load balancing. The figure shows a simple dataflow graph made of one source (parallelism=2) and one operator (parallelism=4). For the sake of simplicity, we marked tuples and state for the same keys range with the same colour. Each primary state for every keys range has only one replica group. In Figure 1.A, the first instance is failing while the third one is overloaded. In Figure 1.B, the hand-over protocol seamlessly moves the processing and the state of both yellow and violet keys ranges. As the state of the yellow key range was on a failing node, our system must reply lost tuples. Meanwhile, the fourth instance processes violet tuples and creates a new partial state. The hand-over protocol merges this partial state with the current replica and the last incremental checkpoint. Simultaneously, our system provisions a new instance and migrates the red state as it detects an overloaded second instance. In Figure 1.C, the system is finally stable. The violet state is migrated and replicated on the fourth and second instance, respectively. The yellow state is restored on the second instance and replicated on the new instance. The red state is replicated on the new instance.

4.3 Evaluation Plan

We assess the capabilities of our system through the following set of *Key Performance Indicators* (KPIs):

1. the execution of our protocols must have negligible effect on query processing performance
2. the system must guarantee exactly-once stream processing and state consistency
3. performing a load balancing or a resources scaling operation must improve resource utilization of the physical infrastructure and prevent bottlenecks (e.g., operator back-pressure)

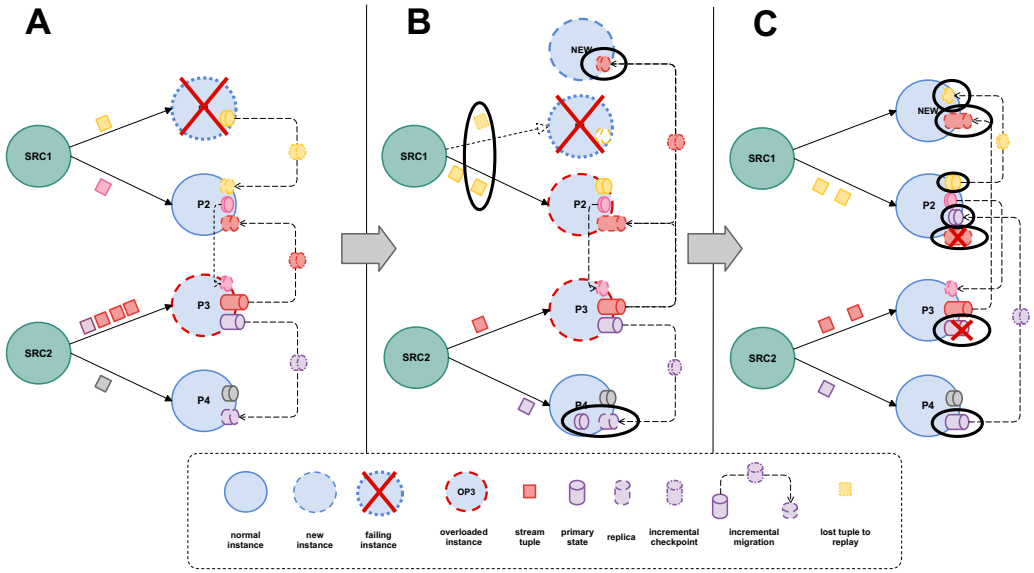


Figure 1: Our protocols in action: tuples and state for the same keys range are marked with the same colour. Primary state for every keys range is incrementally replicated only once. Sensible steps of the hand-over protocol are circled.

To meet above KPIs, we intend to design a suite of benchmarks that thoroughly stresses our proposed system. We plan to define a set of metrics (e.g., tuple processing throughput and latency, migrated state items, checkpoint size) and measure them in our system on different real-world workloads, with distinct scaling and balancing policies, and different replica factors. Finally, we expect to compare our results with baseline systems.

4.4 Future directions

We envision a system able to continuously process stream tuples despite data rate spikes and failures. This system can also seamlessly migrate itself among cluster, e.g., from one IaaS-provider to a cheaper vendor, between two operational environments. Incremental state migration will be a building block of such operations. Other orthogonal research areas may be: (i) investigating the usage of new storage hardware, e.g, NVRAM and SSD, (ii) considering non-keyed state and queryable state, (iii) providing elastic job maintenance, (iv) exploring data compression techniques to reduce state size, and (v) investigating incremental state migration (and resource elasticity) in case of Hybrid Transactional-Analytical Processing (HTAP) workloads.

Acknowledgments: We would like to thank our advisor Prof. Volker Markl, as well as Prof. Tilmann Rabl for his valuable guidance, as well as Dr. Asterios Katsifodimos, Dr. Sebastian Breß, and Dr. Alireza Rezaei Mahdiraji for their support. This work has been partially supported by the European Commission through PROTEUS (ref. 687691).

5. REFERENCES

[1] A. Alexandrov, R. Bergmann, et al. The stratosphere platform for big data analytics. *The VLDB Journal*, 2014.

[2] H. Attiya, A. Bar-Noy, et al. Sharing memory robustly in message-passing systems. In *ACM PODC*. 1990.

[3] P. Carbone, G. Fóra, et al. Lightweight asynchronous snapshots for distributed dataflows. *CoRR*, abs/1506.08603, 2015.

[4] P. Carbone, A. Katsifodimos, et al. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 30(40), 2015.

[5] R. Castro Fernandez, M. Migliavacca, et al. Integrating scale out and fault tolerance in stream processing using operator state management. In *ACM SIGMOD*. 2013.

[6] —. Making state explicit for imperative big data processing. In *USENIX ATC*. 2014.

[7] J. Ding, T. Fu, et al. Optimal operator state migration for elastic data stream processing. *CoRR*, abs/1501.03619, 2015.

[8] B. Gedik, S. Schneider, et al. Elastic scaling for data stream processing. *IEEE Trans. Parallel Distrib. Syst.*, 2014.

[9] E. Hoch, Y. Ben-Yehuda, et al. Bizur: A key-value consensus algorithm for scalable file-systems. *CoRR*, abs/1702.04242, 2017.

[10] G. A. Korsah, A. T. Stentz , et al. The dynamic hungarian algorithm for the assignment problem with changing costs. Tech. Rep. CMU-RI-TR-07-27, 2007.

[11] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955.

[12] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 1957.

[13] D. Murray, F. McSherry, et al. Naiad: A timely dataflow system. In *ACM SOSP*. 2013.

[14] M. Nasir, G. Morales, et al. The power of both choices: Practical load balancing for distributed stream processing engines. *CoRR*, abs/1504.00788, 2015.

[15] —. When two choices are not enough: Balancing at scale in distributed stream processing. *CoRR*, abs/1510.05714, 2015.

[16] R. Sumbaly, J. Kreps, et al. The big data ecosystem at linkedin. In *ACM SIGMOD*. 2013.

[17] Y. Wu and K. Tan. Chronostream: Elastic stateful stream computation in the cloud. In *IEEE ICDE*. 2015.

[18] M. Zaharia, T. Das, et al. Discretized streams: Fault-tolerant streaming computation at scale. In *ACM SOSP*. 2013.

Spatio-Temporal Locality in Hash Tables

Matt A. Pugh
Supervised by Prof. Stratis Viglas
University of Edinburgh
10 Crichton St.
Scotland
matt.pugh@ed.ac.uk

ABSTRACT

The overall theme of this Ph.D. is looking at ways to use emerging NVM (Non-Volatile Memory) technologies in real-world data-science scenarios. It is hoped that the exploitation of the characteristics of the technology will result in performance improvements, defined as being either/or an increase in computational throughput and energy-use reduction. Primarily, this has been through the inclusion of temporal locality into HopH (Hopscotch Hashing) by [2]. The problem of highly-skewed access patterns affecting lookup time and required computation is shown through a simple model. A simulator is then used to measure the expected performance gains of incorporating temporal locality, given different HT (Hash Table) configurations. This work was originally motivated by NVM, as a way to mask the extra latency anticipated, but the work is applicable to HTs in DRAM (Dynamic Random-Access Memory) also. The second area of interest in the Ph.D. is looking at exploiting the characteristics of NVM for different families of machine learning algorithms, though this paper focuses solely on the former.

1. INTRODUCTION

At this stage, the primary focus has been at introducing spatial and temporal locality into HopH – taking the concept of minimising amortised average lookup time into HopH will provide spatio-temporal locality with an amortised complexity of $\mathbf{O}(1)$ for the typical HT operations; `Get()`, `Insert()`, and `Delete()`. Details on the approaches undertaken are provided in Section 3. This is motivated by skewed access patterns, following Zipf's law, and high-performance systems, such as RDBMS (Relational DataBase Management System) join operators.

In its simplest form, a Hash Table (HT) T is an M-bucket long associative array-like data structure that maps a key k of any hashable type to value v. A given bucket B_i in T can have S slots for key/value tuples. The occupancy (or load factor) of a HT is $0 \geq L \geq 1$[1]. The benefit of a HT is that the associated value's position i in memory is calculated directly using a hash function $h(k)$, $i = h(k) \mod M$.

1.1 Self-Reorganising Data-Structures

A ST (Splay Tree) is a self-adjusting binary search tree introduced by [6] that has the property that the most frequently accessed node is near the head of the tree. STs have the amortised time-complexity for `Get()` and `Insert()` operations of $\mathbf{O}(\log n)$, and the goal of their invention was to minimise average time of lookups in worst-case scenario in real-time terms. Their solution to achieving this is to minimise the number of node traversals by ensuring the most-frequently-accessed nodes are as close to the head as possible, a property which is closely tied to temporal locality, although not exactly the same thing. [6] acknowledge that the computational cost of ongoing reordering is high, and may be problematic given the target application workload. This is clear when considering the rotation of sub-trees in a `Splay()` operation – there is no spatial locality, or cache-friendliness, guaranteed by the structure of the tree in memory, as such pointer-chasing in rotations is inevitable.

In databases, [3] describe *cracking*; a method of self organisation data by query workload. This process is tuned for a RDBMS in which a column of interest A_{CRK} is retained as a copy, and is continuously reorganised based on the queries touching it. This is analogous to data *hotness* as we wish to view it, and can be considered a cache-tuning problem, similar to a LRU (Least Recently Used) cache. As A_{CRK} is a column, they are able to effectively partition the data itself to improve probing response times significantly. Partitioning within a HopH context is less intuitive as there are, at any position, H overlapping neighbourhoods.

2. MODEL AND SIMULATOR

The structure of a HT is entirely dependent on the order in which keys were inserted into it. For example, the hottest element of the neighbourhood, or even entire table, over some given time period, may be in the last available bucket in a neighbourhood, leading to multiple evaluations before finding the tuple required.

2.1 Model

Assume a constructed Hash Table T, of which we have a set of all keys K in T. Let X be the sequence of accesses,

[1]Assuming no chained-structures resulting in $L > 1$ load factor

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

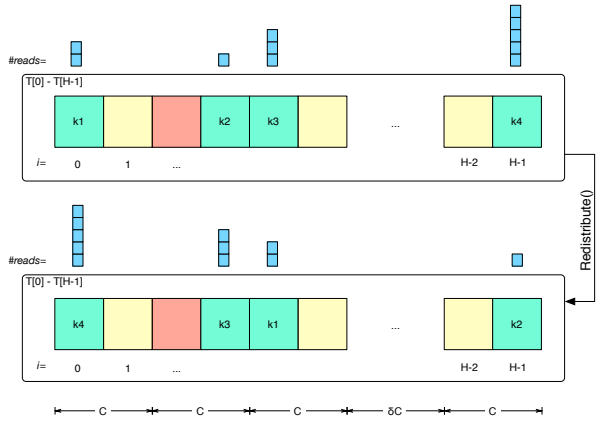


Figure 1: This figure illustrates buckets (coloured blocks) in neighbourhoods (colours) over some contiguous range of buckets within the table T. For a given neighbourhood, access patterns may naturally fall as the above distribution, where C is the cache-line size and each blue block represents a constant number of accesses to the associated key k_i. Suppose a request is made for the hottest item k_4, there will be $3 + \delta$ cache-invalidations before hitting the hottest data k_4 if $T[h(k_4)]$ is not already cached. Reordering the data such that k_4 is in position 0, minimising cache-invalidations, is the optimal distribution. Note that the distribution of colours in T does not change.

such that every element $x \in X$ exists in K. We define a simple cost model $\Phi(T, X)$ that gives a unit cost of a bucket traversal as β, and a slot traversal and key comparison as σ unit cost, where α and γ are the number of respective operations.

$$\Phi(T, X) = \alpha\beta + \gamma\sigma \tag{1}$$

It is simple to use this model with a sufficient access-skew in X to show good potential gains in the worst case of the configuration of T. Consider a RO (Read-Only) T where some key k has hashed into bucket B_i, but the only available slot in the otherwise fully-occupied neighbourhood ν_i is at B_{i+H-1}, slot S. Assume that X is completely skewed, such that there is only one k present, repeated A times. In this case, the cost of accesses is:

$$\Phi(T, X) = AH(\beta) + ASH(\sigma)$$

In this example, it is clear to see that minimising the values of $\alpha = AH$ and/or $\gamma = ASH$ are the only areas of movement upon which we can optimise, if we assign some real-world values, where $S = 3, H = 64, \sigma = 15, \beta = 15, A = 1 \times 10^9$, we obtain a cost of:

$$\Phi(T, X) \simeq 3.84 \times 10^{12}$$

When we assume that X' the hottest element k' in the first slot of the first bucket in a neighbourhood, the cost is naturally far lower:

$$\Phi(T, X') = A(\beta + \sigma) \simeq 3.0 \times 10^{10}$$

This is the lowest possible cost incurred for T and X', and is therefore referred to as the *oracle*. With this reordering, 99.22% of cycles would be saved versus T with X. This approach has the problem that manually attempting to cover a number of configurations of T to find the expected benefit of rearrangement would be tedious. To deal with this, we employ a simulator.

2.2 Simulator

The developed simulator provides the estimated performance benefit of performing reordering for varying table sizes, load factors and skews. This is done using a number of discrete probability distributions to obtain different configurations of T, in order to explore the problem space we are interested in:

1. The number of slots that each bucket will have populated in the simulator's construction is sampled from a Multinomial distribution, for bucket B_i this is ς_i.

2. $N = \lfloor S \times M \times L \rfloor$ samples are drawn from a Zipfian distribution $\mathcal{Z}(\alpha)$. These are the random *hotness* values that are then inserted into each bucket $B_i \in T$ over ς_i slots in each bucket. This approach gives a load factor L and skew α over the data. T is then randomly permuted to distribute the remaining (for $L < 1$) empty buckets throughout the table.

3. As the number of buckets occupied within a neighbourhood ν_i is not a constant, either $\lfloor \xi \rfloor$ or $\lceil \xi \rceil$ buckets per neighbourhood are allocated with probability $\pi_\beta = \lceil \xi \rceil - \lfloor \xi \rfloor$, where ξ is the *expected number of elements in a neighbourhood*, given in [5]. π_β must be within the $[0, 1]$ interval, and is interpreted as the probability that the neighbourhood contains the upper-bound of entries. At each neighbourhood root, neighbourhood occupancy is sampled from a Binomial distribution with probability π_β.

4. Finding the locations within the neighbourhood for occupancy should not be done linearly, and must be at least partly stochastic to emulate the chronological insertions over multiple neighbourhoods. Approaching this in a linear manner would construct T such that all insertions happened to populate the table in exactly linear order; this behaviour is not realistic. Instead, we randomly draw samples from a Poisson distribution that has a mean $\lambda = 2$, in order that the mass of the distribution is towards the beginning of the neighbourhood, but may be further on.

2.2.1 Output & Discussion

Table 1 shows the output obtained thus far from the simulator, this shows that even on a coarse reordering policy, we begin to approx a factor of 2 improvement in terms of work performed. Performing fine reordering over the same configuration invariably leads to better results than coarse. A key caveat is that this metric concerns itself only with the hottest element in the table, extensions are underway to take a more wholistic view of potential performance gains. The fact that as $L \to 1$, gains appear to disappear is entirely expected. This is due to the fact that as a table becomes full, it is to be expected that most neighbourhoods will only have one bucket in T, therefore reordering is not possible.

Table 1: Simulator results showing the oracle measurements for the simulator, in terms of percentage of instructions avoided.

Size M	Load L	Skew α	Coarse (%)	Fine (%)
1.00E+03	0.7	1.1	30.59	92.1
1.00E+03	0.7	2.1	31.49	99.53
1.00E+03	0.7	3.1	1.13	3.53
1.00E+03	0.8	1.1	27.86	83.61
1.00E+03	0.8	2.1	44.04	96.3
1.00E+03	0.8	3.1	0.58	2.08
1.00E+03	0.9	1.1	47.5	95.7
1.00E+03	0.9	2.1	48.43	99.04
1.00E+03	0.9	3.1	0.83	1.71
1.00E+03	1	1.1	0.99	2.64
1.00E+03	1	2.1	9.36	14.9
1.00E+03	1	3.1	0.59	1.1
1.00E+05	0.7	1.1	29.86	82.3
1.00E+05	0.7	2.1	35.39	98.92
1.00E+05	0.7	3.1	35.74	95.81
1.00E+05	0.8	1.1	32.82	93.67
1.00E+05	0.8	2.1	33.68	99.64
1.00E+05	0.8	3.1	32.64	95.69
1.00E+05	0.9	1.1	32.97	99.33
1.00E+05	0.9	2.1	35.43	97.79
1.00E+05	0.9	3.1	31.42	97.17
1.00E+05	1	1.1	0.01	0.02
1.00E+05	1	2.1	0.65	1.96
1.00E+05	1	3.1	0.01	0.01
1.00E+07	0.7	1.1	34.16	97.81
1.00E+07	0.7	2.1	34.14	99.2
1.00E+07	0.7	3.1	33.8	97.37
1.00E+07	0.8	1.1	33.03	98.04
1.00E+07	0.8	2.1	32.52	96.91
1.00E+07	0.8	3.1	31.81	94.19
1.00E+07	0.9	1.1	31.87	95.88
1.00E+07	0.9	2.1	26.05	77.39
1.00E+07	0.9	3.1	33.89	99.79
1.00E+07	1	1.1	0	0
1.00E+07	1	2.1	0	0
1.00E+07	1	3.1	0	0

3. METHODS

HopH is used as the basis for the solution. The argument of reordering based on the *hotness* of the data itself given by [1] is highly aligned with the goals for this work. This work differs as the specific objective is achieving spatio-temporal locality, in order to minimise average lookup times. The value of S is selected such that a bucket B_i fits within a cache-line size $C = 64$B. In order that the size of the bucket $|B_i| \leq C$, we choose S, where the size of a tuple $|\tau| = 8 + 8 = 16$B, and μ is the size of any required meta-data, to be $S = \left\lfloor \frac{C-\mu}{|\tau|} \right\rfloor$. Different access patterns are simulated by drawing samples from a Zipfian distribution $\mathcal{Z}$, whose parameter α affects the skew of the samples obtained.

3.1 Reordering Strategies

This section describes a number of re-ordering strategies for the placement of data in a neighbourhood. These methods look at coarse, down to fine tuple-level, and heuristic reordering operations.

3.1.1 Fine - Intra-Bucket (FIntrB)

This method simply sorts the tuples within a bucket B_i (that must be of the same neighbourhood) by hotness. This is performed using a priority queue, inserting the tuples and ordering by their number of accesses, before reinserting them into B_i. As we know that $|B_i| \leq C$, there are no further bucket traversals required, and any potential gains are purely in terms of operations performed within B_i for a `Contains()` or `Get()` and, as such, will be minimal.

3.1.2 Coarse Inter-Bucket (CIB)

We can express the overall hotness of a bucket B_i by the summation of accesses to all tuples contained within it. With this method, we do not care about the order of the tuples within slots, but simply that the most-frequently-hit bucket is closest to the neighbourhood root. A clear compromise of this strategy is that it does not care about the distribution of accesses within B_i; in the example where $S = 3$ and B_i has one element with many accesses, but two without, and B_j has a uniform distribution whose total (summed) access is more than B_i, B_j will be promoted first.

3.1.3 Fine Inter-Bucket (FIB)

By far the most expensive operation, this seeks to redistribute the neighbourhood at a tuple-level using a priority queue to order tuples by hotness, before reinserting them into all buckets within the neighbourhood. The positions of buckets within the neighbourhood do not change. In terms of memory use, this strategy is the most demanding. Potentially, if every possible bucket in a neighbourhood ν_i belongs to that neighbourhood, there will be many elements to copy and reinsert over H buckets. Although the memory traversal will be sequential, this will involve $H-1$ cache-invalidations. The trade-off for this cost is that we are guaranteed to have all tuples in correct order, with none of the compromises of FIB (Fine Inter-Bucket) or CIB (Coarse Inter-Bucket).

3.1.4 Heuristic

The simplest approach is a direct-swap heuristic, which simply compares the current tuple τ_i with the first tuple of the neighbourhood, τ_r, if the τ_i is the hotter of the two. This approach should not have a high computational overhead, as in traversing the neighbourhood to find τ_i, we already stored a reference to τ_r upon first encountering it. If τ_i is hotter than τ_r, swap them and their distribution entries.

3.1.5 Approximate Sorting

This approach exploits the spatial locality afforded by HopH; we are guaranteed that all H buckets within neighbourhood ν_i are in the same, homogeneous region of memory. Once these pages are in the cache-hierarchy, latencies in accessing them are reduced and, depending on the cache layer, very efficient. As the `Get()` or `Contains()` operation traverses ν_i, a Bubblesort-like operation can be applied based on hotness.

3.2 Epochs & Squashing

In order to be adaptive over time, there must be a series of rules that govern how the hotness of data changes over times and accesses. For approaches where there is an absolute trigger for reordering, an epoch is defined at a neighbourhood level, and is its state before a reordering method is invoked.

For non-triggered reordering methods, there must be a continual state of adaptation for the associated meta-data.

Once tuples in ν_i have been rearranged, and the numeric values of tuple accesses have been reduced in some manner, the hottest element in ν_i should remain so (*preservation of hotness*). For epoch-based approaches, this means the skew and shape of the distribution should be roughly equal after an epoch, but smaller in magnitude. For non-epoch-based approaches, the distribution should be more fluid, dynamically changing all values in ν_i regularly. Those tuples in ν_i that have not had many accesses should have historic access counts reduced, until sufficient epochs have passed that they no longer have any hotness in epoch-based reordering (*cooling*).

3.3 Deletions

Handling deletions in HotH (Hotscotch Hashing) should follow two principles further to a baseline HopH deletion:

1. For per-slot level meta-data, the distribution entry (number of reads) for the tuple to be deleted should also be deleted from any per-bucket counter if present, before being erased itself.

2. If any slot that is not the first slot of a bucket contains the tuple to be deleted, subsequent slots should be shift towards the first slot, to minimise unnecessary traversal gaps in the bucket structure.

After this, should the bucket be empty after tuple deletion, the standard HopH process is followed.

3.4 Invocation

Finding the balance between the severity of the reordering strategy employed, and how and when to trigger it are key points in this work. We explore a number of invocation strategies, and the various forms of meta-data required to permit them.

3.4.1 Trigger-based

The simplest method of invocation is that of setting a minimum number of accesses to a bucket or neighbourhood, dependant upon the reordering strategy used, where a counter $\rho = 0$ is meta-data within the bucket structure. Upon every `Get()` or `Contains()` call, ρ is incremented and evaluated. Once ρ exceeds some instantiation-defined value σ, the reordering is invoked.

3.4.2 Probabilistic-based

Instead of storing any meta-data at a bucket/neighbourhood level, we can simply state that with some probability $\pi_r = \mathbf{P}(\text{Perform Redistribution})$, we will perform a reordering. A key point in using this method is to avoid paying the cost of the given reordering strategy often; however the generation and evaluation of PRN (Pseudo Random Number)s is itself a non-zero cost. Conclusions drawn from experimentation using a number of PRNG (Pseudo Random Number Generator)s show that the `xorshift` a good candidate. Further work is looking into simpler masking / shifting of an integer value, as statistically-sound randomness isn't mandatory.

4. EXPERIMENTS

1. The base experiment looks at the Avg.CPU (Average CPU Cycles) metric of creating tables based on the same input data, and performing the same set of queries, in the same order, amongst different table configurations and comparing their results. Input data is randomly generated, and HopH is used as a baseline.

2. A SHJ (Symmetric Hash Join) is constructed of two tables. A HotH configuration is compared with one backed by HopH, to measure the difference the adaptive approach we propose makes to realistic scenario. Experiments evaluating SHJ performance will use existing data-sets, over different relation sizes.

5. DISCUSSION

Implementations of all methods described have been complete, with results expected soon. As the overarching theme of this thesis is NVM, work will be conducted to exploit its characteristics to aid HotH. An approach by [4] provides a way for leaf nodes being stored in a DRAM/NVM hybrid B+Tree, where the system prefers `Contains()` operations to `Get()`, as keys are stored in quick DRAM and values in NVM. Following a similar methodology for hybrid NVM / DRAM should provide a fast key lookup for `Contains()`, as we can now fit many keys within C, but also gives slower reordering due to the extra cost of moving around NVM vs. DRAM. Unlike the problem solved by [4], we must still ensure spatial and temporal locality. Intuitively, this could mean analog structures of M elements in DRAM and NVM with a translation function $t(i, s) \rightarrow j$ that takes the bucket and slot numbers i, s respectively, and maps it to a position (offset) j in the NVM table.

6. ACKNOWLEDGMENTS

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

7. REFERENCES

[1] S. Albers and M. Karpinski. Randomized splay trees: Theoretical and experimental results. *Information Processing Letters*, 81(4):213–221, 2002.

[2] M. Herlihy. Hopscotch Hashing. pages 0–15.

[3] S. Idreos, M. Kersten, and S. Manegold. Database Cracking. *CIDR '07: 3rd Biennial Conference on Innovative Data Systems Research*, pages 68–78, 2007.

[4] I. Oukid, J. Lasperas, A. Nica, T. Willhalm, and W. Lehner. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*, pages 371–386, 2016.

[5] R. Pagh and F. F. Rodler. Cuckoo Hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[6] D. D. Sleator and R. E. Tarjan. Self-adjusting Binary Search Trees. *Journal of the ACM*, 32(3):652–686, 1985.

Graph Pattern Mining for Business Decision Support

André Petermann
supervised by Erhard Rahm
Database Research Group
University of Leipzig
petermann@informatik.uni-leipzig.de

ABSTRACT

To which extent can graph pattern mining enrich business intelligence? This question was the seed whose sprout became my PhD research. To find an answer, I investigated graph-based data integration, the calculation of business measures from graphs and suitable data mining techniques based thereon. The latter should identify correlations between occurrences of specific graph patterns and values of business measures. Finally, interesting patterns should be presented to decision makers. With real world applications in mind, I additionally considered the requirements of big data scenarios at all stages. In this paper, I summarize my recent contributions and give an outlook on the work required to finally answer the motivating question.

1. INTRODUCTION

To make good decisions, enterprises have a permanent desire to understand the reasons for certain values of business measures. In a classical business intelligence development lifecycle a domain expert is choosing potential impact factors and the analytical model is tailored to evaluate measures by these factors. However, this approach often leads to oversimplified models and, thus, unexpected patterns may remain hidden. Hence, the use of graph models for business intelligence is a promising approach for two reasons: First, some patterns are too complex to be represented using tuples. In particular, this applies to patterns where most of the information is about relationships.

Second, graphs can loosen the coupling of experts' bias and analytical results because data represented by rich graph models like the property graph model [20] allows not only to evaluate instance data but also metadata occurrence, i.e., schema-related information is part of the result and must no be specified in a query. For example, to reveal patterns between objects of classes A and B, ideally analysts just want to ask *"Which patterns typically connect As and Bs?"* and expect an answer like *"Mostly via a sequence of Cs and Ds, but sometimes only via Es"*. In contrast, using a structured model like common data warehouse models, they need to ask several questions like *"Are As and Bs frequently connected via Ds?"* and get simple *"Yes"* or *"No"* answers.

Summarized, wrapping the schemas of data sources into a graph super-model enables more generic queries and mining of self-descriptive patterns. In my PhD research, I developed the BIIIG approach (**B**usiness **I**ntelligence with **I**ntegrated **I**nstance **G**raphs) to enable such flexible graph-based analyses of business data. Figure 1 provides an overview of the approach. In the remainder of this paper, I will give a brief overview of my past and future work.

2. CONTRIBUTIONS

In the following, I will provide an overview of the contributions made during my past PhD research.

2.1 Graph Representation of Business Objects

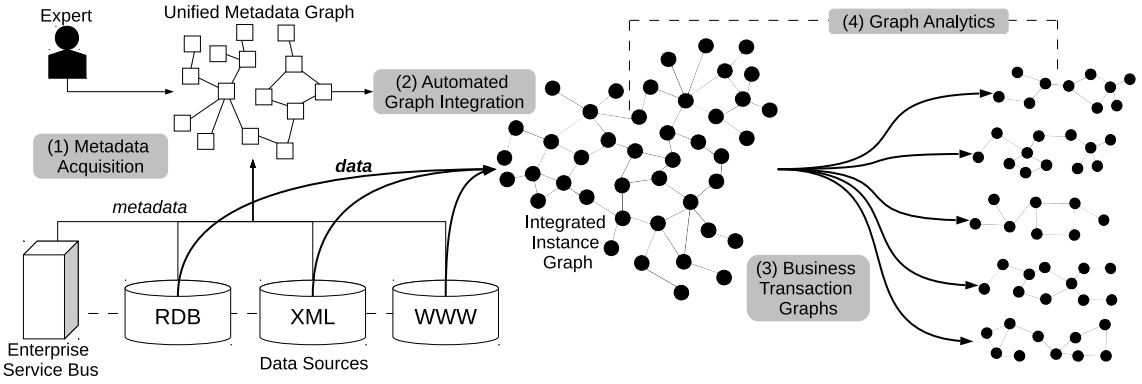
Business data of a company implicitly describes a graph but is typically stored in one or more business information systems based on relational databases. Thus, I first had to consider the process of turning data organized in tables into graphs. In [16], I proposed a semi-automated solution to this problem and implemented a prototype based on a productive graph database [19]. The approach was evaluated using real and synthetic data. In the following, I will briefly discuss my approach to graph-based data transformation and integration but, due to limited space, only for relational databases.

In the initial step (step 1 of Figure 1) metadata of one or more data sources is acquired, stored in a graph model (*unified metadata graph*) and enriched by a domain expert. In this graph, every vertex represents a class of domain objects (class-like tables) and every edge an association between classes (foreign keys or m:n tables). Both, vertices and edges, further contain information about their source system, semantic type, keys and attributes.

In the second step (step 2 of Figure 1), vertices and edges of the metadata graph are interpreted to generate SQL statements. These are used to query instances (data objects and relationships) from the source databases. Afterwards, all data objects are transformed into vertices and all relationships into edges of a so-called *integrated instance graph*. I decided to use the property graph model [20], i.e., a directed labeled multigraph with named attributes (properties). For both, vertices and edges, labels represent their semantic type and all attributes are stored using properties. Another popular model to represent such graphs is the resource description framework (RDF) [10]. However, RDF is

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

Figure 1: Overview of the BIIIG approach [16]



more general and provides no dedicated structures for logical relationships, labels and properties. In consequence every attributed relationship must be represented by a subgraph [7] and the total number of edges would be much higher.

Besides model transformation, the second step may also include data integration. Every vertex has a globally unique *source identifier* composed from identifiers for source system, class and record. Thus, the approach supports relationships across data sources. Such relationships may exist for two reasons: First, data objects of different systems may reference each other, for example, a ticket of a customer issue tracking system may reference an invoice stored in an accounting system. Second, certain master data is held redundantly and copies refer to a global business key (e.g., customer number). For the latter case, I proposed *vertex fusion*, a strategy to automatically merge the resulting vertices and to redirect their relationships.

2.2 Business Transaction Graphs

Data warehouse models use a schema (e.g., star schema) that needs to be defined in advance to link facts and dimensions. Data mining techniques based thereon can evaluate the co-occurrence of certain dimensional values (e.g., feature vectors). The major aim of the BIIIG approach was to enable an additional evaluation of the relationship structure among interrelated facts as well as between facts and dimensions. Analyzing such structural patters is promising, for example, to reveal interaction patterns between customers and certain employees that lead to high sales profit. Here, the first challenge was to find a suitable abstraction to enable such analyses. For this reason, I introduced the concept of *business transaction graphs* [16] as the base for measure aggregation (Section 2.3) and graph pattern mining (Section 2.5). A business transaction graph represents, for example, a single execution of a business process like trading or manufacturing goods.

I proposed a domain-specific algorithm to automatically extract a collection of such graphs from the integrated instance graph (step 3 in Figure 1). Figure 2 shows four example business transaction graphs of a sales process. For sake of ease, edge types are omitted. The algorithm is based on the observation that *transactional data* (e.g., `Email`, `Quotation`, `SalesOrder`) only link each other in the case of a causal connection. Here, *causally connected* means object B (e.g., an invoice) would not exist without the prior existence of object A (e.g., a quotation). Thus, the algorithm first identifies connected components of transactional data and, afterwards, adds all *master data* (e.g., `Customer`, `Employee`, `Product`) that is directly connected to one of the component's vertices. In consequence, every transactional vertex belongs to exactly one graph while master data instances may be part of many graphs. The algorithm's only requirement is the categorization of vertices to represent either master or transactional data. This categorization is done by a domain expert at the class level and taken over by their instances.

Due to the bad availability of datasets from real business information systems, I designed and implemented FoodBroker [17], a data generator based on business process simulation. The generated data's schema is inspired by real business information systems. Further on, every master data object has a quality criterion and will, if participating, influence the process execution positively or negatively. For example, the more poor master data objects interact in a process the higher is the chance for a bad process outcome like financial loss. Thus, data generated by FoodBroker is suitable to evaluate the BIIIG approach.

2.3 Business Measure Aggregation

To analyze graph collections, first, measures need to be calculated on the graph-level. For this reason, I proposed the *graph aggregation* operation [14, 16]. Aggregation derives a scalar value from an input graph's vertices and edges including labels and properties, e.g., to count contained vertices of a certain type or to sum all values of a specified property. The actual calculation is specified by a user-defined function γ that is executed for every graph of a collection. For example, the attributes `isClosed` and `soCount` attached to the graphs of Figure 2 represent the results of two different aggregation functions $\gamma_{\texttt{isClosed}}$ and $\gamma_{\texttt{soCount}}$. While $\gamma_{\texttt{soCount}}$ counts vertices of type `SalesOrder`, $\gamma_{\texttt{isClosed}}$ will check, if the graph contains a closed sales quotation, i.e., if the sales process is finished. The result of an aggregation function can be used to filter a graph collection. In our example, only graphs with $\gamma_{\texttt{isClosed}} = true$ were selected to apply $\gamma_{\texttt{soCount}}$. Since vertices of type `SalesOrder` only exist in the case of a confirmed (won) `Quotation`, this aggregation result can be used to categorize graphs into won ($\gamma_{\texttt{soCount}} > 0$) and lost ($\gamma_{\texttt{soCount}} = 0$) ones.

2.4 Scalable Frequent Subgraph Mining

To find correlations between certain business measures values and graph patterns, pattern frequencies need to be computed. This primitive operation is the well known problem of frequent subgraph mining [5]. Since the problem is NP-complete and graph collections in business applications can be very large I required a massive parallel solution to minimize total response times. There are three distributed

approaches to (exact and complete) frequent subgraph mining based on MapReduce [4, 11, 12]. However, none of these approaches is capable to mine directed multigraphs.

Thus, I discussed an extension of the popular gSpan algorithm [22] to support directed multigraphs in [15] and proposed DIMSpan [18], the first approach to frequent subgraph mining based on distributed in-memory dataflow systems like Apache Spark [23] or Apache Flink [2]. In comparison to the existing MapReduce based approaches, DIMSpan not only requires fewer disk access but also shuffles less data over the network and can reduce the total number of expensive isomorphism resolutions to a minimum. In experimental evaluations I have shown that a lightweight data structure as well as effective and fast compression techniques based thereon are key techniques for good scalability in big data scenarios. Figure 3 shows example evaluation results of DIMSpan. The chart on the left hand side shows a perfect scalability for increasing input data volume, since computation time for a portion of 100K graphs is decreasing for a growing number of graphs at different minimum support thresholds s_{min}. The chart on the right hand side shows good speedup for an increasing cluster size.

2.5 Category Characteristic Patterns

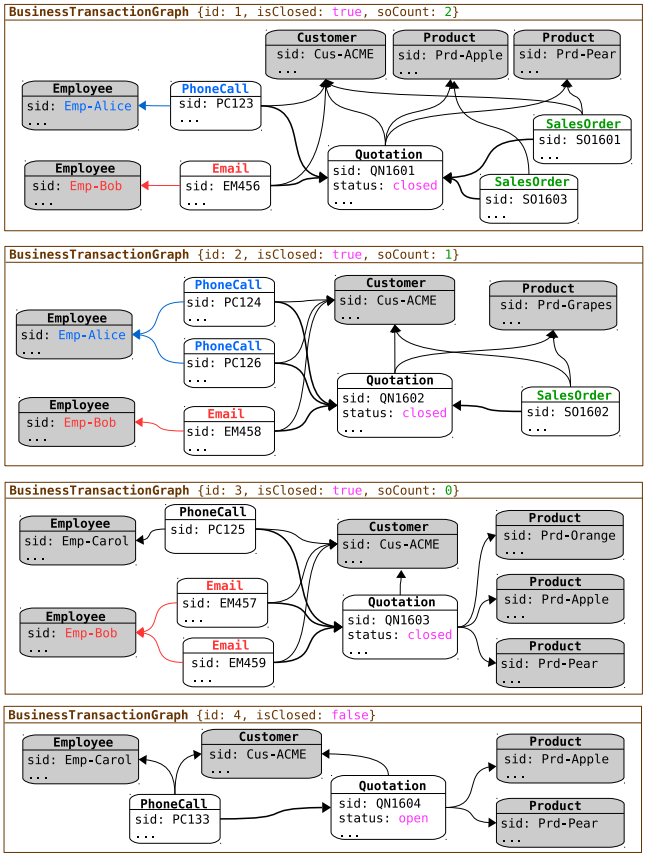
Since we are able to categorize graphs based on aggregated measures and can compute pattern frequencies, we can also mine correlations between categories and certain patterns. In [14] I proposed an analytical workflow to identify such *category characteristic patterns*. Figure 2 shows four example graphs where the top 3 represent finished executions of a sales process ($\gamma_{\texttt{isClosed}} = true$) categorized into won ($\gamma_{\texttt{soCount}} > 0$) and lost ones ($\gamma_{\texttt{soCount}} = 0$). Blue and red color are used to highlight example patterns. The pattern in blue color represents *'a phone call made by Alice'* and the one in red color *'an email sent by Bob'*. To enable the extraction of patterns combining labels and values of certain properties I additionally use a specific transformation between categorization and mining.

In contrast to basic frequent subgraph mining, I require patterns not just to be frequent but to be characteristic for a measure category. For example, the blue pattern is interesting, as it occurs in all of the won cases but not in the lost one. By contrast, the red pattern occurs in all graphs of both categories and, thus, is considered to be trivial. Therefore, I use an *interestingness measure* comparing a pattern's frequency in different categories. The measure is a function that evaluates the relative support of a pattern within a category in relation to its average relative support in all categories. Based on this measure, the analyst sets an *interestingness threshold* to prune patterns by minimum interestingness. Additionally, there is a *candidate threshold* to specify the minimum support of a pattern inside a category to be considered as a candidate. This parameter is used to save computations in exchange for result completeness.

2.6 Framework Integration

The implementation of the initial prototype [19] only covered data integration and simple analytical queries. To find a suitable platform for complex workflows including measure calculation and pattern mining, I performed an in-depth comparison of graph databases [7] and examined the suitability of different graph processing technologies [14]. I found out that none of the existing systems could satisfy my

Figure 2: Example Business Transaction Graphs [14]



requirements, especially they miss support for graph collections and graph properties. Thus, I joined the development of Gradoop [9], a scalable framework supporting complex workflows [15] of multiple operations on both graphs and graph collections.

The aggregation and vertex fusion operators proposed in [16] became part of Gradoop's *extended property graph model*. Additionally, DIMSpan [18] as well as the algorithms to extract business transaction graphs [16] and the one to identify category characteristic patterns [14] were implemented to fit a dedicated interface for plug-in algorithms and are part of Gradoop's open source repository[1]. Besides operators related to the BIIIG approach, the framework provides further valuable analytical operators such as *graph grouping* [8] and *graph pattern matching* [6].

3. PROBLEMS AND FUTURE WORK

In first evaluations of mining characteristic patterns from FoodBroker data I found out that the expected result was returned but the number of patterns quickly became very large and may overwhelm analysts. However, I was already able to identify two particular "data science" problems and their potential solutions: First, the method described in Section 2.5 eliminates trivial patterns for each category but not combinations of trivial and characteristic patterns. Thus, I'll investigate ranking results using a fast analytical method of graph p-value calculation [13]. Based thereon most significant patterns should be presented first.

[1] `www.gradoop.com`

Second, patterns should contain different levels of dimensional attributes. To provide a simple example, on the one hand an analyst won't be interested in the pattern *bread* and *butter*, if there are more specific patterns like *wholegrain bread* and *butter*. On the other hand, if *bread* and *butter* is not returned, the more general pattern of *bakery products* and *butter* could be. Thus, I will extend the DIMSpan algorithm to mine dimensional attributes across multiple levels. This approach has already been studied for itemsets [3] but not for graphs.

Finally, I will evaluate BIIIG in a real world scenario in cooperation with a large-scale enterprise. The evaluation will be based on GRADOOP and cover all steps of my approach. The company will not only provide real business data but also valuate analytical results and scalability.

4. SUMMARY

In my past PhD research, I contributed to the fields of graph data management and graph data mining. In contrast to other graph-based approaches to business intelligence [1, 21], BIIIG covers all steps from data integration to analytical results and requires no advance definition of an analytical schema. To the best of my knowledge, I proposed the first approach to integrate data from multiple source into a single instance graph and the first one using metadata-driven automation. Further on, I was the first who discussed the usage of graph collections to analyze the structure of interrelated business objects and to enable novel data mining techniques based thereon. Additionally, I presented the first horizontally scalable approach to transactional frequent subgraph mining using a distributed in-memory dataflow system and the first supporting directed multigraphs. To finish my PhD research, I will improve applicability by returning cross-level results and ranking them by significance.

5. ACKNOWLEDGMENTS

This work is partially funded within the EU program *Europa fördert Sachsen* of the European Social Fund and by the German Federal Ministry of Education and Research under project ScaDS Dresden/Leipzig (BMBF 01IS14014B).

6. REFERENCES

[1] D. Bleco and Y. Kotidis. Business intelligence on complex graph data. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 13–20. ACM, 2012.

[2] P. Carbone et al. Apache flink: Stream and batch processing in a single engine. *Data Eng.*, 38(4), 2015.

[3] T. Eavis and X. Zheng. Multi-level frequent pattern mining. In *International Conference on Database Systems for Advanced Applications*, pages 369–383. Springer, 2009.

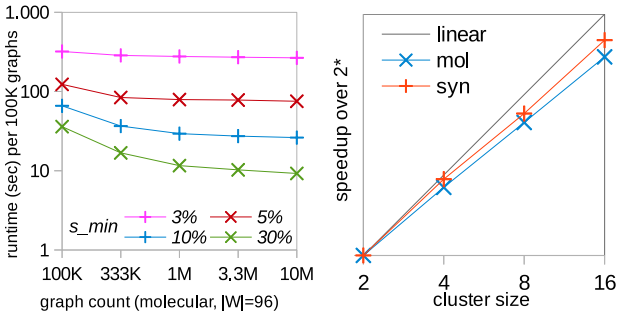
[4] S. Hill et al. An iterative mapreduce approach to frequent subgraph mining in biological datasets. In *Conference on Bioinformatics, Computational Biology and Biomedicine*, pages 661–666. ACM, 2012.

[5] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(01):75–105, 2013.

[6] M. Junghanns, M. Kieling, A. Averbuch, A. Petermann, and E. Rahm. Cypher-based graph pattern matching in gradoop. In *Proc. 5th Int. Workshop on Graph Data Management Experiences and Systems*. ACM, 2017.

[7] M. Junghanns, A. Petermann, N. Neumann, and E. Rahm. Management and analysis of big graph data: Current systems and open challenges. *Big Data Handbook, Springer*, 2017.

Figure 3: DIMSpan: Scalability for increasing data volume (left) and cluster size (right) [18]



[8] M. Junghanns, A. Petermann, and E. Rahm. Distributed grouping of property graphs with gradoop. In *17th Conference on Database Systems for Business, Technology, and Web (BTW)*, 2017.

[9] M. Junghanns, A. Petermann, N. Teichmann, K. Gómez, and E. Rahm. Analyzing extended property graphs with apache flink. In *1st SIGMOD Workshop on Network Data Analytics*, page 3. ACM, 2016.

[10] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. 2006.

[11] W. Lin, X. Xiao, and G. Ghinita. Large-scale frequent subgraph mining in mapreduce. In *Int. Conf. on Data Engineering (ICDE)*, pages 844–855. IEEE, 2014.

[12] W. Lu, G. Chen, A. Tung, and F. Zhao. Efficiently extracting frequent subgraphs using mapreduce. In *Int. Conf. on Big Data*, pages 639–647. IEEE, 2013.

[13] G. Micale, R. Giugno, A. Ferro, M. Mongioví, D. Shasha, and A. Pulvirenti. Fast analytical methods for finding significant colored graph motifs. *To be published in Data Mining and Knowledge Discovery*, 2017.

[14] A. Petermann and M. Junghanns. Scalable business intelligence with graph collections. *it-Information Technology*, 58(4):166–175, 2016.

[15] A. Petermann, M. Junghanns, S. Kemper, K. Gómez, N. Teichmann, and E. Rahm. Graph mining for complex data analytics. In *Int. Conf. on Data Mining Workshops (ICDMW)*, pages 1316–1319. IEEE, 2016.

[16] A. Petermann, M. Junghanns, R. Müller, and E. Rahm. BIIIG: Enabling Business Intelligence with Integrated Instance Graphs. In *Int. Conf. on Data Engineering Workshops (ICDEW)*, pages 4–11. IEEE, 2014.

[17] A. Petermann, M. Junghanns, R. Müller, and E. Rahm. Foodbroker-generating synthetic datasets for graph-based business analytics. In *Workshop on Big Data Benchmarks*, pages 145–155. Springer, 2014.

[18] A. Petermann, M. Junghanns, and E. Rahm. Dimspan-transactional frequent subgraph mining with distributed in-memory dataflow systems. *arXiv preprint arXiv:1703.01910*, 2017.

[19] A. Petermann et al. Graph-based Data Integration and Business Intelligence with BIIIG. *PVLDB*, 7(13), 2014.

[20] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[21] Z. Wang et al. Pagrol: Parallel graph olap over large-scale attributed graphs. In *30th Int. Conf. on Data Engineering (ICDE)*, pages 496–507. IEEE, 2014.

[22] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *International Conference on Data Mining (ICDM)*, pages 721–724. IEEE, 2002.

[23] M. Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2, 2012.

Efficient Clustering for Large-Scale, Sparse, Discrete Data with Low Fundamental Resolution

Veronika Strnadová-Neeley,
supervised by John R. Gilbert
University of California, Santa Barbara
veronika@cs.ucsb.edu

ABSTRACT

Scalable algorithm design has become central in the era of large-scale data analysis. My contribution to this line of research is the design of new algorithms for scalable clustering and data reduction, by exploiting inherent low-dimensional structure in the input data to overcome the challenges of significant amounts of missing entries. I demonstrate that, by focusing on a property of the data that we call its *fundamental resolution*, we can improve the efficiency of clustering methods on sparse, discrete-valued data sets.

1. INTRODUCTION AND BACKGROUND

The necessity for efficient algorithms in large-scale data analysis has become clear in recent years, as unprecedented scaling of information has sprung up in a variety of domains, from bioinformatics to social networks to signal processing. In many cases, it is no longer sufficient to use even quadratic-time algorithms for such data, and much of recent research has focused on developing efficient methods to analyze vast amounts of information.

Here we focus on scalable clustering algorithms, a form of unsupervised learning that is invaluable in exploratory data analysis [16]. Many successes in the effort to design these algorithms have focused on leveraging an inherent structure in the data, and its structure may be best expressed in various ways. The data may be best described as lying in an inherently low-dimensional Euclidean space, along a low-dimensional manifold, or it may have certain self-repeating, or *fractal* properties. All these structural properties have been explored to some degree in order to design more efficient clustering algorithms.[11, 2, 6, 7, 8, 17]

My work focuses on leveraging a property of large-scale, discrete-valued data that I call its *fundamental resolution*, a concept that can be explained by comparing the images in Figure 1. More pixels produce a clearer image, but only up to a point – we cannot distinguish the leftmost image from that in the middle, even though more pixels are used to render the image in the leftmost position. Similarly, in many large, discrete-valued data sets, the fundamental resolution, rather than the number of data points, determines the extent to which we can distinguish points from one another before the data becomes redundant. If we know a large data set with many missing values has a fundamental resolution, we can more easily single out data points that are noise and fill in missing entries.

Figure 1: Fundamental resolution of an image: the more pixels we use, the clearer it gets, but only up to a point.

In the following, I present efficient algorithms for clustering large-scale, discrete-valued data sets with missing values by leveraging the fundamental resolution of the data. Previously, my collaborators and I have demonstrated that the underlying fundamental resolution of binary-valued genetic mapping data can be used to quickly cluster large genetic mapping data sets. I am now generalizing this clustering approach to large-scale, discrete-valued data, such as that found in the recommender systems domain. Genetic mapping and recommender systems present similar challenges to clustering algorithms, due to the large degree of sparsity and the sheer scale of the input data in these domains.

2. RELATED WORK

Much attention has been paid to the intuition that many large-scale data sets lie in an inherently low-dimensional space, which explains the popularity of matrix factorization methods for large scale data analysis. Methods such as principal component analysis rely on an SVD decomposition in order to project a high-dimensional data set into a lower dimensional space [10, 9]. Spectral clustering is another such example, and has been modified in recent years to improve in running time [11]. More recently, the CUR decomposition [12] has gained popularity as a sparse matrix factorization method that is both fast and in some cases more interpretable than a decomposition based on eigenvectors. With matrix factorization approaches, clustering the projected data in the lower-dimensional space often results in better clustering performance. However, my work focuses on data that does not necessarily lie in a low-dimensional Euclidean subspace – many dimensions in the input may be relevant in data with a low fundamental resolution. In addition, there is no clear answer on how to deal with noise and missing entries when factorizing a large data matrix, whereas my work takes these issues into account.

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

Other forms of lower-dimensional inherent structure have also been explored to speed up the clustering of large-scale data. A data set's fractal dimension has been exploited for clustering [8], but this method is not scalable to large data sets. An approach based on exploiting a low fractal dimension and entropy of a data set has been successfully applied to quickly search massive biological data sets.[17] However, here we focus here on efficient clustering, not efficient search.

Older, popular methods such as the well-known DBSCAN[6], algorithm seek to preserve the shape of data, but rely on the input lying in a metric space. In addition, these methods typically require at least quadratic time when the input data lies in three or more dimensions[7], and again do not account for missing values. Popular nonlinear dimensionality reduction methods, such as Laplacian eigenmaps[2], also don't account for missing data and noise, and many such approaches do not scale well.

3. EXPLOITING THE FUNDAMENTAL RESOLUTION OF GENETIC MAP DATA

Genetic map data for a homozygous *mapping population* can be represented as a binary matrix X, composed of rows x_u, where each entry x_{ui} can take on one of two values a or b, or it can be missing. [4] In this application domain, errors occur when an entry was erroneously recorded during sequencing – that is, it was flipped from a to b or from b to a – and errors typically occur at a fixed rate ϵ. The goal of genetic mapping is to produce a map of the genome, which shows the correct clustering and ordering of the input x_u. Such maps have applications in health, agriculture, and the study of biodiversity.

Producing a genetic map typically requires three stages: 1) Clustering the vectors x_u into linkage groups, 2) Ordering the vectors within each linkage group, and 3) Determining the correct genetic distance between the ordered vectors. In previous work ([13], [14]), we showed that by exploiting the fundamental resolution of genetic map data, we can quickly and accurately cluster the input vectors and reduce them from a large-scale, noisy, and incomplete data set into a small set of *bins* that more accurately represent the genome.

3.1 Scalable Clustering

We have shown that, using the well-known *LOD score* similarity that is ubiquitous in genetics, we can design a fast and accurate algorithm to cluster input data for genetic mapping [13]. The LOD score is a logarithm of odds, comparing the likelihood of two vectors being in the same cluster to the likelihood that they were generated by chance:

$$\mathrm{LOD}(x_u, x_v) = \log_{10} \frac{\mathrm{P(data}|x_u \text{ and } x_v \text{ in same linkage group)}}{\mathrm{P(data}|\text{ pure chance)}}$$

Because we have binary data, the denominator in the LOD fraction is simply $(\frac{1}{2})^\eta$, where η is the number of entries that are non-missing in both x_u and x_v. For example, if $x_u = [\ a\ \ b\ \ b\ \ -\ \ b\ ]$ and $x_v = [\ a\ \ a\ \ -\ \ -\ \ a\ ]$, then the denominator is $(\frac{1}{2})^3$, because the first, second, and last entries are non-missing entries in both x_u and x_v. The numerator is a function of the estimated *recombination fraction* of the genetic data, and is explained in more detail in our previous work [13]. The LOD score does not obey the triangle inequality, which together with the presence of errors (flipped entries) and missing values, eliminates the possibility of accurately clustering the data with existing efficient algorithms.

Dataset	Input Size	BubbleCluster	
		F-score	Time
Barley	64K	0.9993	15 sec
Switchgrass	113K	0.9745	8.9 min
Switchgrass	548K	0.9894	1.9 hrs
Wheat	1.582M	N/A	1.22 hrs

Table 1: Clustering performance on Barley, Switchgrass, and Wheat from the Joint Genome Institute using BubbleCluster. State-of-the-art mapping tools are unable to cluster data sets at this scale. (Table originally appeared in [13])

Our algorithm BubbleCluster, which resembles the DBSCAN method [6], utilizes the LOD score to efficiently cluster the data. First, we build a sketch of the clustering by linking together points that exceed a high LOD threshold, which only occurs for vectors with many matching binary values. Then, points with more missing values are linked to the point in the skeleton attaining the highest LOD score. The fundamental resolution limits the number of unique input vectors and thus as the data size grows, it is more likely that enough high-quality points exist to build the skeleton and accurately place the remaining points.

BubbleCluster allows for efficient clustering of genetic map input data into linkage groups. As Table 1 shows, our method achieved both high precision and recall (expressed as the F-score) on real genetic data. It was also the first method to successfully cluster genetic map data at large scales, including the grand challenge hexaploid bread wheat genome [3], and outperformed state-of-the-art mapping tools in terms of clustering performance on simulated data [13].

To further aide the efficiency and accuracy of the genetic mapping process, we introduced a fast data reduction method that quickly converts the large-scale, noisy, and incomplete input data into a small-scale, more accurate and more complete set of points which more clearly represent the genetic map. I will next describe this data reduction method, based on the fundamental resolution of the genetic mapping input data.

3.2 Efficient Data Reduction

Genetic map data has a fundamental resolution that is linear in the dimensionality of the input vectors. We can leverage this property of the data to efficiently reduce it to a much smaller and more accurate set of vectors we call *bins*, that represent positions along the genetic map as illustrated in Figure 2. The data reduction process uses a recursive bisection method to quickly reduce the input vectors within each linkage group to bins [14].

The binary nature of the data limits the number of possible unique input points to 2^n, where n is the dimensionality of the data. However, the fundamental resolution of the data limits this number much further – the fundamental resolution of a genetic map is equivalent to the number of possible unique positions on the map. With a homozygous mapping population (binary data), this number is $O(kn)$, where k is the number of linkage groups (clusters) [14]. Therefore, when the number of input points is much larger than the fundamental resolution, many points must be identical, helping us filter out errors. Furthermore, the large data set size allows us to fill in missing data if we can cluster together points that belong to the same unique map position.

If we know two points belong in the same position on the

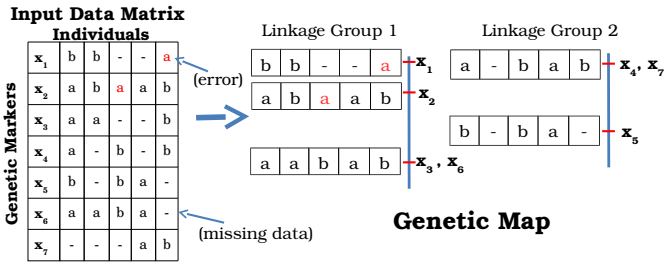


Figure 2: The fundamental resolution of genetic map data allows us to reduce the large-scale, incomplete and noisy input to a more complete set of representative vectors that more clearly describe positions along the genetic map.

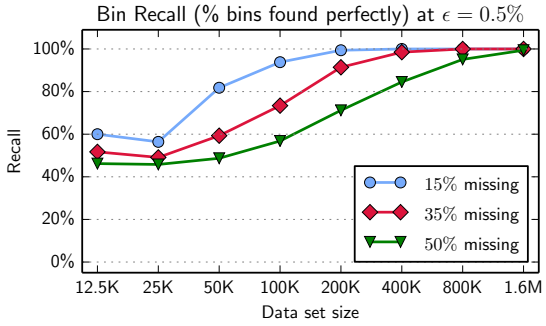


Figure 3: As the data size grows, more map position vectors (bins) are recovered perfectly, while lower missing rates allow our algorithm to recover the bins with less data.

genetic map, we can infer which values are errors and what the missing data should be. In Figure 2, for example, x_3 and x_6 belong to the same position, so we can fill in the missing data in both x_3 and x_6 based on each other's values. A similar idea applies to errors – the more points belong to the same position, the more clear it becomes which values are errors, as long as ϵ is fairly low.

We designed an algorithm that uses recursive bisection to quickly clusters together points in the same genetic map position. At each step, we use a maximum a posteriori (MAP) estimate of ϵ in order to find the best dimension along which to split the point set. The algorithm returns both an estimated error rate and a set of bins that represent unique positions on the genetic map. We showed that the number of bins and the error rate is consistent with existing real-world maps of wheat and barley[14].

We also simulated genetic map data with realistic error rates and a variety of missing data rates. Our algorithm scaled linearly with data set size for all tested missing rates and error rates in synthetic data. As Figure 3 shows, the bin *recall*, or fraction of bins we can recover perfectly, improves at each missing rate with more data. Note that although an error rate of 0.5% seems low, it is actually much higher than encountered in practice.

4. GENERALIZING TO DISCRETE-VALUED DATA WITH MISSING VALUES

Next, I will describe the last portion of my thesis work, clustering large discrete-valued data sets with missing values. One example of such data is found in the Recommender Systems (RS) domain, and much of the experimentation of these clustering methods will be on RS data.

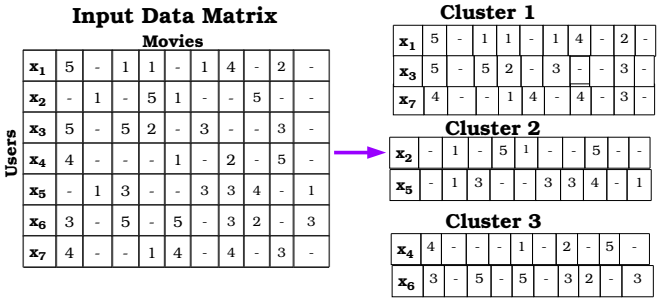


Figure 4: Fundamental resolution in Recommender Systems data equates to finding the vectors that best represent user sub-groups with low disagreement rates.

My hypothesis is that a fundamental resolution exists in many discrete-valued data sets, and can be exploited to efficiently cluster these data at large scales. In the RS domain, we often have a discrete-valued input matrix with many missing values as shown in Figure 4, where an entry X_{ui} represents the rating that user u gave to item i, with ratings typically taking values on a discrete scale. The methods for clustering and reducing the binary genetic map data can be adapted to this more general setting.

The fundamental resolution in RS data can be expressed as the number of unique user sub-groups that rate items very similarly. Recently, Christakopoulou et al.[5]have shown that utilizing user clusters to improve prediction of rating values and recommend better items is an extremely effective approach. An efficient, accurate clustering method for RS data has the potential to enhance such approaches to rating prediction as well as the *top-n* recommendation problem [1].

4.1 The LiRA Similarity Score for Recommender Systems

We have developed a statistical score analogous to the LOD score for RS data called *LiRa*, based on a likelihood ratio. We assume that RS data has a fundamental resolution, and thus users (rows of the input matrix) can be clustered into groups with vectors representing the rating trends of each group. LiRa compares the likelihood of observing the values in two user vectors x_u and x_v assuming the users are in the same cluster, to the likelihood of observing the same data by chance, based on differences in their rating values:

$$\text{LiRa}(x_u, x_v) = \log_{10} \frac{p(\text{differences in } x_u \text{ and } x_v | \text{ same cluster })}{p(\text{differences in } x_u \text{ and } x_v | \text{ pure chance})} \quad (1)$$

LiRa generalizes the LOD score by assuming that differences in two discrete-valued vectors from the same cluster follow a particular multinomial distribution, which is used to compute the numerator. The LiRa score is useful in the RS setting because it leverages more data to make a more accurate judgment of similarity.

We have shown that using the LiRa score to find nearest neighbors in a k-nearest neighbors approach outperforms the popular and widely used Pearson and Cosine similarity scores in terms of rating prediction error [15]. I am currently expanding on the clustering model used to compute the likelihood of users belonging to the same cluster in the LiRa score, which will be useful for efficient data reduction in the discrete-value setting.

4.2 Generalizing Efficient Data Reduction to the Discrete-Valued Domain

As noted previously, my goal is to generalize my previous work on efficient clustering and data reduction in genetic map data to the more general setting of large-scale, discrete-valued data with missing values and noise. The LiRa score from section 4.1 is the first step in this direction, and can be used to produce an initial clustering of the input using a thresholding scheme similar to the BubbleCluster algorithm. The threshold LiRa score within which points will belong to the same cluster will rely on the clustering model used to represent the data. For RS data, a working model is already presented in previous work [15].

After an initial fast clustering, I hope to generalize the data reduction stage to discrete-valued data also. Here, future work involves more precisely defining the point at which the fundamental resolution has been reached. In RS data, the idea is to cluster together users who have very similar rating patterns. One possibility is to only cluster together users if the distribution of their rating differences follows a clustering model. For example, in Cluster 1 in Figure 4, only one pair of ratings for the same item differs significantly: $x_{13} = 1$ and $x_{33} = 5$, giving a rating difference of 4. The remaining ratings are all close together. The recursive bisection method for data reduction in genetic mapping can be modified to this more general case, by dividing user clusters with large differences in rating values, based on MAP estimates of the proportion of each rating difference.

I am currently formalizing the notion of fundamental resolution in the general case, and experimenting with the best clustering model for RS data. As the final piece to my thesis, I hope to demonstrate that the efficient clustering and data reduction methods can be applied to more general data sets, and will be useful in the RS domain for rating prediction.

5. CONCLUSION

I have shown that the concept of *fundamental resolution* can be exploited to design efficient and accurate clustering algorithms in the genetic mapping domain, and I am extending this concept to the more general setting of discrete-valued data. The methods presented here are useful for applications with large-scale, discrete input data with many missing values, such as that found in the Recommender Systems domain. Future directions beyond my thesis work include exploring the connection between fractal dimension and fundamental resolution, as well as defining new clustering models for data sets with a low fundamental resolution.

6. ACKNOWLEDGMENTS

This work is supported by the Applied Mathematics Program of the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231 and by NSF Award CCF-1637564.

7. REFERENCES

[1] D. C. Anastasiu, E. Christakopoulou, S. Smith, M. Sharma, and G. Karypis. Big data and recommender systems. 2016.

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[3] J. A. Chapman, M. Mascher, A. Buluç, K. Barry, E. Georganas, A. Session, V. Strnadova, J. Jenkins, S. Sehgal, L. Oliker, et al. A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome. *Genome biology*, 16(1):26, 2015.

[4] J. Cheema and J. Dicks. Computational approaches and software tools for genetic linkage map estimation in plants. *Briefings in bioinformatics*, 10(6):595–608, 2009.

[5] E. Christakopoulou and G. Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74. ACM, 2016.

[6] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[7] J. Gan and Y. Tao. Dbscan revisited: mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 519–530. ACM, 2015.

[8] A. Gionis, A. Hinneburg, S. Papadimitriou, and P. Tsaparas. Dimension induced clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 51–60. ACM, 2005.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[10] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[11] F. Lin and W. W. Cohen. Power iteration clustering. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 655–662, 2010.

[12] M. W. Mahoney and P. Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[13] V. Strnadová, A. Buluç, J. Chapman, J. R. Gilbert, J. Gonzalez, S. Jegelka, D. Rokhsar, and L. Oliker. Efficient and accurate clustering for large-scale genetic mapping. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pages 3–10. IEEE, 2014.

[14] V. Strnadová-Neeley, A. Buluç, J. Chapman, J. R. Gilbert, J. Gonzalez, and L. Oliker. Efficient data reduction for large-scale genetic mapping. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 126–135. ACM, 2015.

[15] V. Strnadová-Neeley, A. Buluç, J. R. Gilbert, L. Oliker, and W. Ouyang. Lira: A new likelihood-based similarity score for collaborative filtering. *arXiv preprint arXiv:1608.08646*, 2016.

[16] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[17] Y. W. Yu, N. M. Daniels, D. C. Danko, and B. Berger. Entropy-scaling search of massive biological data. *Cell systems*, 1(2):130–140, 2015.

Query Processing Based on Compressed Intermediates

Patrick Damme
Supervised by Wolfgang Lehner
Database Technology Group
Technische Universität Dresden, Germany
patrick.damme@tu-dresden.de

ABSTRACT

Modern in-memory column-stores employ lightweight data compression to tackle the growing gap between processor speed and main memory bandwidth. However, the compression of intermediate results has not been investigated sufficiently although accessing intermediates is as expensive as accessing the base data in these systems. Therefore, we introduce our vision of a *balanced query processing based on compressed intermediates* to improve query performance. In this paper, we provide an overview of the important research challenges on the way to this goal, present our contributions so far, and give an outlook on our remaining steps.

1. INTRODUCTION

With increasingly large amounts of data being collected in numerous areas ranging from science to industry, the importance of online analytical processing (OLAP) workloads increases constantly. OLAP queries typically address a small number of columns, but a high number of rows and are, thus, most efficiently processed by column-stores. The significant developments in the main memory domain in recent years have rendered it possible to keep even large datasets entirely in main memory. Consequently, modern column-stores follow a main memory-centric architecture. These systems have to face some new architectural challenges.

Firstly, they suffer from the new bottleneck between main memory and the CPU caused by the contrast between increasingly fast multi-core processors and the comparably low main memory bandwidth. To address this problem, column-stores make extensive use of data compression. The reduced data sizes achievable through compression result in lower transfer times, a better utilization of the cache hierarchy, and less TLB misses. However, classical *heavyweight* compression algorithms such as Huffman [10] or Lempel Ziv [20] are too slow for in-memory systems. Therefore, numerous *lightweight* compression algorithms such as differential coding [14, 16] and null suppression [1, 16] have been proposed, which are much faster and, thus, suitable for in-memory column-stores. Furthermore, especially for lightweight compression algorithms, many operators can directly process the compressed data without prior decompression.

Secondly, in main memory-centric column-stores, accessing the intermediate results during query processing is as expensive as accessing the base data, since both reside in main memory. Thus, intermediates offer a great potential for performance improvement, which can be exploited in two orthogonal ways: (1) intermediates should be avoided whenever possible [12, 15], or (2) intermediates should be represented in a way that facilitates efficient query processing.

In this thesis, we focus on the second approach by investigating lightweight compression of intermediates in main memory-centric column-stores. This direction has not been investigated sufficiently in the literature so far. Existing systems usually keep the data compressed only until an operator cannot process the compressed data directly, whereupon the data is decompressed, but not recompressed – due to the resulting computational overhead. However, using modern hardware and state-of-the-art lightweight compression algorithms, this computational overhead can be outweighed by the benefits of compressed data. Thus, our vision is a *balanced query processing based on compressed intermediates*. That is, in a query execution plan of compression-aware physical operators, *every* intermediate result shall be represented using a suitable lightweight compression algorithm which is selected in a compression-aware query optimization such that the benefits of compression outweigh its costs. To achieve this goal, this thesis addresses three aspects of the problem: the *structural aspect* on the basics of lightweight compression (Section 2), the *operational aspect* on physical operators for compressed data (Section 3), and the *optimization aspect* on compression-aware optimization strategies (Section 4). These aspects are designed to build upon each other, as will become clear in the following sections.

2. STRUCTURAL ASPECT

The *structural aspect* lays the foundations of this thesis by focusing on the basics of lightweight compression algorithms and on efficient transformations between the compressed formats of different algorithms. Thereby, our primary focus is on integer sequences due to their outstanding importance in column-stores: Other fixed-width data types, such as decimals, can be stored as integers and variable-width data types, such as strings, usually *need* to be represented by fixed-width integer codes from a dictionary to enable efficient processing. We have already completed our planned research in this aspect of the thesis.

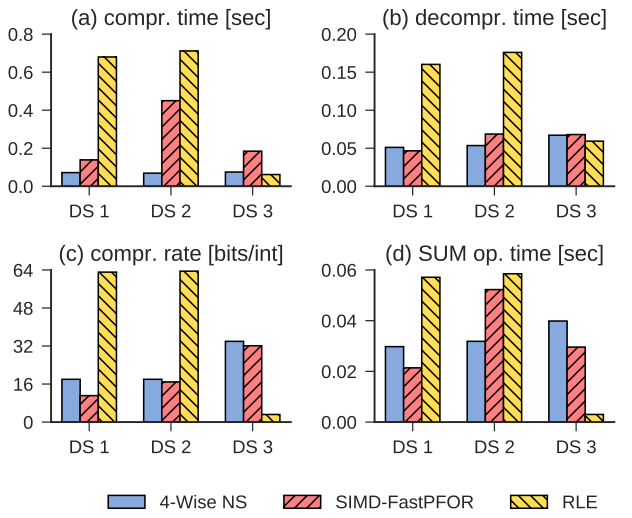
Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

2.1 Lightweight Data Compression

In the field of lossless lightweight data compression, we distinguish between *techniques*, i.e., the abstract ideas of how compression works conceptually, and *algorithms*, i.e., concrete instanciations of one *or more* techniques. So far, we consider five lightweight compression techniques for sequences of integers: frame-of-reference (FOR) [8, 21], differential coding (DELTA) [14, 16], dictionary coding (DICT) [1, 21], run-length encoding (RLE) [1, 16], and null suppression (NS) [1, 16]. FOR and DELTA represent each data element as the difference to either a certain given reference value (FOR) or to its predecessor (DELTA). DICT replaces each value by its unique key in a dictionary. The objective of these three well-known techniques is to represent the original data as a sequence of small integers, which is then suited for the actual compression using the NS technique. NS is the most studied lightweight compression technique. Its basic idea is the omission of leading zero bits in small integers. Finally, RLE tackles uninterrupted sequences of occurrences of the same value, so called *runs*. Each run is represented by its value and length. Obviously, these techniques exploit different data characteristics, such as the value range, the number of distinct values, and repeated values.

In the literature, numerous algorithms have been proposed for these techniques, e.g., [1, 8, 14, 16, 17, 19, 21], to name just a few examples. For our purposes of applying decompression and recompression *during* query execution, we depend on highly efficient implementations of these existing algorithms. One way to achieve these is to use single instruction multiple data (SIMD) extensions of modern processors, such as Intel's SSE and AVX, which allow the application of one operation to multiple data elements at once. In fact, the employment of SIMD instructions has been the major driver of the research in this field in recent years [14, 17, 19]. We have contributed to the corpus of proposed efficient implementations, e.g., through our vectorized algorithm for RLE [5], which is based on vectorized comparisons.

As lightweight compression algorithms are always tailored to certain data characteristics, their behavior in terms of performance and compression rate depends strongly on the data. Selecting the best algorithm for a given base column or intermediate requires a thorough understanding of the algorithms' behaviors subject to the data properties. Unfortunately, a sufficient comparative analysis had been missing in the literature. Thus, we conducted an experimental survey of several vectorized state-of-the-art compression algorithms from all five techniques as well as combinations thereof on numerous datasets, whereby we systematically varied the data characteristics [4, 5]. Figure 1a-c provide a sample of our results (the code was compiled using `g++ -O3` and the evaluation system was equipped with an Intel Core i7-4710MQ and 16 GB RAM). Our comparative analysis revealed several new insights. For instance, we could show how different data distributions affect the algorithms. We found that especially outliers in the distributions lead to a significant degradation in the performance and/or compression rate of certain algorithms. Furthermore, for fixed data characteristics, the best algorithm regarding performance is not necessarily the best regarding compression rate. Finally, we could show that combinations of different techniques can heavily improve the compression rate and even the (de)compression speed depending on the data. Summing up our findings, we can state that there is no single-best



DS	Value distribution	Run length
1	normal($\mu = 2^{10}, \sigma = 20$)	constant(1)
2	67% normal($\mu = 2^{6}, \sigma = 20$) 33% normal($\mu = 2^{27}, \sigma = 20$)	constant(1)
3	uniform($[0, 2^{32} - 1]$)	normal($\mu = 20, \sigma = 2$)

Figure 1: Behavior of three compression algorithms (a-c) and a SUM operator on the respective compressed formats (d) for three datasets with different characteristics, each having 100M data elements.

compression algorithm, but the choice depends on the data properties and is non-trivial. Our extensive experimental survey was made feasible by our benchmark framework for compression algorithms [6], which facilitates an efficient and organized evaluation process.

2.2 Direct Data Transformation

Assuming that the optimal compression algorithm was selected for a column, this algorithm might become suboptimal if the data properties change *after the decision*. The properties of the base data might change over time through DML operations. While this case might be handled offline, the problem is more urgent for intermediates, whose properties can change dramatically through the application of an operator. For instance, a column containing outliers might be stored in a format that can tolerate these, perhaps at the price of a slow decompression. A selection operator might remove the outliers, making a faster non-outlier-tolerant algorithm a better choice than the original one. This motivates the need for a *transformation* of the compressed representation of the data in some *source format* to the compressed representation in some *destination format*.

A naïve approach would take two steps: (1) Apply the decompression algorithm of the source format to the data, thereby materialize the entire uncompressed data in main memory. (2) Apply the compression algorithm of the destination format to that uncompressed data. The advantage of this approach is that it builds only upon existing (de)compression algorithms. However, since it materializes the uncompressed data in main memory, it is prohibitively expensive from our point of view, since we need to transform intermediates *during* query execution.

To address this issue, we introduced *direct transformation algorithms* in [7]. This novel class of algorithms is capable of accomplishing the transformation in *one step*, i.e., without the materialization of the uncompressed data. To provide an example, we proposed a direct transformation from RLE to 4-Wise NS. In 4-Wise NS [17], the compressed data is a sequence of compressed blocks of four data elements each. The direct transformation algorithm RLE-2-4WiseNS roughly works as follows: For each pair of run value and run length in the RLE-compressed input, it creates one block of four copies of the run value, compresses it *once*, and stores it out *multiple times* until it reaches the run length. That way, it saves the intermediate store and load as well as the repeated block compression performed by the naïve approach. Our experiments showed that this and other direct transformations yield significant speed ups over the naïve approach, if the data characteristics are suitable.

3. OPERATIONAL ASPECT

In our currently ongoing work in the *operational aspect*, we investigate how to integrate lightweight data compression into the query execution. Thereby, we assume that a multitude of compression algorithms is available to the system. We addressed the challenge of easily fulfilling this prerequisite in [9].

3.1 Processing Model for Compressed Data

Our vision of a query processing based on compressed intermediates can best be investigated using a processing model that actually *materializes all intermediates*. Furthermore, since we focus on column-stores and since lightweight compression algorithms are designed for sequences of values, *all* intermediates should use a *columnar representation*. Hence, we chose *column-at-a-time* as the processing model.

One example of a system that uses this processing model is MonetDB [11], which internally expresses queries in the Monet Algebraic Language (MAL) [2]. The central data structure of MAL is the binary association table (BAT), which is used to represent both, base data and intermediates. Conceptually, a BAT consists of a *head* containing record ids and a *tail* containing the actual data. However, since the head always contains a dense sequence of integers, it can be omitted. Thus, a BAT is essentially just an array of data elements, making it a perfect target for lightweight compression. MAL formally defines a set of operators that consume and produce BATs, such as selection, join, and projection. We decided to use MAL as the foundation of our work, but intend to adapt MAL operators to multiple compressed formats, which we discus in the next section.

3.2 Physical Operators for Compressed Data

When adapting MAL operators to compressed data, different *degrees of integration* are possible. Figure 2 presents the cases we plan to investigate. In general, an operator might consume i inputs and produce o outputs, each of which might be represented in its individual compressed format. Figure 2a shows the baseline case of processing only uncompressed data. In the following, we assume we want to support n compressed formats for *one* operator.

A first approach to support compressed intermediates is shown in Figure 2b. The original operator for uncompressed data is surrounded by a wrapper, which temporarily decompresses the inputs and recompresses the outputs. This

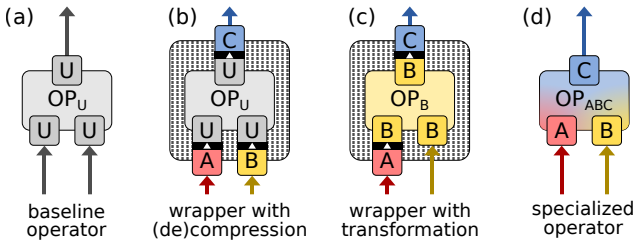


Figure 2: Integration of compression and operators. A to C are compressed formats; U is uncompressed.

approach is called *transient decompression* and was proposed in [3], but to the best of our knowledge, it has never been investigated in practice. For efficiency, the decompression(recompression) should not work on the entire inputs(outputs), but on small chunks fitting into the L1 cache. Changing the compressed format of the intermediates is possible by configuring the wrapper's input and output formats accordingly. The advantage of this approach is its simplicity: It reuses the existing operator and relies only on n already existing (de)compression algorithms. However, it does not exploit the benefits of working directly on compressed data.

The second approach is to adapt the operator such that it can work *directly* on compressed data (Figure 2c). Existing works such as [1, 13, 18] have already proposed *certain* operators on *certain* compressed formats. We plan to contribute to this line of research by covering the formats of recent vectorized compression algorithms. We have already investigated a SUM operator on compressed data and Figure 1d illustrates how significantly its performance depends on the data properties. We assume a common format for all inputs and outputs of the operator; for arbitrary combinations of formats, the operator is again wrapped. However, in this case the wrapper utilizes the *direct transformation* algorithms we developed in the structural aspect. Note that transformations are required only for those inputs(outputs) that are not represented in the operator's native format. The idea of bringing compressed inputs into a common format has already been proposed in [13], but only for joins on dictionary encoded data – and without *direct* transformations. We expect this approach to yield considerable speed ups compared to the first approach, since (i) the compressed data inside the wrapper is smaller, and (ii) the operator works directly on the compressed representation, such that it might, e.g., process more data elements in parallel using SIMD instructions. This approach requires n variants of the operator and $n^2 - n$ transformations, whereby the latter can be reused for all other operators. Nevertheless, the existence of a wrapper still causes a certain overhead.

The final approach tries to maximize the efficiency by tailoring the operator to a specific combination of formats (Figure 2d), making a wrapper unnecessary. Unfortunately, this approach implies the highest integration effort, requiring n^{i+o} operator variants. Thus, we intend to evaluate the potential of this approach first by considering a few promising combinations. If the results show significant improvements over the second approach, we could address the high integration effort, e.g., using code generation techniques.

The investigation of the above approaches is our current work-in-progress. Our ultimate goal is to integrate them into an existing column-store, most likely into MonetDB.

4. OPTIMIZATION ASPECT

There is no single-best compression algorithm, but the decision depends on the data characteristics [5]. Thus, compression must be employed wisely in a query plan to make its benefits outweigh its computational overhead. This motivates the development of compression-aware query optimization strategies, our future work in the *optimization aspect*.

The query optimizer is one of the most complex components of a DBMS. The crucial tasks it fulfills – such as algebraic restructuring and mapping logical to physical operators – are still fundamental for compressed query execution. Due to the high complexity, deeply integrating our compression-aware strategies into an existing optimizer is beyond the scope of this thesis. Instead, we envision a second optimization phase. This phase takes the optimal plan output by an existing optimizer as input and enriches it with compression by selecting an appropriate compressed format for each intermediate and replacing the physical operators by our derived operators for compressed data (Figure 3). In the following, we briefly describe the research challenges we will have to face to achieve this goal.

Local vs. global optimization. A simple approach could be to select the best format for each intermediate in isolation. While this implies a small search space, it might fail to find the optimal plan, e.g., by changing the format too often. A global optimization, on the other hand, requires effective pruning rules to cope with the huge search space.

Creation of a cost model. Due to the complex behavior of lightweight compression algorithms and, therefore, the operators based on them, the comparison of alternative decisions should be based on a cost model. Given a set of data properties, this model must provide estimates for, e.g., the compression rate and operator runtimes.

Estimation of the data characteristics. To use the cost model effectively, the characteristics of the data must be known. However, estimating the properties of all intermediates prior to query execution is non-trivial. Erroneous estimates might result in sub-optimal decisions. Therefore, adaptive optimization strategies might be a solution.

5. CONCLUSIONS

Modern in-memory column-stores address the RAM-CPU-bottleneck through lightweight data compression. However, employing compression has not been investigated sufficiently for intermediate results, although they offer great potential for performance improvement. In this context, we introduced our vision of a *balanced query processing based on compressed intermediates*. We discussed all relevant aspects of the problem in detail: (1) Our completed work in the structural aspect, where we contributed (i) an extensive experimental survey of lightweight compression algorithms and (ii) direct transformation algorithms. (2) Our ongoing work in the operational aspect, where we contribute different variants of physical operators on compressed data. (3) Our future work in the optimization aspect, where we will contribute compression-aware query optimization strategies.

Acknowledgments

This work was partly funded by the German Research Foundation (DFG) in the context of the project "Lightweight Compression Techniques for the Optimization of Complex Database Queries" (LE-1416/26-1).

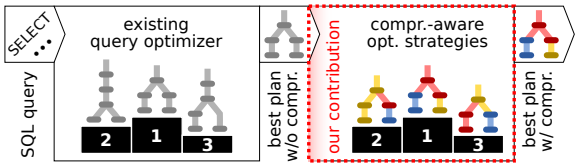


Figure 3: Compression-aware query optimization. Colors in the query plans stand for different compressed formats; grey stands for uncompressed data.

6. REFERENCES

[1] D. J. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD*, pages 671–682, 2006.

[2] P. A. Boncz and M. L. Kersten. MIL primitives for querying a fragmented world. *The VLDB Journal*, 8(2):101–119, 1999.

[3] Z. Chen, J. Gehrke, and F. Korn. Query optimization in compressed database systems. In *SIGMOD*, pages 271–282, 2001.

[4] P. Damme, D. Habich, J. Hildebrandt, and W. Lehner. Insights into the comparative evaluation of lightweight data compression algorithms. In *EDBT*, pages 562–565, 2017.

[5] P. Damme, D. Habich, J. Hildebrandt, and W. Lehner. Lightweight data compression algorithms: An experimental survey (experiments and analyses). In *EDBT*, pages 72–83, 2017.

[6] P. Damme, D. Habich, and W. Lehner. A benchmark framework for data compression techniques. In *TPCTC*, pages 77–93, 2015.

[7] P. Damme, D. Habich, and W. Lehner. Direct transformation techniques for compressed data: General approach and application scenarios. In *ADBIS*, pages 151–165, 2015.

[8] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE*, pages 370–379, 1998.

[9] J. Hildebrandt, D. Habich, P. Damme, and W. Lehner. Compression-aware in-memory query processing: Vision, system design and beyond. In *ADMS/IMDM@VLDB*, pages 40–56, 2016.

[10] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

[11] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: Two decades of research in column-oriented database architectures. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 35(1):40–45, 2012.

[12] T. Kissinger, B. Schlegel, D. Habich, and W. Lehner. QPPT: query processing on prefix trees. In *CIDR*, 2013.

[13] J. Lee, G. K. Attaluri, R. Barber, N. Chainani, O. Draese, F. Ho, S. Idreos, M. Kim, S. Lightstone, G. M. Lohman, K. Morfonios, K. Murthy, I. Pandis, L. Qiao, V. Raman, V. K. Samy, R. Sidle, K. Stolze, and L. Zhang. Joins on encoded and partitioned data. *PVLDB*, 7(13):1355–1366, 2014.

[14] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Software – Practice and Experience*, 45(1):1–29, 2015.

[15] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[16] M. A. Roth and S. J. Van Horn. Database compression. *SIGMOD Record*, 22(3):31–39, 1993.

[17] B. Schlegel, R. Gemulla, and W. Lehner. Fast integer compression using SIMD instructions. In *DaMoN*, pages 34–40, 2010.

[18] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. SIMD-Scan: Ultra fast in-memory table scan using on-chip vector processing units. *PVLDB*, 2(1):385–394, 2009.

[19] W. X. Zhao, X. Zhang, D. Lemire, D. Shan, J. Nie, H. Yan, and J. Wen. A general simd-based approach to accelerating compression algorithms. *ACM Transactions on Information Systems*, 33(3):15:1–15:28, 2015.

[20] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[21] M. Zukowski, S. Héman, N. Nes, and P. A. Boncz. Super-scalar RAM-CPU cache compression. In *ICDE*, 2006.

A Hardware-Oblivious Optimizer for Data Stream Processing

Constantin Pohl
supervised by Prof. Dr. Kai-Uwe Sattler
Technische Universität Ilmenau
Ilmenau, Germany
constantin.pohl@tu-ilmenau.de

ABSTRACT

High throughput and low latency are key requirements for data stream processing. This is achieved typically through different optimizations on software and hardware level, like multithreading and distributed computing. While any concept can be applied to particular systems, their impact on performance and their configuration can differ greatly depending on underlying hardware.

Our goal is an optimizer for a stream processing engine (SPE) that can improve performance based on given hardware and query operators, supporting UDFs. In this paper, we consider different forms of parallelism and show measurements exemplarily with our SPE PipeFabric. We use a multicore and a manycore processor with Intel's AVX/ AVX-512 instruction set, leading to performance improvements through vectorization when some adaptations like micro-batching are taken into account. In addition, the increased number of cores on a manycore CPU allows an intense exploitation of multithreading effects.

Keywords

Data Stream Processing, AVX, Xeon Phi, SIMD, Vectorization, Parallelism

1. INTRODUCTION

Technological advancement leads to more and more opportunities to increase application performance. For stream processing, data arrives continuously at different rates and from different sources. A stream processing engine (SPE) has to execute queries fast enough that no data is lost and results are gathered before the information is already outdated. A solution to achieve this is a combination of software optimizations paired with modern hardware for maximizing parallelism. It is difficult to find an optimal parametrization though, e.g. for the number of threads or load balancing between them. It gets even worse when different hardware properties come into play, like memory hierarchies or CPU core count.

For this paper, we consider different aspects and paradigms of parallelization that are applicable on data stream processing. In addition, first measurements on SIMD and multithreading realized on an Intel Core i7 and Intel Xeon Phi Knights Landing (KNL) with our SPE PipeFabric are shown. Our final goal is a full optimizer on a SPE capable of dealing with unknown UDFs in queries as well as with arbitrary hardware the system uses. Two consequential tasks arise from this.

- Exploitation of opportunities given by modern hardware, like wider CPU registers on Intel's AVX-512 instruction set, manycore processors with 60+ cores on a chip or increased memory size.
- Analysis of performance impacts by possible UDFs and operations, like computational effort or possible parallelization degree in case of data dependencies.

The rest of this paper is organized as follows: Next Section 2 is a short recapitulation about stream processing, possible parallelism and opportunities given by hardware. Section 3 summarizes related work done on SIMD parallelism, hardware-oblivious operations and stream partitioning in context of data stream processing. Our results are presented in Section 4, followed by Section 5 with future work. Section 6 with conclusions tops off this work.

2. PREREQUISITES

This section shortly summarizes requirements on stream processing and parallelization opportunities in addition to information on used hardware, like supported instruction sets.

2.1 Data Stream Processing

As already mentioned, high throughput and low latency are main requirements for stream processing. A data stream delivers continuously one tuple of data after another, possibly never ending. Queries on data streams have to consider that tuples can arrive at alternating rates and they get outdated after a while, because storing all of them is impossible. Operating with windows of certain sizes are a common solution for this property.

As a consequence, a query has to be processed fast enough to produce no outdated results as well as keeping up with eventually fast tuple arrival rates. Handling multiple tuples

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

at once through partitioning of data or operators exploiting parallelism possibilities is therefore a must.

2.2 Parallelism Paradigms

There are mainly three forms of parallelism on stream processing that can be exploited.

Partitioning. Partitioning can be used to increase speedup through splitting data on different instances of the same operator. Therefore every instance executed in parallel has to add its function on a fraction of data, increasing throughput of a query. However, additional costs for assigning data to instances and merging results afterwards arise, influencing the optimal partitioning degree.

Task-Parallelization. Operators of a query can execute in parallel if data dependencies allow such parallelism. A pipelining schema provides possibilities to achieve this, realized by scheduling mechanisms.

Vectorization. A single instruction can be applied on multiple data elements, called SIMD. For stream processing, some preparatory work is necessary to use this form of parallelism, like storing a number of tuples before processing them at once with SIMD support. On the one hand, it increases the throughput of a query while on the other hand batching up tuples worsens latency.

We focus on partitioning and vectorization, because task-parallelism is mainly a scheduling problem that is not of further interest at this point.

2.2.1 Partitioning

A speedup through partitioning is achieved mainly with multithreading. Each partition that contains operators processing incoming tuples uses a thread, which leads to challenges on synchronization or load balancing, especially on manycores, as shown by Yu et al. [6] for concurrency control mechanisms. Partitioning is a key for high performance when using a manycore processor, which provides support for 200+ threads at the cost of low clock speed. To investigate the right partitioning degree between reduced load on each partition and increased overhead from threads as well as an appropriate function for forwarding tuples to partitions additional observations have to be made. Statistics are a common solution, but can be far away from optimal performance in worst cases.

2.2.2 Vectorization

To use vectorization, certain requirements must be fulfilled. Without batching up tuples it is impossible to apply a vector function on a certain attribute. This leads to the next challenge, the cache-friendly formation of a batch. Without careful reordering, any vectorization speedup is lost through expensive scattered memory accesses. A possible solution for this is provided by gather and scatter instructions that index through masking certain memory addresses for faster access. Additional requirements on vectorization arise through the used operator function and data dependencies between tuples. The function must be supported by used instruction set, while dependencies are solved through rearrangements or even fundamental changes on the function of the operator.

Figure 1 shows the processing model of a query with batching. Tuples arrive one at a time on the data stream, being

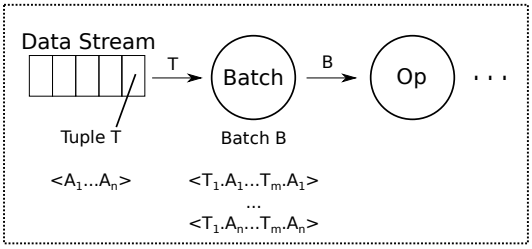


Figure 1: Processing Model

gathered first on a batching operator with attribute grouping in memory realized by vectors until batch size is reached and then forwarded to the next operator.

2.3 Hardware Opportunities

There are mainly two different ways to increase computational speed on hardware, distributed and parallel computing. Distributed computing uses usually many high-end processors connected to each other, sharing computational work between them. The disadvantage comes with communication costs. With requirements of low latency, we focus on parallel computing. Manycore processors like the Xeon Phi series from Intel use simpler cores, but many of them inside their CPU. This eliminates most of the communication costs, improving latency while providing wide parallelization opportunities compared to a single multicore processor.

The latest Xeon Phi KNL uses up to 72 cores with 4 threads each, available through hyperthreading. In addition, the AVX-512 instruction set can be used for 512bit wide operations (SIMD). This leads to great possibilities on partitioning and vectorization to reduce latency of a query. There are more interesting improvements on KNL like clustering modes, huge page support or high-bandwidth memory on chip (MCDRAM) which we will address in future work.

3. RELATED WORK

For parallelism through vectorization and partitioning on data streams, a lot of research has been done already, especially since manycore processors are getting more and more common. To achieve performance benefits, those manycore CPUs rely massively on SIMD parallelism and multithreading for speeding up execution time.

For data stream processing, the functionality of operations like joins or aggregations to give an example, are basically the same. However, for realization the stream processing properties have to be taken into account.

Polychroniou et al. [4] take a shot on different database operators, implementing and testing vectorized approaches for each one. Results show a performance gain up to a magnitude higher than attempts without SIMD optimization. In addition to this, their summary of related work gives a good review about research done with SIMD on general database operations.

For stream partitioning, the degree in terms of numbers of partitions as well as the strategy like the used split function for data tuples are the main focus of research. Gedik et al. [2] visited elastic scaling on stream processing where parallelization degree on partitioning is dynamically adjusted on

runtime, even for stateful operations. They reviewed typical problems when auto-parallelization is used like in most of other approaches.

For an optimizer that is hardware-oblivious, additional points must be considered. Hardware-oblivious means, that the optimizer is able to maximize performance on any hardware used, e.g a multicore or a manycore processor. Heimel et al. [3] implemented an extension for MonetDB called Ocelot, which is a proof of concept for hardware-oblivious database operators. They show that such operators can compete with hand-tuned operators that are fitted exactly for used processing units, like CPUs and GPUs. Teubner et al. [5] attended to the same topic before, looking deeper into hardware-conscious and hardware-oblivious hash joins.

4. EXPERIMENTS

With our experiments, we want to show the grade of impact on vectorization and multithreading for data stream processing. We therefore use two different processors, an Intel Core i7-2600 multicore CPU as well as an Intel Xeon Phi KNL 7210. As already mentioned before, KNL supports AVX-512 with 512bit register size and 256 threads, in contrast to i7s AVX with 256bit and 8 threads.

4.1 Preliminary Measurements

First measurements in Table 1 show needed runtime for corresponding CPUs when vectorization is enabled or disabled. Therefore an array with 64*1024*1024 elements is traversed, applying an addition operator (using 32bit precision) or square root operator (using 64bit precision) on each of the elements.

With vectorization, the speedup gain ideally corresponds directly with the number of elements processed at once, e.g. when using 32bit integers and the register size is 512bit, 16 elements are processed with one operator execution, leading to an expected 1/16th of runtime. However, this is not the case, because these elements needed to be accessed in memory (ideally in cache). With increased complexity (in terms of CPU cycles) this accessing costs are reduced compared to operators costs, as it can be seen in Table 1 with simple addition and complex square root. When computing the root, vectorization effectively doubles the execution speed on i7 processor and even more on KNL. On KNL, the registers can hold up to 8 64bit floating point numbers, resulting in around eight times faster execution on square root. On addition, however, even with prefetching mechanism it is not possible to pull data fast enough into the registers, because a simple addition just uses one CPU cycle. Therefore the full speedup cannot be achieved.

Results on square root on i7 processor can be explained through underlying hardware. Ideally, with AVX, 256bit registers and 64bit numbers, the speedup should result in

Processor	Vectorization	Addition	Square Root
i7-2600	disabled	42ms	187ms
	enabled	30ms	92ms
KNL 7210	disabled	98ms	998ms
	enabled	40ms	129ms

Table 1: Vectorization Runtime
Traversing array of 64*1024*1024 elements

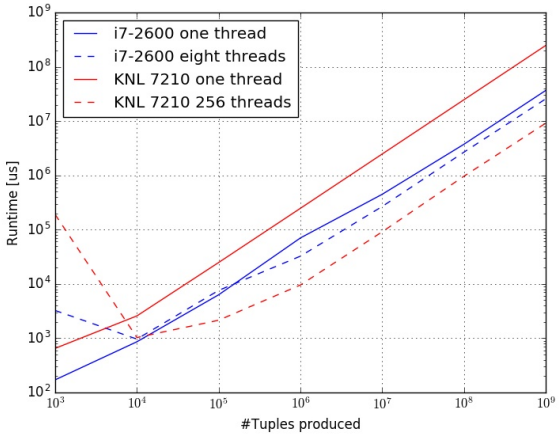


Figure 2: Query with Multithreading

around four times faster execution, however, it is only doubled. Further research points to how 256bit register are realized on i7-2600 (Sandy Bridge) - as processor of the first generation of AVX instruction set, it still uses two 128bit registers combined to achieve 256bit width. On performance there is only a small benefit of using 256bit loads and stores compared to 128bit, leading to only doubled speedup.

4.2 SPE Tests

Our SPE PipeFabric is a framework for data stream processing, written in C++. The data streams as source of tuples can be constructed for example through network protocols. A query consists out of different stream processing operators that combined are forming a dataflow graph. It supports selections, projections, aggregates, groupings, joins and table operations, as well as complex event processing. The focus of the framework lies on low latency, realized through efficient C++ template programming.

For the tests, the data stream produces tuples through a generator. Increasing the number of attributes or using different data types just add a constant delay for each tuple, increasing runtime without changing the curves significantly. Therefore only a single integer as an attribute is counted up. In Figure 2, the needed time to produce a certain amount of tuples (up to 10^9) is measured (note the logarithmic scale of y-axis). With low number of produced tuples, the overhead through thread generation worsens execution time. This changes very quick, providing an intense speedup. When generation is singlethreaded, KNL performs worse than i7-2600 caused by slower clock speed. But when cores are fully utilized, running maximum number of threads through OpenMP, KNL can outperform the multicore CPU easily.

Figure 3 shows speedup achieved on i7-2600 and KNL on queries with vectorization. Therefore tuples are batched first with different batch sizes on each run, followed by an aggregation operator which applies a simple addition or a complex square root on single attributes. These aggregations are performed with and without vectorization, the difference on runtime results directly into speedup, e.g. when runtime is halved, the speedup is 100%.

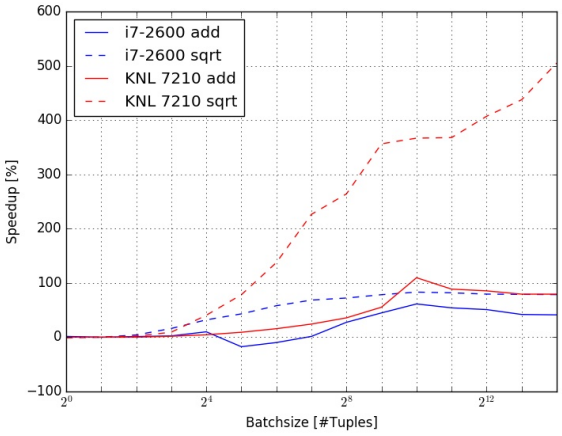


Figure 3: Query with SIMD

Taking a further look on Figure 3 reveals that speedup increases with batch size. This is relatively obvious, because with more tuples that can be processed at once by increased throughput, runtime of the query gets lower. However, this comes with the cost of latency, because results are delayed until a batch is full. An additional observation between addition and square root as aggregation operator can be made. With addition function, the performance gain is relatively low. This is because accessing the batch in cache takes longer than applying vectorized addition on it, even with prefetching mechanism. With square root, the operation takes significantly longer (in CPU cycles), so it is not limited that much by memory access on cache anymore.

5. RESEARCH PLAN

Vectorization and partitioning are the main two strategies which provide the most performance gain when set up accordingly to stream and query properties. Regarding vectorization, a batching mechanism is needed to exploit the full parallelism of wider CPU registers. With increased batch size, results of the query are delayed leading to increased latency but higher throughput. For partitioning, too many or too few partitions apart from optimum can even worsen the query execution time, same with uneven load balancing between partitions as shown by Fang et al. [1]. It is all a matter of right parametrization, depending on query, operators and underlying hardware.

This leads to our future work, where we will analyze impact on performance for certain strategies to finally come to an hardware-oblivious optimizer for queries on data stream processing, capable of dealing with UDFs as well as with different hardware sets. In difference to hardware-conscious optimizers which deliver only good precision for optimizations on certain hardware, our optimizer should generally being able to adapt its strategy on any given modern hardware.

However, it is a tradeoff between speedup and latency, greatly influenced by hardware used. We focus on parallelization on multicore and manycore CPUs, especially the latter one, because it is the most promising architecture for performance increasements with given requirements. With technical advancement additional chances are given, e.g. memory on package with high bandwidth, called MCDRAM on the latest manycore Xeon Phi KNL processor. When UDF support is realized, it is necessary to investigate key parameters for optimization, e.g. complexity of used function and data dependency.

6. CONCLUSION

For data stream processing, high throughput in terms of being able to process as many data as possible at the same time as well as low latency with fast responses on queries are main requirements. Exploiting parallelism is the answer, which is possible at different levels and degrees. In this paper, we show influence of parallelism on instruction level with vectorization as well as multithreading with our SPE PipeFabric and compare first results between multicore and manycore CPU.

SIMD effects improve performance when data is stored in a cache-friendly way within contiguous memory. For stream processing, each tuple cannot be processed one after another, therefore a batching mechanism is needed. This batching has to take care of storing data carefully for SIMD processing. Increased register width of Intel's Xeon Phi KNL with AVX-512 support leads to significant performance gains when not blocked by slow memory accesses. On the one hand, the computational workload must be high enough to surpass memory or cache accesses. This is not the case when additions on an aggregation operator are performed, as we showed in our experiments. On the other hand, with increased complexity SIMD operations are difficult to realize and must be supported by used instruction set.

Multithreading is another important factor when it comes to a manycore processor. Slow clock speed leads to poor singlethreaded performance compared to a multicore processor. This disadvantage is negated when enough cores can be utilized and parallelism is maximized. However, communication between threads, memory accesses and scheduling from threads to cores are no trivial tasks for optimizing performance, therefore more measurements are needed to prove results.

7. REFERENCES

[1] J. Fang, R. Zhang, T. Z. Fu, Z. Zhang, A. Zhou, and J. Zhu. Parallel Stream Processing Against Workload Skewness and Variance. *CoRR*, 2016.

[2] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu. Elastic Scaling for Data Stream Processing. *IEEE'14*, pages 1447–1463, 2014.

[3] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious Parallelism for In-memory Column-stores. *VLDB*, pages 709–720, 2013.

[4] O. Polychroniou, A. Raghavan, and K. A. Ross. Rethinking SIMD Vectorization for In-Memory Databases. SIGMOD, pages 1493–1508, 2015.

[5] J. Teubner, G. Alonso, C. Balkesen, and M. T. Ozsu. Main-memory Hash Joins on Multi-core CPUs: Tuning to the Underlying Hardware. ICDE, pages 362–373, 2013.

[6] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores. *VLDB*, pages 209–220, 2014.

Generalizing Matching Knowledge using Active Learning

Anna Primpeli
supervised by Christian Bizer
Data and Web Science Group
University of Mannheim
anna@informatik.uni-mannheim.de

ABSTRACT

Research on integrating small numbers of datasets suggests the use of customized matching rules in order to adapt to the patterns in the data and achieve better results. The state-of-the-art work on matching large numbers of datasets exploits attribute co-occurrence as well as the similarity of values between multiple sources. We build upon these research directions in order to develop a method for generalizing matching knowledge using minimal human intervention. The central idea of our research program is that even in large numbers of datasets of a specific domain patterns (matching knowledge) reoccur, and discovering those can facilitate the integration task. Our proposed approach plans to use and extend existing work of our group on schema and instance matching as well as on learning expressive rules with active learning. We plan to evaluate our approach on publicly available e-commerce data collected from the Web.

1. INTRODUCTION

Data integration is a long standing and very active research topic dealing with overcoming the semantic and syntactic heterogeneity of records located in the same or separate data sources [3]. While early work focused on integrating data from small numbers of datasets in a corporate context, there is an increasing body of research on integrating large numbers of datasets in the Web context, where an increased level of heterogeneity exists on both the instance and schema-level.

The matching approaches dealing with the task of integrating large numbers of datasets can be categorized by the addressed integration scenario. One scenario is the N:1, in which multiple datasets are matched against a central source; for instance, web tables against DBpedia [8] or product entities against a central catalog. The second scenario is the N:M, in which datasets are matched with each other without the help of an intermediate schema or a knowledge base.

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

Widely used matching systems such as COMA [2] indicate the need of rich matcher and aggregator libraries in order to solve different types of heterogeneity and find correspondences. A major finding of our group on integrating small numbers of datasets is that specific matchers and aggregators deriving from such rich libraries as well as property specific data normalization techniques can be combined into high quality, domain specific matching rules [5]. Such rules achieve a twofold goal: firstly, they give an insight into the current task by encoding matching knowledge, and secondly they achieve high quality correspondences by adapting to the nature of every matching scenario.

Research on integrating large numbers of datasets has shown that it is valuable to exploit attribute co-occurrence in the schema corpus as well as the similarity of data values not only between a central data source and a single external data source, but also the similarities of data values between multiple sources [4, 10]. A weak spot that can be observed in these approaches is that the employed data normalization techniques, similarity functions, and matching rules are not customized for the different types of entities and thus produce lower quality results than customized techniques.

The proposed research program builds upon this work and aims as its first goal to investigate the extent to which it is possible to generalize matching knowledge in order to improve matching quality in large-scale N:1 and N:M matching situations. The rationale for this approach is that typical patterns reoccur among entities of a certain domain. An example of such a pattern would be *"When matching entities representing the product type phones, it is effective to compare the attributes [brand, producer] using the following preprocessing methods [tokenization, lowercasing], the following similarity function [Levenshtein distance] and the following threshold [0.75]"*.

In most matching situations, collaboration between humans and machines is helpful in order to judge corner cases or to boot-strap matching with a certain amount of supervision [12]. Previous work of our group on guiding collaboration between humans and machines on small-scale matching scenarios has shown that using active learning can produce high quality matching results even with a small amount of human interaction [6]: The employed active learning approach was evaluated against six different datasets reaching between 0.8 and 0.98 F1 score after asking the human annotator to label ten pairs of entities as positive or negative matches. Building upon this work and extending certain steps of the active learning process, we formulate the second goal of the thesis, which is steering human attention in

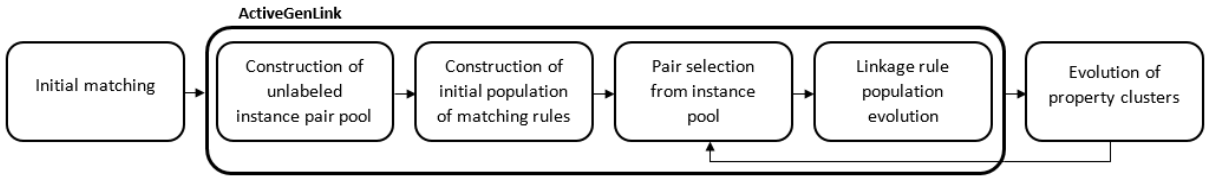


Figure 1: Proposed matching approach pipeline

large-scale matching situations with the aim to learn relevant, high quality matching knowledge.

Summing up, in the context of this work, we aim to answer the following research questions:

- Are domain specific patterns transferable within large-scale matching situations?
- How can we maximize the benefit of human supervision with respect to the discovery of those patterns?

In order to answer the above stated research questions, we will experiment with the N:1 and N:M matching scenarios using datasets created in the context of the Web Data Commons project[1] such as web tables, schema.org data and product data.

The remainder of this paper is organized as follows. Section 2 describes the proposed matching approach. Section 3 presents the data which we plan to use for evaluation. Finally, Section 4 gives a short overview of our workplan.

2. PROPOSED MATCHING APPROACH

This section gives an overview of the current plan for generalizing matching knowledge using active learning for the N:1 matching scenario. The planned approach involves three main steps. The first step is matching on the instance and schema-level with the goal to generate a preliminary set of schema correspondences which forms the basis for later refinement. Next, we build upon the concepts of the *ActiveGenLink* algorithm [6], an active learning approach based on genetic programming, which uses human supervision to evolve matching rules applicable on the instance-level. The final step is the refinement of the schema correspondences based on the instance-level matching results. The two last steps are iterated until the desired accuracy or the maximum amount of questions to the human annotator have been reached.

Figure 1 shows the steps of the matching process which will be the main guideline of this research. Below the individual steps are further explained, and the related state-of-the-art work upon which we build our proposed approach is presented together with our suggested methodological contributions.

2.1 Initial matching

The first step of our algorithm involves matching on the instance and schema-level with the goal to generate an initial set of correspondences which will be refined in the next steps of the algorithm. To achieve this, we employ existing techniques for large-scale matching.

For the N:1 scenario we use the T2K matching algorithm which involves cycles of instance and schema matching and achieves an F1 score of 0.8 on the instance-level and 0.7 on the schema-level for the task of matching web tables to DBpedia [11].

The resulting schema correspondences of this step are grouped into clusters, with each cluster representing a specific property such as *product name* or *brand*. The motivation behind property clusters is that matching information concerning one property can further affect the other elements of the cluster, as it will be later explained in Section 2.6.

In this step, we plan to reuse existing work to form our baseline for further improvement using active learning.

2.2 Construction of unlabeled instance pair pool

The second step involves the construction of an unlabeled pool of pairs of instances that are potential candidates for labeling by the user. Considering the complexity involved with matching large-scale data as well as our goal for creating generalized matching rules, the unlabeled instance pair pool should be constructed on the basis of two guidelines: computational space reduction and preservation of matching knowledge.

To achieve computational space reduction we propose an indexing and a blocking technique. We use three types of information to build an index value out of every instance: the instance name, the attribute labels and the attribute values using words or n-grams. After indexing, we make sure that the candidates for the unlabeled instance pair pool hold valuable information while eliminating the rest of them. To ensure this, different pair characteristics are evaluated. Such possible entity characteristics aside being likely matches, would be if the involved entities are described by many frequent properties and if they are head or tail entities, based on how often they occur in the data corpus.

After defining such informativeness criteria, we linearly scan over the entity names of the central source and we generate a pair if it is considered informative. Next, the generated pair is added in the unlabeled instance pair pool.

Thus, in this step we need to discover which characteristics make a pair a good candidate for the instance pair pool and how to combine them in order to draw the line between informative and non-informative pairs.

2.3 Construction of initial population of matching rules

As a next step, the initial linkage rules are created. We build upon the linkage rule definition introduced in [6]. A linkage rule is defined as a combination of different operators having a tree representation that gradually transforms with the evolution of the *GenLink* algorithm, a variation of the genetic algorithm [5]. A linkage rule contains the following set of operators:

[1]http://webdatacommons.org/

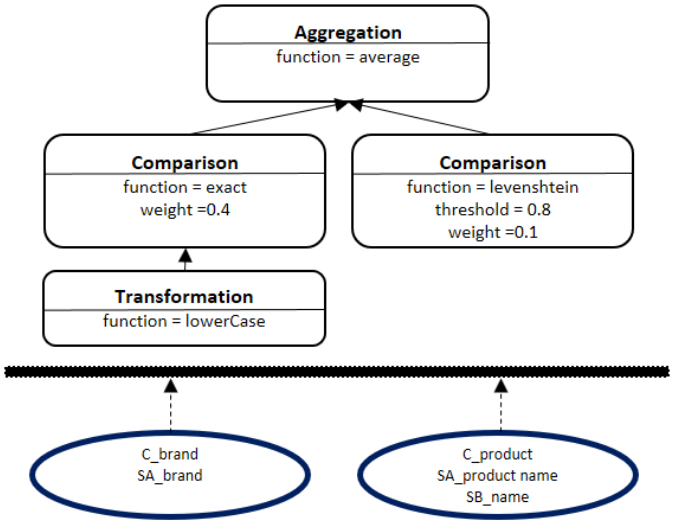


Figure 2: An example matching rule

(a) Property Operator: Selects the values of a property.

(b) Transformation Operator: Defines the transformation functions for the selected property values. Such transformations may be: case normalization, address standardization, stop-word removal, and structural transformations such as segmentation and extraction from values from URIs [5].

(c) Comparison Operator: Defines the distance metric and threshold that should be used to compare the selected values.

(d) Aggregation Operator: Defines the way to combine the results of the different comparison operators of the previous level.

The difference in our setting is that the property operators do not refer to specific properties but to property clusters, as introduced in Section 2.1. Thus, when a rule is applied to a specific instance pair from the pool of unlabeled pairs, the property operator checks if both entities contain a property which is part of any property cluster. If this is the case, the property operator outputs a pair of values. Otherwise it outputs an empty set of values. The functionality of the other operators remains the same.

Figure 2 shows an example rule of our matching approach. In the illustrated example the property operator selects the clusters that represent the *product brand* and the *product name* properties. In the next level, the values of the property *brand* are lowercased and a specific comparison operator is defined. Based on the weights and threshold the comparison operators normalize the similarity score to the range [0,1]. Finally, the results of the comparison operators are aggregated into a single score using the average aggregator which finally decides if the pair is a positive or a negative correspondence.

2.4 Pair selection from instance pool

In this step a pair of instances is selected from the instance pool and presented to the human annotator who provides a label as matching or non-matching. The goal of this step is to define a query strategy that selects the most informative pair to be labeled, thus minimizing the human involvement in the whole process.

For this we build on the three query strategies employed by [6]: 1. *query by committee* evaluates the unlabeled pairs against the current population of matching rules and selects the pair that causes the biggest disagreement, 2. *query by divergence* selects one pair out of every group in the similarity space, thus considering pairs which convey different similarity patterns, and 3. *query by restricted committee* uses the query by committee strategy but only considers the disagreements between the top K optimal matching rules of the current population.

Our strategy will build upon the existing ones and further clarify which other criteria should be considered in order to maximize the benefit of the selected pair. One possible direction of our query strategy could be to prefer pairs that contain many properties so that information about a bigger variety of properties can be learned after a pair has been annotated. In addition, the usage of a mixture between head and tail entities can prove effective in revealing information concerning the whole domain and not focus only on the frequent entities. Another possible component of our query strategy could be the characteristics of the properties of the selected pairs. Such characteristics might be frequency and the size of the cluster to which they belong. The rationale behind using those features is that if the answer of the human annotator gives further insight for a centroid of a property cluster then other properties may be indirectly affected, as described more detailed later in Section 2.6, thus leading to a faster convergence of the algorithm.

2.5 Linkage rule population evolution

In this step, we exploit the information provided in the previous step by the human annotator as supervision to evolve the population of matching rules. The goal is to gradually generate more customized and accurate matching rules which evaluate correctly against the labeled set of pairs. To achieve this we use the *GenLink* algorithm [5].

GenLink evolves the population of linkage rules in two steps, selection and transformation. Firstly, the matching rules are evaluated on the basis of their fitness on the current set of labeled pairs and selected using tournament selection [7]. The selected rules are then transformed by applying certain crossover operations. More specifically, a crossover operator accepts two linkage rules and returns an updated linkage rule that is built by recombining parts of the parents. In our setting we use the specific set of crossover operations of *GenLink*: *transformation*, *distance measure*, *threshold*, *combine operators*, *aggregation function*, *weight*, and *aggregation hierarchy*.

2.6 Evolution of property clusters

In the final step of our approach, the evolution of the property clusters which preserve the schema-level correspondences takes place. The goal is to gradually improve the matching accuracy on the schema-level whilst exploiting the information on the instance-level.

To achieve this we select the top rules of the current linkage rule population based on their fitness score and apply them on the unlabeled candidates. Possible metrics for calculating the fitness score are F1 or Matthews correlation coefficient in the case that the set of reference links is un-

balanced. As a result, we retrieve instance-level correspondences which we use as input for duplicate-based schema matching with the goal to refine the property clusters.

We follow the approach of DUMAS (Duplicate-based Matching of Schemas) [1] for improving schema correspondences based on instance-level duplicates. In their work, they evolve the meta-level similarities gradually by calculating similarities on the instance-level using the SoftTFIDF measure and then solving the transformed problem as a bipartite weighted matching one. In every iteration, schema-level matches are either confirmed, doubted, or rejected.

After calculating the schema-level similarities, the property clusters of our setting are refined. We will investigate how the move of one element from a cluster may affect the rest of the related elements. The indirect effects may be a result of frequent co-occurence or strong similarity. For example, consider the property cluster setting $C_1 = \{A, B, C\}$ and $C_2 = \{D, E\}$. Assuming that property A matches to properties D and E, we move A to cluster C_2. If we additionally know that property B is very close on the similarity space to property A, then B follows A thus formulating the final state of property clusters as: $C_1 = \{C\}$ and $C_2 = \{A, B, D, E\}$.

2.7 Convergence and output

The process iterates by selecting a new pair from the unlabeled instance pair pool, evolving the linkage rules, and further property cluster refinement as described in Sections 2.4, 2.5 and 2.6. The cycle of iterations terminates when either the evolved linkage rules achieve the desired fitness score or the maximum number of questions to the user has been reached.

The outputs of the proposed matching approach are instance and schema-level correspondences as well as generalized matching knowledge deriving from the linkage rules with the best fitness score. In the N:1 matching scenario the acquired knowledge can be used to annotate the knowledge base with rules concerning attribute relevance for matching, appropriate similarity functions, data normalization transformations, aggregation functions, and thresholds. In the N:M matching scenario we aim to exploit the resulting matching knowledge rules to annotate the implicit, mediated schema created through holistically matching entities.

3. EVALUATION

We plan to evaluate our approach on e-commerce data we already created in the context of the Web Data Commons project. The dataset contains 13 million product-related web pages retrieved from the 32 most frequently visited websites. We have manually annotated 500 electronic product entities and created a product catalog with 160 products of the same electronic categories. The total number of correspondences contained in our gold standard is 75,000, of which 1,500 are positive [9].

Other possible use cases for evaluating our approach would be web tables, linked open data and schema.org annotations. Web Data Commons provides the WDC Web Tables Corpus[2], the largest non-commercial corpus of web tables deriving from 1.78 billion HTML pages with 90.2 million relational tables.

[2]http://webdatacommons.org/webtables/

4. WORKPLAN

The outlined research program is currently in its first year. As an initial step towards accomplishing the goals defined in the context of this work we will focus on the N:1 matching scenario by applying the steps presented in Section 2. Next we will move on to the N:M matching scenario for which special indexing and blocking techniques need to be defined in order to deal with the increased complexity. Finally, granting that our proposed approach meets our goals, we aim to enhance existing knowledge bases by annotating them with matching knowledge.

5. REFERENCES

[1] A. Bilke and F. Naumann. Schema matching using duplicates. In *Proc. of the 21st Int. Conf. on Data Engineering*, pages 69–80. IEEE, 2005.

[2] H.-H. Do and E. Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proc. of the 28th Int. Conf. on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002.

[3] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *Proc. of the 32nd Int. Conf. on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[4] B. He and K. C.-C. Chang. A holistic paradigm for large scale schema matching. *SIGMOD Record*, 33(4):20, 2004.

[5] R. Isele. *Learning Expressive Linkage Rules for Entity Matching using Genetic Programming*. PhD thesis, University of Mannheim, 2013.

[6] R. Isele, A. Jentzsch, and C. Bizer. Active learning of expressive linkage rules for the web of data. In *Proc. of the 12th Int. Conf. on Web Engineering*, pages 411–418. Springer, 2012.

[7] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*, volume 5. Springer Science & Business Media, 2006.

[8] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, and C. Bizer. DBpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

[9] P. Petrovski, A. Primpeli, R. Meusel, and C. Bizer. The WDC gold standards for product feature extraction and product matching. In *Proc. of the 17th Int. Conf. on Electronic Commerce and Web Technologies*, pages 73–86. Springer, Cham, 2016.

[10] E. Rahm. The case for holistic data integration. In *Proc. of the 20th Advances in Databases and Information Systems Conf.*, pages 11–27. Springer, 2016.

[11] D. Ritze, O. Lehmberg, and C. Bizer. Matching HTML tables to DBpedia. In *Proc. of the 5th Int. Conf. on Web Intelligence, Mining and Semantics*, page 10. ACM, 2015.

[12] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data Curation at Scale: The Data Tamer System. In *Proc. of the Conf. on Innovative Data Systems Research*, 2013.

Comparing entities in RDF graphs

Alina Petrova
supervised by Prof. Ian Horrocks
and Prof. Bernardo Cuenca Grau

Department of Computer Science
University of Oxford

alina.petrova@cs.ox.ac.uk

ABSTRACT

The Semantic Web has fuelled the appearance of numerous open-source knowledge bases. Knowledge bases enable new types of information search, going beyond classical query answering and into the realm of exploratory search, and providing answers to new types of user questions. One such question is how two entities are comparable, i.e., what are similarities and differences between the information known about the two entities. Entity comparison is an important task and a widely used functionality available in many information systems. Yet it is usually domain-specific and depends on a fixed set of aspects to compare. In this paper we propose a formal framework for domain-independent entity comparison that provides similarity and difference *explanations* for input entities. We model explanations as conjunctive queries, we discuss how multiple explanations for an entity pair can be ranked and we provide a polynomial-time algorithm for generating most specific similarity explanations.

1. INTRODUCTION

Information seeking is a complex task which can be accomplished following different types of search behaviour. Classical information retrieval focuses on the query-response search paradigm, in which a user asks for entities similar to the input keywords or fitting the formal input constraint. Yet there exists a broad area of exploratory search that is characterized by open-ended, browsing behaviour [18] and that is much less well studied. Exploratory search encompasses activities like information discovery, aggregation and interpretation, as well as *comparison* [13].

Comparing entities, or rather, information available about the entities, is an important task and in fact a widely-used functionality implemented in many tools and resources. On the one hand, systems that highlight similarities between entities can focus on *how much* entities are alike, giving a similarity score to a pair (or a group) of entities [6]. On the other hand, systems can focus on *how* or *why*, in which aspects two entities are similar and different, by comparing entities and finding similar features. Such comparison is done in many domains and for various types of entities: hotels,[1] cars,[2] universities,[3] shopping items,[4] to name a few. However, as a rule, such systems perform a side-by-side comparison of items in a *domain-specific* manner, i.e., following a fixed, hard-coded template of aspects to compare (e.g., in case of hotels, it could be price, location, included breakfast, rating etc.). In a few more advanced systems, similarities are computed with respect to the type of information available about the input entities rather than following a rigid pattern. One such example is the Facebook Friendship pages.[5] Given two Facebook users, a friendship page contains all their shared information, be it public posts, photos, likes or mutual friends, as well as their relationship, if any (e.g., married, friends etc.). However, as in the aforementioned examples, comparison is done over a limited set of attributes.

Relying on a fixed set of aspects is a reasonable solution for tabular data with rigid and stable structure. On the other hand, a more flexible approach to entity comparison is needed for Linked Data, namely for loosely structured RDF graphs. However, all current systems with such functionality compare items following a predefined, domain-specific list of values to compare. Thus, an interesting research problem would be to create a framework for entity comparison that is domain- and attribute-independent.

The Semantic Web has fuelled the appearance of numerous open-source knowledge bases (KBs). Such KBs enable both automatic information processing tasks and manual search, and they facilitate new types of information search, going beyond classical query answering and providing answers to new types of user questions. For example, using KBs one can answer questions like *how are the two entities similar* or *what differs them*, i.e., perform entity comparison.

In this paper we propose to study such questions posed over one of the most common types of KBs — RDF graphs. In particular, we provide a formal framework for posing such questions and we model answers to these questions as similarity and difference *explanations*. We then discuss how

[1] www.flightnetwork.com/pages/hotel-comparison-tool/
[2] http://www.cars.com/go/compare/modelCompare.jsp
[3] http://colleges.startclass.com/
[4] http://www.argos.co.uk/static/Home.html
[5] Original announcement (cashed by the Wayback Machine): https://web.archive.org/web/20101030105622/http://blog.facebook.com/blog.php?post=443390892130

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

multiple explanations to a question can be ranked and we provide a polynomial-time algorithm for generating most specific similarity explanations. Finally, we outline directions of future research.

2. PRELIMINARIES

In what follows we use the standard notions of conjunctive queries (CQs), query subsumption and homomorphism. We disallow trivial CQs of the form $\top(X)$. We model RDF graphs as finite sets of triples, where a triple is of the form $p(s, o)$, p and s being URIs and o being a URI or a literal. Furthermore, we use the notion of a *direct product* of two graphs, adapted to RDF graphs:

Definition 1. Let I and J be RDF graphs, $t_1 = R(s_1, o_1)$ and $t_2 = R(s_2, o_2)$ be two triples. The *direct product* of t_1 and t_2, denoted as $t_1 \otimes t_2$, is the triple $R(\langle s_1, s_2\rangle, \langle o_1, o_2\rangle)$. The *direct product* $I \otimes J$ of I and J is the instance:

$$\{t_1 \otimes t_2 \mid t_1 \in I \text{ and } t_2 \in J\}.$$

3. COMPARISON FRAMEWORK

There are multiple ways of how we can define similarity and difference explanations and how we can model entity comparison. In our framework the formalism of choice is conjunctive queries (CQs). We model formal explanations as conjunctive queries and we consider the problem of finding such explanations as an instance of the query reverse engineering problem.

3.1 Similarity explanations

We would like similarity explanations to highlight common patterns for input entities. Thus, we model them as queries that return both of these entities, i.e, they match patterns fitting both entities. Let $\langle I, a, b\rangle$ be a tuple consisting of an RDF graph I and two URIs from the domain of I $a, b \in dom(I)$ representing input entities. Furthermore, given a query Q, let $Q(I)$ be the answer set returned by Q over I.

Definition 2. Given $\langle I, a, b\rangle$, a *similarity explanation* for a and b is a unary connected conjunctive query Q_{sim} such that $\{a, b\} \subseteq Q_{sim}(I)$.

Example 1. Given two entities Marilyn_Monroe and Elizabeth_Taylor and the Yago RDF graph [14], a possible similarity explanation is:

$$\begin{aligned} Q_{sim}(X) = &\mathsf{hasWonPrize}(X, \mathsf{Golden\_Globe}),\\ &\mathsf{diedIn}(X, \mathsf{Los\_Angeles}),\\ &\mathsf{hasGender}(X, \mathsf{female}),\\ &\mathsf{actedIn}(X, Y1),\\ &\mathsf{isLocatedIn}(Y1, \mathsf{United\_States}),\\ &\mathsf{isMarriedTo}(X, Y2),\\ &\mathsf{hasGender}(Y2, \mathsf{male}),\\ &\mathsf{hasWonPrize}(Y2, \mathsf{Tony\_Award}), \textit{etc.} \end{aligned}$$

Q_{sim} can be interpreted the following way: both Monroe and Taylor received a Golden Globes award, died in Los Angeles, acted in movies that were shot in the US and were married to men who received a Tony Award.

Using this definition, we can formulate the following decision problem: given $\langle I, a, b\rangle$, $SimExp$ is a problem of whether there exists a CQ Q such that $\{a, b\} \subseteq Q(I)$. The corresponding functional problem is to compute a query Q such that $\{a, b\} \subseteq Q(I)$, given $\langle I, a, b\rangle$. Note that both the definition of Q_{sim} and $SimExp$ can be easily generalized from a pair of entities to a set of input entities.

We specifically chose the condition to be $\{a, b\} \subseteq Q(I)$ for two reasons. Firstly, the form of Q does not depend on the rest of the data: it does not matter whether there exist other entities that match the graph pattern described by the query; moreover, queries fitting the subsumption condition will not be affected if new data is added. This is very important in the context of RDF graphs, since web data is intrinsically incomplete.

Secondly, it is known that the definability problem is coNExpTime-complete for conjunctive queries [3,15]. On the other hand, $SimExp$ can easily be shown to be in PTime: for conjunctive queries, it is sufficient to take the full join of all tables in the database instance.

Let $\mathbf{Sim}(a, b)$ be the set of all similarity explanations for a given $\langle I, a, b\rangle$. Obviously $\mathbf{Sim}(a, b)$ can be quite big, containing numerous explanations, however, we are interested in the most informative ones. Our assumption is that the more specific a similarity explanation is, the better.

Definition 3. Given $\langle I, a, b\rangle$, a *most specific similarity explanation* is a similarity explanation Q^{msp}_{sim} s.t. for all similarity explanations Q'_{sim} wrt $\langle I, a, b\rangle$: $Q^{msp}_{sim} \subseteq Q'_{sim}$.

The decision problem $SimExp^{msp}$ is the problem of deciding whether Q_{sim} is a most specific similarity explanation for the given $\langle I, a, b\rangle$.The related functional problem is to compute a most specific Q_{sim}.

The subsumption relation divides the set of all similarity explanations $\mathbf{Sim}(a, b)$ into $\subseteq$-equivalent classes. If $\mathbf{Sim}(a, b)$ is not empty, then $\mathbf{Sim}(a, b)^{msp}$ is not empty, and there exists a finite most specific similarity explanation Q^{msp}_{sim}, whose size is bounded by the size of I. This explanation can in fact be constructed in PTime (see Section 4).

3.2 Difference explanations

Analogous to similarity explanations, we model difference explanations as CQs, but this time we require only one of the input entities to be in the answer set.

Definition 4. Given $\langle I, a, b\rangle$, a *difference explanation* for a wrt b is a unary connected conjunctive query Q^a_{dif} such that $a \in Q^a_{dif}(I)$, but $b \notin Q^a_{dif}(I)$.

The notion of a difference explanation can be generalized to sets of entities: given an RDF graph I, a set of entities Pos and a set of entities Neg, a *difference explanation* for I and Pos wrt Neg is a unary connected CQ Q^{Pos}_{dif} s.t. $\forall p \in Pos$: $p \in Q^{Pos}_{dif}$ and $Neg \cap Q^{Pos}_{dif} = \emptyset$.

Given $\langle I, a, b\rangle$, $DifExp$ is the problem of deciding whether there exists a difference explanation Q^a_{dif}. The generalized difference explanation problem $DifExp$ can be solved using the most specific similarity explanation problem $SimExp^{msp}$: given I, Pos and Neg, first construct a most specific similarity explanation Q for entities in Pos (done in PTime), and then check whether none of the elements of Neg are in

the answer set of Q (conjunctive query evaluation is NP-COMPLETE). Hence, the complexity of generalized $DifExp$ is NP-COMPLETE.

Furthermore, we would like to introduce another definition of a difference explanation that is dependent on the similarities between a and b. We would like the difference explanation for a to be as relevant as possible, hence we model it to be dependent not only on the information about b, but also on the common patterns for a and b. One possible way to do so is the following: let $const(Q)$ be the set of constants appearing in a query Q and let $const(R(\bar{x}))$ be the set of constants appearing in an atom $R(\bar{x})$.

Definition 5. Given $\langle I, a, b\rangle$, a *difference explanation* for a wrt b and Q_{sim} is a different explanation $Q_{dif}^{a,sim}$ such that $\forall R(\bar{x}) \in Q_{dif}^{a,sim}$: $const(R(\bar{x})) \cap const(Q_{sim}) \neq \emptyset$.

Example 2. Let the input entities be $a =$ John_Travolta and $b =$ Quentin_Tarantino. Let Q_{sim} for a and b be an explanation that both persons starred in Pulp_Fiction: $Q_{sim}(X) =$ starredIn$(X,$ Pulp_Fiction$)$. Relevant difference explanations could be that Travolta also starred in Grease and other movies, while Tarantino has directed several movies, including Pulp Fiction: $Q^a_{dif}(X) =$ starredIn$(X,$ Grease$)$ and $Q^b_{dif}(X) =$ directed$(X,$ Pulp_Fiction$)$. On the other hand, an explanation that Travolta (unlike Tarantino) is married to Kelly Preston is rather irrelevant, since we have not compared the two persons with respect to their marital status.

4. TECHNICAL RESULTS

4.1 Algorithm for computing a most specific similarity explanation

We compute a most specific similarity explanation by constructing the *direct product* of the RDF graph, similar to the construction of the direct product of a database instances with itself [15]. Any RDF graph I $a \in dom(I)$ can be associated with a *canonical* unary conjunctive query $q_I(x_a)$ such that for each fact $R(c,d)$ in I there is an atom $R(x_c, x_d)$ in q_I, where x_c and x_d are variables and x_a is a free variable. Note that a is an answer to $q_I(x_a)$ over I. The following algorithm produces a most specific similarity explanation. In it, we first produce an instance with the domain from $dom(I)^2$, i.e., tuples $\langle c,d\rangle$ for $c,d \in dom(I)$, and then construct a canonical conjunctive query of this instance.

Claim 1. If $J \neq \emptyset$, then J is a maximal connected component of $I \otimes I$ such that $a \otimes b = \langle a,b\rangle \in dom(J)$.

Proof sketch: Firstly, if $J \neq \emptyset$, then $\langle a,b\rangle \in dom(J)$, by Step 1. Secondly, the while-loop on Step 5 is in fact the greedy procedure that generates the maximal connected component in $I \otimes I$. Indeed, the condition $R(c,e), R(d,f) \in I$ ensures that the fact $R(\langle c,d\rangle, \langle e,f\rangle)$ is in $I \otimes I$, and the condition that there must exist a fact in J that contains $\langle c,d\rangle$ or $\langle e,f\rangle$ ensures connectedness.

Claim 2. Let $\langle I, a, b\rangle$ be an input of Algorithm 1. Let $q_J(x_{\langle a,b\rangle})$ be the output, and J the instance obtained after the while loop on Step 5. Then all of the following hold.

(i) $\{a,b\} \subseteq q_J(I)$,

Algorithm 1: Algorithm for computing a most specific similarity explanation

Input: an RDF graph I, entities a, b from $dom(I)$.
Output: a most specific similarity explanation for a and b.

1 Let $J = \{R(\langle a,b\rangle, \langle c,d\rangle) \mid R(a,c), R(b,d) \in I\} \cup \{R(\langle c,d\rangle, \langle a,b\rangle) \mid R(c,a), R(d,b) \in I\}$;
2 **if** $J = \emptyset$ **then**
3 **return** *empty query;*
4 Let $J^* = \emptyset$;
5 **while** $J \neq J^*$ **do**
6 $J^* := J$;
7 $J := J \cup \{R(\langle c,d\rangle, \langle e,f\rangle) \notin J \mid R(c,e), R(d,f) \in I,$ and $\exists$ a fact in J that contains $\langle c,d\rangle$ or $\langle e,f\rangle\}$;
8 Construct $q_J(x_{\langle a,b\rangle})$;
9 **foreach** $x_{\langle c,c\rangle}$ *in* q_J, $c \notin \{a,b\}$ **do**
  `// Replace` $x_{\langle c,c\rangle}$ `with constant` c
10 $q_J(x_{\langle a,b\rangle}) := q_J(x_{\langle a,b\rangle})[x_{\langle c,c\rangle} \to c]$;
11 **return** $q_J(x_{\langle a,b\rangle})$.

(ii) For a connected unary conjunctive query $q'(x)$, if there exist homomorphisms $h_1, h_2 : q' \to I$ such that $h_1(x) = a$ and $h_2(x) = b$, then there exists a homomorphism $h : q' \to J$ such that $h(x) = \langle a,b\rangle$.

Corollary 1. Algorithm 1 produces a most specific similarity explanation.

4.2 Properties of the resulting query

The algorithm 1 runs in time polynomial to the size of the input RDF graph, and the size of resulting most specific similarity explanation is also polynomial to I. It should be noted that the output query tends to be non-minimal. For example, since Marilyn_Monroe and Elizabeth_Taylor acted in several movies that were shot in the US, $Q(X)$ will contain atoms like:

actedIn$(X, Y1)$, isLocatedIn$(Y1,$ United_States$)$,
actedIn$(X, Y2)$, isLocatedIn$(Y2,$ United_States$)$,
actedIn$(X, Y3)$, isLocatedIn$(Y3,$ United_States$)$, etc.

To avoid such redundancy, we can take the *core* of the query (i.e., apply the query minimization algorithm). Taking the core is an NP-COMPLETE problem [9, 11], hence, obtaining a most specific similarity explanation without redundant atoms is an NP-COMPLETE task.

5. RELATED WORK

So far only few works have studied *explanations* over RDF graphs [5, 10, 12], and there is no single formal definition of an explanation over RDF data. A lot of attention has been paid to discovering connections ("associations") between nodes [12], which boils down to finding and grouping together paths in the graph that connect one input node to another one. Such connectedness explanations are orthogonal (rather than alternative) to the similarity explanations modelled as queries, which we propose to study. The two types of explanations are intended to capture different relations between nodes: the former explore possible paths that link the two nodes together, while the latter seek to find commonalities in the neighbourhoods of the input nodes.

The problem of *reverse engineering a query* given some examples originated in late 1970s and was first introduced for the domain of relational databases [20]. Later it was extensively researched with respect to different query formats: regular languages [1], XML queries [7], relational database queries [16, 17, 19], graph database queries [4] and SPARQL queries [2]. The problem of QRE for RDF data was first studied by Arenas et al. [2] and was implemented by Diaz et al. [8]. In [2], the authors consider three different variations of QRE problem: the basic variation that requires the input mappings to be part of the answer set ($\Omega \subseteq [\![Q]\!]_G$); the one that allows positive examples Ω together with negative examples $\bar{\Omega}$ (such that $\Omega \subseteq [\![Q]\!]_G$ and $\bar{\Omega} \cap [\![Q]\!]_G = \emptyset$); and the variation that requires the examples from Ω to be *exactly* the answer set of Q ($\Omega = [\![Q]\!]_G$). The complexity of these three variations is then provided for fragments of SPARQL with AND, FILTER and OPT.

6. FUTURE WORK

As part of my PhD, I would like to continue studying the problem of entity comparison using RDF graphs in several research directions. So far we have investigated similarity and difference explanations, and we rank the former according to the preference condition based on subsumption. In particular, we assume that the highest ranked explanations are most specific similarity explanations. On the one hand, we would like to apply a similar rationale to difference explanations and to study most general difference explanations as most preferred ones. On the other hand, these may not be the optimal choices for a given user, hence we need to investigate other possible ranking conditions as well as means of user-specific ranking of explanations.

RDF graphs are inherently incomplete, hence it would be useful to consider a scenario where an explanation is produced over an RDF graph and a domain ontology that contains knowledge not explicitly present in the graph. Consider a graph G consisting of two facts: $Teacher(Bob)$ and $teaches(Alice, CS)$, — and a simple EL ontology O consisting of one axiom: $\exists teaches.Class \sqsubseteq Teacher$. Let the two input entities be *Alice* and *Bob*. Then a similarity explanation wrt G, O could be $Q(X) = Teacher(X)$, while we are unable to generate such a CQ using only graph data.

In our framework explanations are modelled as CQs, and while CQs are formulas with relatively high readability for a user, it is of interest to be able to verbalize explanations, transforming them into natural language sentences. For example, a formal explanation $Q(X) = \mathsf{livesIn}(X, \mathsf{London}), \mathsf{friendsWith}(X, Y), \mathsf{worksAt}(Y, \mathsf{Oracle})$ could be transformed into an English sentence *"Both input entities live in London and are friends with someone who works at Oracle"*.

While CQs correspond to a large part of queries issued over relational databases, i.e., they have relatively high expressivity, they cannot express things like negation or disjunction, which is a limitation. Hence, an interesting problem would be to consider more expressive languages, in particular, union of CQs and CQs with inequalities and numeric comparison.

Lastly, we are planning to implement a comprehensive comparison system that would compute most specific similarity explanations and most general difference explanations, to test it on real-world RDF graphs and to perform usability tests.

7. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[2] M. Arenas, G. I. Diaz, and E. V. Kostylev. Reverse engineering SPARQL queries. In *Proc. of the 25th Int. Conf. on World Wide Web*, pages 239–249, 2016.

[3] P. Barcelo and M. Romero. The complexity of reverse engineering problems for conjunctive queries. In *Proc. of 20th Int. Conf. on Database Theory*, 2017.

[4] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *18th Int. Conf. on Extending Database Technology (EDBT)*, 2015.

[5] G. Cheng, Y. Zhang, and Y. Qu. Explass: exploring associations between entities via top-k ontological patterns and facets. In *International Semantic Web Conference*, pages 422–437. Springer, 2014.

[6] S.-S. Choi, S.-H. Cha, and C. C. Tappert. A survey of binary similarity and distance measures. *J. Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[7] S. Cohen and Y. Y. Weiss. Learning tree patterns from example graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 31, 2015.

[8] G. Diaz, M. Arenas, and M. Benedikt. SPARQLByE: Querying RDF data by example. *Proceedings of the VLDB Endowment*, 9(13), 2016.

[9] G. Gottlob and A. Nash. Efficient core computation in data exchange. *Journal of the ACM*, 55(2):9, 2008.

[10] P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing relationships in RDF knowledge bases. In *Int. Conf. on Semantic and Digital Media Technologies*, pages 182–187, 2009.

[11] P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1):117–126, 1992.

[12] J. Lehmann, J. Schüppel, and S. Auer. Discovering unknown connections - the DBpedia relationship finder. *CSSW*, 113:99–110, 2007.

[13] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[14] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[15] B. ten Cate and V. Dalmau. The Product Homomorphism Problem and Applications. In *18th International Conference on Database Theory (ICDT 2015)*, pages 161–176, 2015.

[16] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. In *Proc. of the 2009 ACM SIGMOD Int. Conf. on Management of data*, pages 535–548, 2009.

[17] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query reverse engineering. *The VLDB Journal*, 23(5):721–746, 2014.

[18] R. W. White and R. A. Roth. Exploratory search: beyond the query-response paradigm, 2009.

[19] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *Proc. of the 2013 ACM SIGMOD Int. Conf. on Management of Data*, pages 809–820, 2013.

[20] M. M. Zloof. Query-by-example: A data base language. *IBM systems Journal*, 16(4):324–343, 1977.

Scalable Linkage across Location Enhanced Services

Fuat BASIK
Supervised By: Hakan Ferhatosmanoğlu and Buğra Gedik
Department of Computer Engineering, Bilkent University, Turkey
fuat.basik@bilkent.edu.tr

ABSTRACT

In this work, we investigate methods for merging spatio-temporal usage and entity records across two location-enhanced services, even when the datasets are semantically different. To address both effectiveness and efficiency, we study this linkage problem in two parts: *model* and *framework*. First we discuss models, including *k-l* diversity— a concept we developed to capture both spatial and temporal diversity aspects of the linkage, and probabilistic linkage. Second, we aim to develop a framework that brings efficient computation and parallelization support for both models of linkage.

1. INTRODUCTION

An important portion of digital footprint left behind by entities interacting with online services contains spatio-temporal references. This footprint is a fertile resource for business intelligence applications [11]. We refer to the services that create spatio-temporal records of their usage as *Location Enhanced Services* (LES). For instance, Foursquare/Swarm[1] — a popular social networking service, records the locations of users when they check-in at a point-of-interest (POI) registered in the system. Similarly, mobile phone service providers generate a record every time a call is made, which includes the cell tower whose coverage area contains the user's location.

Records with similar location and time naturally observe similar phenomena. The data analyst can gather such data from multiple sources, which are typically anonymized due to privacy concerns. These sources could generate semantically different datasets, or the semantic link between the sources could have been lost due to anonymization. As most data science tasks require large amount of data for accurate training with higher confidence, scientists need to combine data from multiple sources to produce accurate aggregate patterns. For example, spatio-temporal usage records belonging to the same real-world user can be matched across records from two different location-enhanced services, even when the datasets are semantically different. Another example would be linkage of the sensor data from different vendors that are embedded to the same moving system, i.e. self-driving cars. This linkage enables data scientists and service providers to obtain information that they cannot derive by mining only one set of usage records. Consider a LES provider who combines user segmentation results derived from its own usage records with social segmentation results derived from the publicly available Swarm records. There are several algorithmic and systems challenges to merge information from multiple sources of anonymized spatio-temporal data that are collected with necessary permissions. To cover both effectiveness and efficiency, we divide this linkage problem into two parts: *model* and *framework*.

To develop effective models, one needs to define a similarity or probabilistic measure for linkage, which considers time, location, and the relationship between the two. This is relatively simpler for many record linkage tasks [4], where linkage is defined based on a similarity measure defined over records (such as Minkowski distance or Jaccard similarity). In spatio-temporal linkage, for a pair of users from two different datasets to be considered as matching, their usage history must contain records that are close both in space and time; and there must not be *negative matches*, such as records that are close in time, but far in distance. We call such negative matches, *alibi*s. To address these challenges, we introduce two *linkage models*. The first one is based on *k-l diversity* — a new concept we have introduced to capture both spatial and temporal diversity aspects of the linkage. A pair of entities, one from each dataset, is called *k-l* diverse if they have at least k co-occurring records (both temporally and spatially) in at least l different locations, and, such pairs of entities must not have any alibis. The second model we aim to develop is based on *probabilistic* linkage — in which we seek to model the matching probability of two entities based on their spatio-temporal history. A pair of entities are called *match, or linked* with probability P, which is proportional to their common events aggregated on grids, and timestamps. P is inversely proportional to number of all other entities simultaneously acting at the same grid.

Considering that location-based social networks get millions of updates every day, linkage over hundreds of days of data would take impractically long amount of time. Naïve record linkage algorithms that compare every pair of records take $\mathcal{O}(n^2)$ time [6], where n is the number of records. The generic entity matching tools do not provide the necessary optimization for scalability and efficiency of spatio-temporal

[1] www.foursquare.com / www.swarmapp.com

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

linkage [3]. In order to merge data sets in a reasonable time, we will develop a scalable *framework* that takes advantage of the spatio-temporal structure of the data. The *ST-Link* algorithm we have recently modeled to realize the *k-l* diversity model in real world, uses two filtering steps before pairwise comparisons of candidate users, and makes use of spatial index trees, temporal sliding windows and log-structured merge trees [7]. In addition to effective indexing techniques, we believe efficiency could benefit from parallelization of computation.

2. LINKAGE MODELS

Datasets. We denote the two spatio-temporal usage record datasets from the two LES across which the linkage is to be performed as $\mathcal{I}$ and $\mathcal{E}$.

Entities and events. *Entities*, or users, are real-world systems or people who use LES. We use the terms user and entity interchangeably. They are represented in the datasets with their ids, potentially anonymized, which are different for the two LES. *Events* correspond to usage records generated by a LES as a result of users interacting with the service. For an event $e \in \mathcal{E}$ (or $i \in \mathcal{I}$), $e.u$ (or $i.u$) represents the entity associated with the event. We use $U_\mathcal{E}$ and $U_\mathcal{I}$ to denote the set of entity ids in the datasets $\mathcal{E}$ and $\mathcal{I}$, respectively. We have $U_\mathcal{E} = \{e.u : e \in \mathcal{E}\}$ and $U_\mathcal{I} = \{i.u : i \in \mathcal{I}\}$.

Location and time. Each event in the dataset contains location and time information. The location information is in the form of a region, denoted as $e.r$ for event e. We do not use a point for location, as for most LES the location information is in the form of a region. We assume the time information is a point in time.

2.1 k-l Diversity

The core idea behind the *k-l* diversity model is to locate pairs of entities whose events satisfy *k-l* diversity. Furthermore, such pairs of entities must not have any alibis.

Co-occurrence. Two events from different datasets are called co-occurring if they are close in space and time. For two records $i \in \mathcal{I}$ and $e \in \mathcal{E}$, closeness is defined in terms of intersection of regions. To capture closeness in time, we use a parameter α, and call two events are close in time if they are within a window of α time units of each other.

Alibi. While a definition of similarity is necessary to link events from two different datasets, a definition of dissimilarity is also required to rule out pairs of entities as potential matches in our linkage. Such *negative matches* enable us to rule out incorrect matches and also reduce the space of possible matches throughout the linkage process. We refer to these negative matches as *alibi*s. In this work, we use alibi to define events from two different datasets that happened around the same time but at different locations, such that it is not possible for a user to move from one of these locations to the other within the duration defined by the difference of the timestamps of the events.

Entity linkage. Let $x \in U_\mathcal{I}$ and $y \in U_\mathcal{E}$ be two entities. In order to be able to decide whether two entities are the same, we search for k co-occurring event pairs and at least l of them are at diverse locations. However, each co-occurring event pair does not count as 1, since each of these events could co-occur with many other events. Let $C(i, e)$ be the function to represent aforementioned co-occurrence relation of records i, and e, We weight these co-occurring event pairs as:

$$w(i,e) = |\{i_1.u : C(i_1, e) \wedge i_1 \in \mathcal{I}\}|^{-1} \cdot |\{e_1.u : C(i, e_1) \wedge e_1 \in \mathcal{E}\}|^{-1} \quad (1)$$

Given a co-occurring event pair between two entities, we check how many possible entities' events could be matched to these events. For instance, in Figure 1, consider the solid line at the top with the weight 1/6. The event on its left could be matched to events of 2 different entities, and the event on its right could be matched to events of 3 different entities. To compute the weight of a co-occurring pair, we multiply the inverse of these entity counts, assuming the possibility of matching from both sides are independent. As such, in the Figure 1, we get $1/2 \cdot 1/3 = 1/6$.

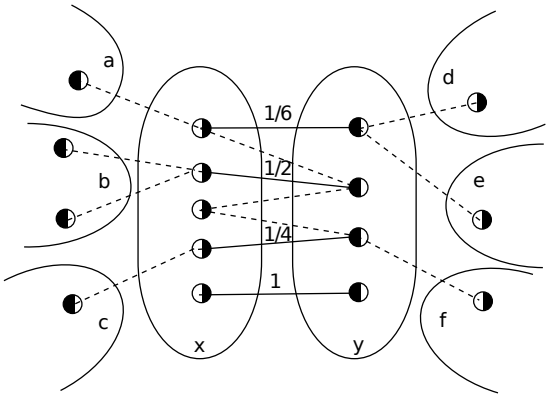


Figure 1: The co-occurring event pairs are shown using dashed lines. Events from a given entities are shown within circles. Entities a, b, c, and y are from one LES, and the entities d, e, f, and x are from the other LES.

l **diverse event pairs.** For the same entity pair to be considered l-diverse, there needs to be at least l unique locations for the co-occurring event pairs in it. However, for a location to be counted towards these l locations, the weights of the co-occurring event pairs for that location must be at least 1. Here, one subtle issue is defining a unique location. Intuitively, when datasets have different granularities for space, using the higher granularity ones to define uniqueness would give more accurate results. This could simply be a grid-based division of the space.

Entities x and y could be linked to each other, if they have k co-occurring event pairs in l diverse locations and their datasets do not contain alibi event pairs. Moreover, we only consider entity pairs for which there is no ambiguity, i.e. no two pairs (x, y) and (x, z) that are *k-l* diverse. Setting too low *k-l* values would lead many ambiguous pairs while too high values would lead many false negatives. To find the balance in between these two, we apply *elbow detection* techniques on k, and l distributions.

2.2 Probabilistic Linkage

Besides k-l diversity, we aim to model the spatio-temporal linkage problem using a probabilistic model. For consistency, we try to use the same notation with k-l diversity model as much as possible.

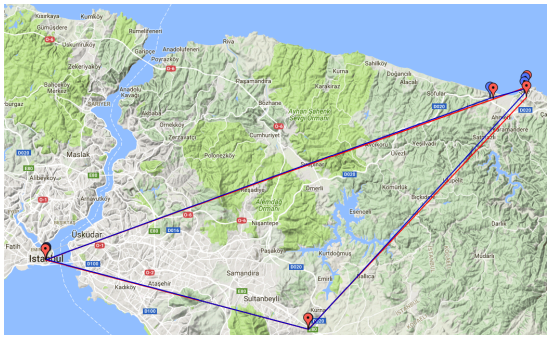


Figure 2: An example of 5 co-occurring events in 3 diverse locations from real world data. All weights are assumed to be 1

Probabilistic model starts by aggregating all of the entities on a common grid using the spatio-temporal features of the datasets. G_k denotes the set of entities in a specific cell in the grid and $|G_k^{\mathcal{I}}(t)|$ denotes the number of entities from dataset $\mathcal{I}$ in cell G_k at some time interval $[t - t + \alpha]$. Let $x \in U_{\mathcal{I}}$ and $y \in U_{\mathcal{E}}$ be two entities, and set of entities co-located with entity x in $U_{\mathcal{E}}$, and set of entities co-located with entity y in $U_{\mathcal{I}}$ at time $[t - t + \alpha]$, in the grid is given as $G_x(t)$, and $G_y(t)$ respectively.

Assuming two sets, S_1 and S_2, where $|S_1| = |S_2| = n$, the number of possible different complete matches (CM) (each element in S_1 has a partner in S_2) between the elements of these sets is $n!$, using trivial combinations without repetition. If the number of elements in the sets are not equal, i.e. $|S_1| = n \neq |S_2| = m$ then the problem turns into choosing m out of n (where $n \geq m$) and calculating complete match with m elements, i.e. $CM(n, m) = \binom{n}{m} m!$.

As the user set of one *LES* is typically not a subset of the second, we define the *partial match (PM)* where only k out of the m elements in S_2 match with k out of n elements in S_1. In this case we also need to choose k out of m and use the complete match, i.e. $PM(n, m, k) = \binom{m}{k} CM(n, k) = \binom{m}{k}\binom{n}{k} k!$.

Let $x \equiv y$ represents entities x, and y are the same real-world entity (match), the probability of a pair of specific two items to match each other is calculated by the number of events where x and y match divided by the number of events in the universal set of all possibilities.

If we are given a single snapshot of the grid at time t the probability of a randomly chosen pair of co-located entities in the different services being the same entity (we are assuming a complete match case only) can be found as following:

$$P(x \equiv y) = \frac{PM(n-1, m-1, m-1)}{PM(n, m, m)} \tag{2}$$

$$= \frac{CM(n-1, m-1)}{CM(n, m)} = \frac{1}{n} \tag{3}$$

$$= \begin{cases} \frac{1}{max(|G_k^{\mathcal{I}}(t)|, |G_k^{\mathcal{E}}(t)|)}, & \text{if } G_x(t) = G_y(t) = G_k \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

This is an intuitive result since a random entity from the smaller set can be equal to any element in the larger set with an equal probability.

If we have more than one sample of the entity (for time slots t_0 to time slot t_T where we don't necessitate the slots to be sequential in time) and the grid we can then use the history of the entity. The probability is similar except taking the tracks of the entities into account:

$$P(x \equiv y) = \begin{cases} \frac{\sum_{k=1}^{m} PM(n-1, m-1, k-1)}{\sum_{k=1}^{m} PM(n, m, k)}, & G_x(t_i) = G_y(t_i)\ \forall t_i \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where $n > m$.

An important issue is the decision on the values of n and m. If the entities x and y have l events sharing the same cell and time interval, one must look for all possible pairs that satisfy this property. Since the number of users in the intersection are smaller and is expected to decrease rapidly for large l values the probability of two users in respective services being the same user with the same track will be considerably high.

It is fair to assume that both the systems have considerable amount of common entities which will match. This commonality needs to be calculated empirically by ground truth values. After this value is found we can use this value to limit the k values (e.g. $k \in [k_1 : k_2]$) when calculating the probabilities and the probability in Theorem turns to:

$$P(x \equiv y) = \frac{\sum_{k=k_1}^{k=k_2} PM(n-1, m-1, k-1)}{\sum_{k=k_1}^{k=k_2} PM(n, m, k)} \tag{6}$$

One can relax the condition of equality for the tracking based on the location accuracy of the services and the chosen grid sizes. Moreover, similar to the *diversity* concept of the k-l diversity model, the distance among the shared cells could be used to distinguish between multiple pairs sharing a common user, with close matching probabilities, i.e. $P(x \equiv y) = P(x \equiv z)$.

3. FRAMEWORK

The second component of this doctoral work is the *framework* to perform the linkage efficiently. Naïve record linkage algorithms that compare every pair of records take $\mathcal{O}(n^2)$ time [6], where n is the number of records. Therefore, there are number of techniques implemented, i.e. indexing, blocking, to prune search space of linkage. To perform the linkage in reasonable time, we take advantage of the spatio-temporal structure of the data. To realize effectiveness of the k-l diversity model, we develop an algorithm called *ST-Link* [2]. Our implementation for the probabilistic model is still ongoing.

The *ST-Link* algorithm uses two filtering steps before pairwise comparisons of candidate entities are performed to compute the final linkage. It first distributes entities (users) over coarse-grained geographical regions that we call *dominating grid cells*. Such grid cells contain most of the activities of their users. For two users to link, they must have a common dominating grid. Once this step is over, the linkage is independently performed over each dominating grid cell. To identify the dominating grids, we make a sequential scan over all records, and utilize a quad-tree based index, which limits the area of the smallest grid from below. During the temporal filtering step, *ST-Link* uses a sliding window based scan to build candidate user pairs, while also pruning this

list as alibis are encountered for the current candidate pairs. Finally, our complete linkage model is evaluated over candidate pairs of users that remain following the spatial and temporal filtering steps. During this linkage step, we will need the time sorted events of the users at hand. For that purpose, during the forward scan, we also create a disk-based index sorted by the user id and event time. This index enables us to quickly iterate over the events of a given user in timestamp order, which is an operation used by the linkage step. Also, if one of the datasets is more sparse than the other, it performs the linkage by iterating over the users of the dense datasets first, making sure their events are loaded only once. This is akin to the classical join ordering heuristic in databases.

Our experimental evaluation shows that k-l diversity model is effective (up to 89% precision and 61% recall), yet the efficiency could benefit from a distributed approach. However, distributed processing is challenging due to mobility of users, and the scale of the data. First, distributing records based on their spatio-temporal features would spread records of a single user to multiple processing nodes, hence lead to high inter-machine communications cost. While the concept of dominating grid cells addresses this issue, scalability would still suffer from spatial skew of real data (in our experiments %18 of all records were residing on a single grid out of 120 grids). Since the temporal filtering techniques requires at least one batch of data to reside at the same machine (this issue exists in both models either for filtering or aggregating), records cannot be written to machines in parallel which would lead to low write performance. With these challenges identified, we are going to focus on optimizations of both models to create a single optimized *framework* which could efficiently perform linkage for both models. Such framework would be beneficial for both industry and academia when performing aggregation of semantically different datasets for social good applications, and when benchmarking the linkage research.

4. RELATED WORK

Record Linkage. One of the earliest appearances of the term *record linkage* is by Newcombe et al. [9]. In the literature, it is also referred to as entity resolution (ER), deduplication, object identification, and reference reconciliation, discussed in [4]. Most of the work in this area focus on a single type of databases and define the linked records with respect to a similarity metric. To the best of our knowledge, linking the users of the usage records, specifically targeted at spatio-temporal datasets is novel.

Spatial Record Linkage and Spatial Joins. Many join algorithms are proposed in the literature for spatial data [8]. Spatial record linkage and join algorithms are not directly applicable for spatio-temporal data as they are based on intersection of minimum bounding boxes, one-sided nearest join, or string similarity. Spatio-temporal joins have constraints on both spatial and temporal domains [1]. [10] is a recent work with similar motivation in which calculates weights of matching between users and applies maximum weight partitioning techniques. Their experiments validate the accuracy of this approach, but they do not focus on scalability.

User Identification. Our work has commonalities with the work done in the area of user identification. For instance, de Montjoye et al. [5] have shown that, given a spatio-temporal dataset of call detail records, one can uniquely identify the 95 % of the population by using 4 randomly selected spatio-temporal points. However, linking users is different from identification, as identification leaves whose data to aggregate question unanswered.

5. CONCLUSIONS & RESEARCH PLAN

In this paper, we introduced two linkage models for matching users across location enhanced services, and discussed implementation techniques. We have already realized a single machine implementation of the k-l diversity model with *ST-Link* algorithm. We are now working on validation and implementation of the probabilistic model, and aim to compare these two models with each other. Our single machine implementations showed that both models could benefit from a parallelized distributed implementation. Therefore, we set the development of a distributed and generic framework as the future goal of this doctoral work.

6. REFERENCES

[1] P. Bakalov and V. Tsotras. Continuous spatiotemporal trajectory joins. In *GeoSensor Networks*, volume 4540 of *Lecture Notes in Computer Science*, pages 109–128. Springer Berlin Heidelberg, 2008.

[2] F. Basik, B. Gedik, C. Etemoglu, and H. Ferhatosmanoglu. Spatio-temporal linkage over location-enhanced services. *IEEE Trans. on Mobile Computing*, PP(99):1–1, 2017.

[3] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: A generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, Jan. 2009.

[4] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[5] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3, 2013.

[6] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice & open challenges. In *VLDB Conference (PVLDB)*, 2012.

[7] S. Ghemawat and J. Dean. LevelDB. https://github.com/google/leveldb, 2015.

[8] E. H. Jacox and H. Samet. Spatial join techniques. *ACM Trans. Database Syst.*, 32(1), Mar. 2007.

[9] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records: Computers can be used to extract "follow-up" statistics of families from files of routine records. *Science*, 130(3381):954–959, 1959.

[10] C. Riederer, Y. Kim, A. Chaintreau, N. Korula, and S. Lattanzi. Linking users across domains with location data: Theory and validation. In *Proc. of the 25th Int. Conf.on WWW*, pages 707–719, 2016.

[11] A. Skovsgaard, D. Sidlauskas, and C. Jensen. Scalable top-k spatio-temporal term querying. In *IEEE Int. Conference on Data Engineering (ICDE)*, pages 148–159, March 2014.

Distributed Similarity Joins on Big Textual Data: Toward a Robust Cost-Based Framework

Fabian Fier
Supervised by Johann-Christoph Freytag
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
fier@informatik.hu-berlin.de

ABSTRACT

Motivated by increasing dataset sizes, various MapReduce-based similarity join algorithms have emerged. In our past work (to appear), we compared nine of the most prominent algorithms experimentally. Surprisingly, we found that their runtimes become inhibitively long for only moderately large datasets. There are two main reasons. First, data grouping and replication between Map and Reduce relies on input data characteristics such as word distribution. A skewed distribution as it is common for textual data leads to data groups which reveal very unequal computation costs, leading to Straggling Reducer issues. Second, each Reduce instance only has limited main memory. Data spilling also leads to Straggling Reducers. In order to leverage parallelization, all approaches we investigated rely on high replication and hit this memory limit even with relatively small input data. In this work, we propose an initial approach toward a join framework to overcome both of these issues. It includes a cost-based grouping and replication strategy which is robust against large data sizes and various data characteristics such as skew. Furthermore, we propose an addition to the MapReduce programming paradigm. It unblocks the Reduce execution by running Reducers on partial intermediate datasets, allowing for arbitrarily large data sets between Map and Reduce.

1. INTRODUCTION

Similarity joins are an important operation for user recommendations, near-duplicate detection, or plagiarism detection. They compute similar pairs of objects, such as strings, sets, multisets, or more complex structures. Similarity is expressed by similarity (or distance) functions such as Jaccard, Cosine, or Edit. A naive approach to compute a similarity self-join is to build the cross product over an input dataset and filter out all non-similar pairs. This approach has a quadratic runtime. In the literature, there are various non-distributed non-parallelized approaches for similarity joins based on a two-phase approach [1, 2, 3, 10, 14]. They compute a set of candidate pairs which is usually orders of magnitudes smaller than the cross product. Subsequently, they verify if the candidates are similar. We refer to them as *filter-and-verification* approaches. Motivated by increasing dataset sizes, MapReduce-based distributed approaches have emerged [5, 12, 13]. We conducted an extensive experimental study on nine current MapReduce-based set similarity join algorithms on textual data (to appear). There are two key findings. First, we compared the runtime of the MapReduce join algorithms to the runtime of competing non-distributed algorithms from the recent experimental survey of Mann et al. [11]. The runtime of MapReduce join algorithms on small datasets is inferior to the runtime of non-distributed approaches. This is not surprising due to the MapReduce overhead. The second finding is that none of the approaches can compute the join on larger (or even arbitrarily large) datasets. The runtimes increase so drastically that we terminated the executions after a long timeout.

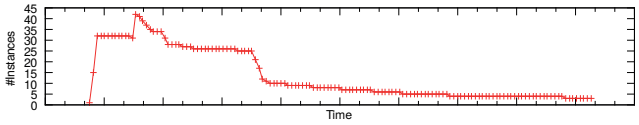


Figure 1: Straggling Reducer Issue.

We identified two main reasons for these runtime issues on large datasets. First, for every MapReduce-based similarity join algorithm we investigated we found non-optimal input datasets that lead to only a few running join Reduce instances while all other instances were left idle. That is, we often observed *Straggling Reducers*. Figure 1 shows the compute instance usage of a non-optimal join execution on a cluster of 48 compute instances. After roughly half the execution time, only a few instances are used. The instance usage is directly connected to data grouping and replication between Map and Reduce. All algorithms under investigation exploit and thus rely on certain data characteristics for replication and grouping. The most relevant characteristics are the global token frequency of the input dataset and the number of tokens in each record of a dataset. *Stop words*, which occur in a majority of records of a dataset, cause skewed data groups within most join approaches we investigated. As second cause, we identified memory overload within Reduce instances. All approaches heavily replicate data to leverage parallelization. The original MapReduce

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

programming paradigm as introduced by Dean et al. [4] requires the Reduce instances to wait for the Map steps to finish before the intermediate data groups are sorted and grouped by key. When the Reduce buffers are filled, data is spilled to disk, often causing high runtime penalties. The use of Combiners is not possible for similarity joins, because Reducers are stateful. This limitation is inherent to standard MapReduce.

In this paper, we propose an initial approach toward a robust framework to compute distributed similarity joins. It overcomes the Straggling Reducer issues and the input dataset size limitation we experienced in our past experiments. Our approach is twofold. First, we find a grouping and replication strategy which distributes compute load evenly over the existing compute instances. This is challenging since it is not sufficient to generate data groups of equal size. The runtime of a join computation within one group is dependent on characteristics of the data in the group such as record lengths. Second, we enable MapReduce to handle large intermediate datasets by proposing an extension for MapReduce which unblocks the Reduce execution based on statistical information gathered in a preprocessing step.

The idea of load balancing in MapReduce based on statistics is not new. The TopCluster algorithm [8] is an online approach which includes cardinality estimations at runtime. Our approach on the other hand needs exact data statistics in order to unblock the Reduce execution. These statistics have to be collected before the join execution. Our approach is comparable to the one by Kolb et al. [9], which involves a preprocessing MR job to collect data statistics and a join job which uses the statistics for an optimal data grouping and replication. We extend this approach by using the knowlege of the group sizes to unblock the Reduce execution. Furthermore, we tailor the grouping and replication to the specific problem of set similarity joins.

The contributions of this paper are as follows:

- We propose a first approach toward a robust distributed similarity join framework.
- We define a robust grouping and replication strategy leading to evenly distributed compute loads amongst the available compute nodes.
- We extend the MapReduce programming paradigm to unblock Reduce execution to handle (potentially arbitrarily) large datasets.

The structure of the paper is as follows. In Section 2, we give an overview on the similarity join problem, algorithmic approaches, and motivate the need for research with the runtime issues we experienced in our past experiments. In Section 3, we introduce our approach for a robust join framework and its interaction with an extension of MapReduce to unblock Reduce execution. In Section 4, we conclude our work and give an outlook on future work.

2. BACKGROUND

Without loss of generality, we use the set similarity self-join as a running example. Our framework can be applied to other filter-and-verification-based similarity joins as well. The set similarity join computes all pairs of similar sets (s_1, s_2) within a set of records S. A similarity function $sim(s_1, s_2)$ expresses the similarity between two records. For sets, there are similarity functions such as Jaccard, Cosine, or Dice. The user chooses a threshold t above which two sets are considered being similar. Formally, given a set S, a similarity function $sim(s_1, s_2)$, and a similarity threshold t, the set similarity join computes the set $\{(s_1, s_2) \in S \times S | sim(s_1, s_2) \geq t, s_1 \neq s_2\}$.

A naive approach computes the similarity on all pairs (s_1, s_2). Since it has a quadratic runtime, it is not feasible even for small datasets. In the literature, filter-and-verification approaches emerged. Their basic idea is to generate an (inverted) index over all input records. For each postings list, they compute the cross product (half of it in the self-join case to be exact) and the union of all these cross products. Each distinct record ID pair in the union is a candidate pair, because the two records contain at least one common token. These candidate pairs are further verified to compute the end result. Sophisticated filtering techniques keep the indexes and the number of candidate pairs small. The most prominent filter is the *prefix filter* [1, 2, 3]. Given a record length, a similarity function, and a similarity threshold, the prefix length is the minimum number of tokens which need to be indexed to guarantee an overlap of at least one common token if it is similar to another record.

Motivated by increasing dataset sizes, MapReduce-based versions of the filter-and-verification approach emerged [5, 12, 13]. The main idea is identical to the non-distributed approaches. It is to compute an inverted index, to compute the cross product on each postings list, and to verify the resulting candidate pairs. The inverted index is built as follows. A Map step computes key-value pairs with a token or a more complex signature as key. The MapReduce framework groups key-value pairs with the same key to one Reduce instance. This instance computes the cross product on the postings list. Depending on the value of the key-value pair (all tokens of the input record vs. only the record ID), the verification takes place within the Reduce, or there are further MapReduce steps to join the original records to the candidate pairs for the verification.

The key generation of all algorithms known to us relies on characteristics of the input data. In the most basic algorithm [7], each token in the input record is used as key. Obviously, the number of record groups is equal to the number of distinct tokens in the input dataset. The size of each record group depends on the global frequency of its key token. The data replication is dependent on the record lengths. For sufficiently large datasets with stop words (tokens which occur in almost every record) and/ or many long records, the Straggling Reducer effect occurs. More sophisticated approaches use a prefix filter, which reduces the number of tokens for replication to a prefix, which is shorter than the record length, but still dependent on it. The use of such filters shifts the Straggling Reducer issue to larger datasets and/ or datasets with longer records, but does not solve it for arbitrarily large datasets.

We expect the input of the similarity join to be text, which is integer-tokenized by a preprocessing step. The tokenization may include changing letter cases, stemming, or stop word removal. Depending on the preprocessing, the properties of input datasets vary by token distribution (stop words, infrequent tokens), dictionary size, and record size. The token distribution of textual data is usually Zipfian, which means that there are few very frequent tokens. This is a challenge for approaches relying on token distribution.

3. APPROACH

In Figure 2, we illustrate the dataflow of our framework. The first step computes exact data statistics. It computes record length frequencies and global token frequencies. These statistics can be computed in linear time and are highly parallelizable. Furthermore, it estimates runtime costs for the join execution, based on data samples with differing average record lengths. The second step computes the actual join. Every Map instance obtains the statistics from the first step via a setup function which is called once before the input data is read. Based on these statistics, it determines a suitable data grouping and replication and assigns keys to its output accordingly. Each join Reducer also obtains the statistics via the setup function. Using the statistics, it can compute the exact size of each group and start computing the join on this group once all data for it has completely arrived. This can happen before all Mappers have finished their execution. Note that this requires a change in the original MapReduce. The Reduce-side shuffling periodically counts the occurrences of each key in its input. It triggers the execution of the first-order function once one of the groups is complete. The Reducer can run any existing state-of-the art non-distributed similarity join.

In the following, we describe how to find a suitable grouping and replication based on the statistics. We use the Jaccard similarity function as an example, because it is the most commonly used function in the literature. Our framework is also applicable to any other set-based similarity function. Jaccard is defined by the intersection divided by the union of two records $\frac{|a \cap b|}{|a \cup b|}$. Note that records with differing lengths can be similar. Figure 3 shows this length relationship for a similarity threshold of 0.7. For each record length on the y axis, it shows on the x axis, which record lengths have to be considered as join candidates. Let us assume the input has a length distribution as depicted in Figure 4. In order to obtain data groups which can be self-joined independently, we group together all records with the same length and replicate each group to all larger length groups it can be similar to. The resulting groups would be $\{1\}, \{2\}, \{3,4\}, \{4,5,6,7\}, \{5,6,7,8\}, \{6,7,8,9,10\}$ etc. Note that these groups have very uneven cardinalities, for example $|\{1\}| = 8.000$, $|\{6,7,8,9,10\}| = 688.000$ etc.

In order to distribute the cardinalities evenly, we propose to apply a hash-based grouping and replication on these groups. Figure 5 shows an example for a hashing factor of 4. A hashing function assigns each record to one of 4 groups. Each record is distributed 4 times, so that it joins each other record in exactly one of the squares in the figure. Note that there is a tradeoff related to the hashing factor. If it is very low, there are only few large groups and the replication is low. If it is high, there are many small groups and the replication is high.

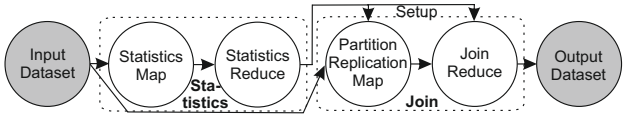


Figure 2: Dataflow Graph of our Execution Framework.

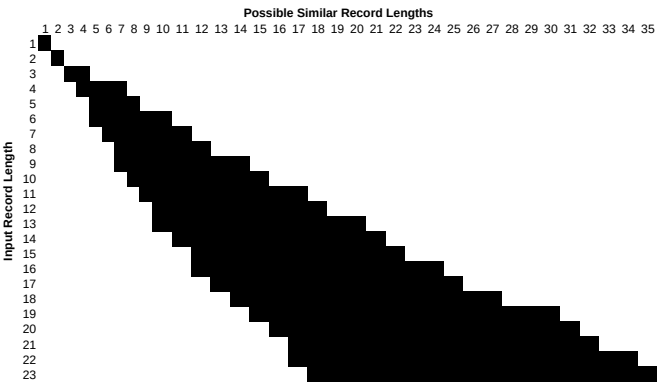


Figure 3: Possible Similar Record Lengths for Jaccard and Similarity Threshold 0.7.

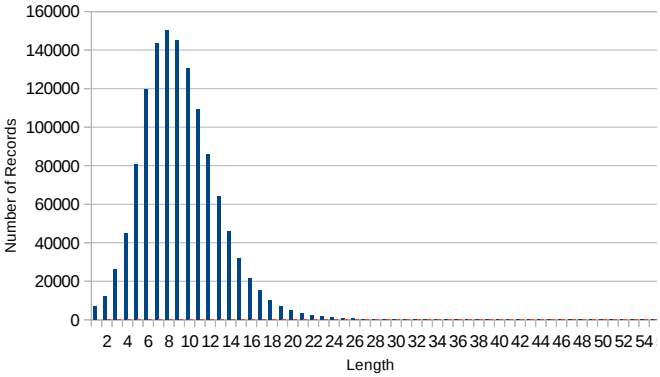


Figure 4: Example Record Length Distribution.

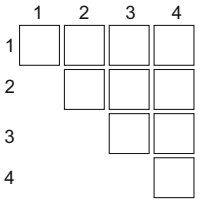


Figure 5: Hash-Based Grouping and Replication with hashing factor h=4.

An even data distribution is not sufficient to prevent Straggling Reducer effects. The costs of joining a partition with long records is higher than the cost of joining a partition with equally many short records. Let us assume that the Reducer of the join step has a quadratic runtime, which represents the worst case. The runtime costs of computing a self-join on one group of records with cardinality $groupSize$ and with an average record length of $avgRecLen$ can be estimated with Equation 1, assuming that the tokens in the records are sorted by a global token order allowing for a merge-join. In Figure 6, we show a plot of this cost estimation function. It shows that the costs grow exponentially with regard to the number of records in the group. The power of the increase grows exponentially with regard to the average length of the records in the group. In order

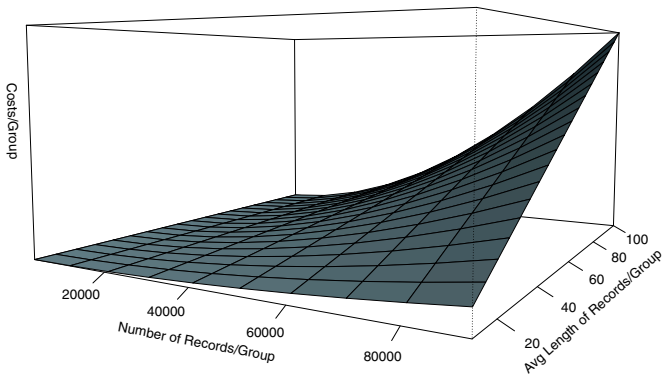


Figure 6: Cost Estimation for one data group.

to avoid a Straggling Reducer effect, our aim is to find a data grouping and replication which at least limits the maximum compute costs over all groups or ideally imposes equal computation costs for each data group. In Figure 6, equal computation costs would occur if all groups would exhibit a combination of number of records and average record lengths on an intersection of the graph with a horizontal plane.

$$\binom{groupSize}{2} * 2 * avgRecLen \tag{1}$$

Our idea is to optimize the overall computation costs with the hashing factor h as variable (Equation 2) and the constraint that the computation cost of each group may not be larger than the maximum cost threshold m, which ensures that no Reducer gets overloaded.

$$\min_{h \in \mathbb{N}^+} \sum_{group} costs(group, h), costs(group, h) \leq m \tag{2}$$

The group-wise costs within this equation could either be estimated by Equation 1 or it might use runtimes on sampled data from the statistics MapReduce step.

4. CONCLUSIONS, FUTURE WORK

In this paper, we introduced a first approach toward a distributed similarity join framework which is robust against arbitrary input dataset sizes and data characteristics such as skew. We plan to detail it out, implement it and run experiments with it. One crucial detail is to ensure that there is a sufficient number of record groups which is complete. If a Reduce instance collects only non-complete groups, straggling will still occur. Another open detail is the choice of the hash function for the join. Grouping and replication strategies from existing MapReduce-based similarity join approaches could be integrated in the proposed strategy. Especially signature creating approaches like MassJoin [5] and sophisticated grouping strategies like MRGroupJoin [6] using the pigeonhole principle are promising.

In future experiments, we are especially interested in the tradeoff between replication and group size. Furthermore, it is interesting if it pays off to use empirical runtime statistics for the join costs or simply estimate the runtime analytically.

5. ACKNOWLEDGMENTS

This work was supported by the Humboldt Elsevier Advanced Data and Text (HEADT) Center.

6. REFERENCES

[1] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

[2] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140. ACM, 2007.

[3] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 5–5. IEEE, 2006.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, pages 137–150, 2004.

[5] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 340–351. IEEE, 2014.

[6] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *Proceedings of the VLDB Endowment*, 9(4):360–371, 2015.

[7] T. Elsayed, J. Lin, and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 265–268. Association for Computational Linguistics, 2008.

[8] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 522–533. IEEE, 2012.

[9] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 618–629. IEEE, 2012.

[10] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: A partition-based method for similarity joins. *Proceedings of the VLDB Endowment*, 5(3):253–264, 2011.

[11] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment*, 9(9):636–647, 2016.

[12] A. Metwally and C. Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *Proceedings of the VLDB Endowment*, 5(8):704–715, 2012.

[13] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[14] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)*, 36(3):15, 2011.

Facilitating User Interaction With Data

Zainab Zolaktaf
Supervised by Rachel Pottinger
University of British Columbia
Vancouver, B.C, Canada
{zolaktaf, rap}@cs.ubc.ca

ABSTRACT

In many domains, such as scientific computing, users can directly access and query data that is stored in large, and often structured, data sources. Discovering interesting patterns and efficiently locating relevant information, however, can be challenging. Users must be aware of the data content and its structure, before they can query it. Furthermore, they have to interpret the retrieved results and possibly refine their query. Essentially, to find information, the user has to engage in a repeated cycle of data exploration, query composition, and query answer analysis. The focus of my PhD research is on designing techniques that facilitate this interaction. Specifically, I examine the utility of recommender systems for the data exploration and query composition phases, and propose techniques that assist users in the query answer analysis phase. Overall, the solutions developed in my thesis aim to increase the efficiency and decision quality of users.

1. INTRODUCTION

With the advent of technology and the web, large volumes of data are generated and stored in data sources that evolve and grow over time. Often, these sources are structured as relational databases that users can directly query and explore. For instance, astronomical measurements are stored in a large relational database, called the Sloan Digital Sky Survey (SDSS) [19, 21]. Climate data collected from various sources is integrated in relational databases and offered for analysis by users [5].

At a high level, user interaction with data involves two phases: a *query composition* phase, where the user composes and submits a query, and a *query answer analysis* phase, where the user analyses query answers produced by the system. During both phases, however, users can face problems in understanding the data.

Consider, for example, the scientific computing domain. The SDSS schema has over 88 tables, 51 views, 204 user-defined functions, and 3440 columns [14]. A variety of users, ranging from high school students to professional astronomers, with varied levels of skills and knowledge, interact with this database. Furthermore, scientific databases are typically used for Interactive Data Exploration (IDE), where users pose exploratory queries to understand the content and find patterns [13]. Efficiently composing queries over this data to discover interesting patterns, is one of their main challenges.

After successfully composing the query, the next step is to interpret query answers. However, the retrieved results can often be difficult to understand. For example, consider an aggregate query `SELECT AVG(TEMPERATURE)` over climate data. In the weather domain, observational data regarding atmospheric conditions is collected by several weather stations, satellites, and ships. For the same data point, e.g., temperature on a given day, there can be conflicting and duplicate values. Consequently, the aggregate query can have an overwhelming number of correct and conflicting answers. Here, mechanisms that aid the user in understanding the query answers are required.

In my thesis, I develop techniques that assist user interaction with data. I consider the data exploration and query composition phase, and examine the utility of recommendation systems for this phase. Furthermore, I consider the query answer analysis phase and devise efficient techniques that provide insights about query answers. More precisely, I study three problems: 1. how do classical recommendation systems perform with regards to exploration tasks in standard recommendation domains, and how can we modify them to facilitate data exploration more rigorously (Section 2)? 2. what are the challenges of recommendation in the relational database context and which algorithms are appropriate for helping users explore data and compose queries (Section 3)? 3. how can we assist users in the query answer analysis phase (Section 4)? Overall, I aim to develop techniques that help users explore data and increase their decision quality.

2. FACILITATING DATA EXPLORATION WITH RECOMMENDER SYSTEMS

One way to facilitate data navigation and exploration is to find and suggest items of interest to users by deploying a *recommendation system* [6, 16, 29]. Classical recommendation systems are categorized into content-based and collaborative filtering methods.

Content-based methods use descriptive features such as genre of movies, or user demographics, to construct informative user and item profiles, and measure similarity between

them. But descriptive features might not be available. Collaborative filtering methods instead infer user interests from user interaction data. The main intuition is that users with similar interaction patterns have similar interests.

The interaction data may include *explicit* user feedback on items, such as user ratings on movies, or *implicit* feedback, such as purchasing history, browsing and click logs, or query logs [11]. An important property of the interaction data is that the majority of items (users) receive (provide) little feedback and are infrequent, while a few receive (provide) lots of feedback and are frequent. But many models only work well when there is a lot of data available, i.e., they make good recommendations for frequent users, and are biased toward recommending frequent items [6, 15, 17].

However, recommending popular items is not sufficient for exploratory tasks. Users are likely already aware of popular items or can find them on their own. Concentrating on popular items also means the system has low overall coverage of the item space in its recommendations. It is essential to develop methods that help users discover new items that may be less common but more interesting. Therefore, we investigate the following research question:

> How do existing recommendation models perform with regard to data exploration tasks in standard recommendation domains, and how can they be modified to facilitate data exploration more rigorously?

To answer this question, we focus on top-N item recommendation, where the goal is to recommend the most appealing set of N items to each user [6]. Informally, the problem setting is as follows: we are given a log of explicit user feedback, e.g., ratings, for different items. We want to assign a set of N unseen items to each user.

2.1 Solution

In our solution [20], we focused on promoting less frequent items, or *long-tail* items, in top-N sets to facilitate exploration. Recommending these items introduces novelty and serendipity into top-N sets, and allows users to discover new items. It also increases the item-space coverage, which increases profits for providers of the items [3, 6, 26, 22]. Our main challenge was in promoting long-tail items in a targeted manner, and in designing responsive and scalable models. We used historical rating data to learn user preference for discovering new items. The main intuition was that the long-tail preference of user u, captured by θ^*_u, depends on the types of long-tail items she rates. Moreover, the long-tail type or weight of item i, captured by w_i, depends on the long-tail preference of users who rate that item. Based on this, we formulated a joint optimization objective for learning both unknown variables, $\boldsymbol{\theta}^*$ and $\mathbf{w}$.

Next, we integrated the learned user preference estimates, $\boldsymbol{\theta}^*$, into a generic re-ranking framework to provide customized balance between accuracy and coverage. Specifically, we defined a re-ranking framework that required three components: 1. an *accuracy recommender* that was responsible for recommending accurate top-N sets. 2. a *coverage recommender* that was responsible for suggesting top-N sets that maximized coverage across the item space, and consequently promoted long-tail items. 3. the user long-tail preference.

In contrast to prior related work [1, 10, 27], our framework learned the personalization rather than optimizing using cross-validation or parameter tuning; in other words, our personalization method was independent of the underlying recommendation model.

	Algorithm	P@5	R@5	L@5	C@5
MT-200K	Random	0.000	0.000	0.871	**0.873**
	Pop [6]	**0.051**	**0.080**	0.000	0.002
	MF [28]	0.000	0.000	**1.000**	0.001
	5D_ACC [10]	0.000	0.000	0.995	0.157
	Cofi^R [24]	0.025	0.046	0.066	0.020
	PureSVD [6]	0.018	0.022	0.001	0.067
	$\theta^{*\text{Dyn900}}_{\text{Pop}}$ [20]	0.027	0.050	0.416	0.171

Table 1: Top-5 recommendation performance.

We evaluated our framework on several standard datasets from the movie domain. Table 1 shows the top-5 recommendation performance for the MovieTweetings 200K (MT-200K) dataset [9] which contains voluntary movie rating tweets from users. For accuracy, we computed precision (P@5) and recall (R@5) [6] wrt the test items of users. Long-tail accuracy (L@5) [10], is the normalized number of long-tail items in top-5 sets per user. Long-tail items are those that generate the lower 20% of the total ratings in the train set, based on the Pareto principle or the 80/20 rule [26]. Coverage (C@5) [10] is the ratio of the number of distinct items recommended to all users, to the number of items.

We compared with non-personalized baselines: Random that has high coverage but low accuracy, and most popular recommendation (Pop) [6], that provides accurate top-N sets but has low coverage and long-tail accuracy. We also compared with personalized algorithms: matrix factorization (MF) with 40 factors, L2-regularization, and stochastic gradient descent optimization [28], a resource allocation approach that re-ranks MF (5D_ACC) [10], CofiRank with regression loss (Cofi^R) [24], and PureSVD with 300 factors [6]. On MT-200K, we chose the non-personalized Pop algorithm as our accuracy recommender, and combined it with a dynamic coverage recommender (Dyn900) introduced in [20]. Our personalized algorithm is denoted $\theta^{*\text{Dyn900}}_{\text{Pop}}$. Table 1 shows that while most baselines achieve best performance in either coverage or accuracy metrics, $\theta^{*\text{Dyn900}}_{\text{Pop}}$ has high coverage, while maintaining reasonable accuracy levels. Furthermore, it outperforms the personalized algorithms, PureSVD and Cofi^R, in both accuracy and coverage metrics.

3. FACILITATING DATA EXPLORATION AND QUERY COMPOSITION

Getting information out of database systems is a major challenge [12]. Users must be familiar with the schema to be able to compose queries. Some relational database systems, e.g, SkyServer, provide a sample of example queries to aid users with this task. However, compared to the size of the database and complexity of potential queries, this sample set is small and static. The problem is exacerbated as the volume of data increases, particularly for IDE. A mechanism that helps users navigate the schema and data space, and exposes relevant data regions based on their query context, is required. We consider using recommendation systems in this setting and focus on the following research question:

> What are the challenges of recommendation in the database context, and which algorithms are suitable for facilitating interactive exploration and navigation of relational databases?

To answer this question, we address top-N *aspect* recommendation, where the goal is to suggest a set of N aspects to the user that facilitate query composition and database exploration. Similar to the collaborative filtering setting in Section 2, we analyse user interaction data, available in a *query log*. Informally, the problem setting is as follows: we are given a query log that is partitioned into sessions, sets of queries submitted by the same user. Furthermore, we also have a relational database *synopsis* with information about the schema of the database (#relations, #attributes, and foreign key constraints) and the range of numerical attributes. Given a new partial session, the objective is to recommend potential query extensions, or aspects.

3.1 Proposed Work

To formulate an adequate solution, the following challenges must be addressed:

1. Aspect Definition. There is no clear notion of "item" or aspect in this setting. Instead, we need to find an adequate set of aspects that can be used to to capture user intent and characterize queries. Given the exploratory nature of queries in the scientific domains, the aspects should enable both schema navigation and data space exploration.

2. Sequential Aspects and Domain-Specific Constraints. Individual elements in a SQL query are sequential and there is dependency between them. For instance, in `SELECT T.A FROM T WHERE X > 10`, the domain of variable X is attributes in table T. Thus, given partial query, only a subset of the aspects are syntactically valid. Queries in the same session, are also submitted sequentially.

3. Session and Aspect Sparsity. In SDSS, the typical session has six SQL queries and lasts thirty minutes [21] which indicates aspect sparsity in queries and sessions.

The relational database setting exhibits some similarities to standard recommendation domains (e.g., movie): Some aspects, e.g., tables, attributes, data regions, are popular while the majority of them are unpopular. Some sessions are frequent, i.e., many queries are submitted, while the majority are infrequent. Scalability and responsiveness is important in both domains.

Analogous to our work in Section 2, our main hypothesis is that merely recommending popular aspects is not sufficient for exploratory tasks. Although popular aspects can help familiarize novice users with concepts like the important tables and attributes, given the exploratory nature of queries in IDE, recommendations are deemed more useful if they can help users narrow down their queries and expose relevant data regions. For example, recommending a specific interval like $b_1 <$ `BRIGHTNESS` $< b_2$ is more useful than just suggesting the attribute `BRIGHTNESS`.

Based on these intuitions, we will focus on recommending interesting aspects that enable data exploration and schema navigation for users of a relational database, and in particular, in IDE settings. Using the query log and the database synopsis, we will devise a set of aspects that include not just the relations, attributes, and user-defined functions, but also *intervals* of numeric attributes, e.g., $b_1 <$ `BRIGHTNESS` $< b_2$. Subsequently, we can use a vector-based query representation model where each element denotes the presence of a certain aspect. Alternatively, a graph-based representation [23] might be more suitable. After formulating similarity measures between queries (or sessions) [2], we can use a nearest neighbour model to suggest relevant aspects to the user.

In contrast to prior work that focuses on supervised learning and query rewriting [7], we focus on aspect definition and extraction. In contrast to [4, 7, 8], we rely on the database synopsis only. Accessing a large scientific database like SDSS to retrieve the entire set of tuples is expensive. In contrast to [14] our recommendations include intervals not just tables and attributes. The intermediate query format in [18] is complementary to our work.

4. FACILITATING QUERY ANSWER ANALYSIS

After users have successfully submitted a query, their next challenge is to analyse and understand the query answers. When the answer set is small, this task is attainable. The challenge is in examining and interpreting large, or even conflicting, answer sets.

To illustrate the problem, consider again climate data that is reported by various sources and integrated in relational databases. Because the sources were independently created and maintained, a given data point can have multiple, inconsistent values across the sources. For example, one source may have the high temperature for Vancouver on 06/11/2006 as 17C, while another may list it as 19C. As a result of this *value-level* heterogeneity, an aggregate query such as `SELECT AVG(TEMPERATURE)` does not have a single *true* answer. Instead, depending on the choice of data source combinations that are used to answer the query, different answers can be generated. Reporting the entire set of answers can overwhelm the user. Here, mechanisms that summarize the results and help the user understand query answers are required. Therefore, we study the following research question:

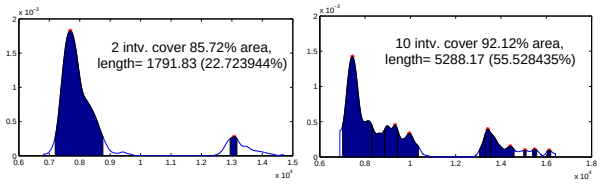
> After a query has been submitted to the system, how can we help the user understand and interpret the query answers?

Specifically, we address the problem of helping users understand aggregate query answers in integration contexts where data is segmented across several sources. We assume meta-information that describes the mappings and bindings between data sources is available [25].Our main concern is how to handle the value-level heterogeneity that exists in the data, to enable the user to better understand the range of possible query answers.

4.1 Solution

In our solution [30], we represented the answer to the aggregate query as an *answer distribution* instead of a single scalar value. We then proposed a suite of methods for extracting statistics that convey meaningful information about the query answers. We focused on the following challenges 1. determining which statistics best represent and answer's distribution 2. efficiently computing the desired statistics. In deriving our algorithms, we assumed prior knowledge regarding the sources is unavailable and all sources are equal.

A *high coverage interval* is one of the statistics we extract to convey the shape of the answer distribution and



(a) S_1 (b) S_4

Figure 1: High coverage intervals tell where the majority of answers can be found.

the intervals where the majority of viable answers can be found. Figure 1 shows the multi-modal answer distributions of the aggregate query `AVG(TEMP)`, on Canadian climate data (S_1) [5] and synthetic data (S_4) [30], and their corresponding high coverage intervals.

5. SUMMARY AND OUTLOOK

The goal of my thesis is to devise techniques that facilitate user interaction with data. I address three aspects:

- (Accomplished) Facilitating data exploration with recommender systems in standard domains (Section 2).
- (In progress) Facilitating data exploration and query composition in the relational database context (Section 3). I am currently working on extracting a dataset, and narrowing down the problem statement.
- (Accomplished) Facilitating query answer analysis by extracting statistics and semantics about the range of query answers (Section 4).

6. REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. Improving aggregate recommendation diversity using ranking-based techniques. *TKDE*, 24(5):896–911, 2012.

[2] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for olap sessions. *Knowledge and information systems*, 39(2):463–489, 2014.

[3] Pablo Castells, Neil J. Hurley, and Saul Vargas. *Recommender Systems Handbook*, chapter Novelty and Diversity in Recommender Systems. Springer US, 2015.

[4] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Query recommendations for interactive database exploration. In *International Conference on Scientific and Statistical Database Management*, pages 3–18. Springer, 2009.

[5] Climate Canada. Canada climate data. `http://climate.weatheroffice.gc.ca/climateData/canada_e.html`, 2010.

[6] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, 2010.

[7] Julien Cumin, Jean-Marc Petit, Vasile-Marian Scuturici, and Sabina Surdu. Data exploration with sql using machine learning techniques. In *EDBT*, 2017.

[8] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 517–528. ACM, 2014.

[9] Simon Dooms, Toon De Pessemier, and Luc Martens. Movietweetings: a movie rating dataset collected from twitter. In *CrowdRec at RecSys*, 2013.

[10] Yu-Chieh Ho, Yi-Ting Chiang, and Jane Yung-Jen Hsu. Who likes it more?: mining worth-recommending items from long tails by modeling relative preference. In *WSDM*, pages 253–262, 2014.

[11] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[12] HV Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24. ACM, 2007.

[13] Martin L Kersten, Stratos Idreos, Stefan Manegold, Erietta Liarou, et al. The researchers guide to the data deluge: Querying a scientific database in just a few seconds. *PVLDB Challenges and Visions*, 3, 2011.

[14] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. Snipsuggest: context-aware autocompletion for sql. *Proceedings of the VLDB Endowment*, 4(1):22–33, 2010.

[15] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local collaborative ranking. In *WWW*, pages 85–96, 2014.

[16] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. *arXiv preprint arXiv:1205.3193*, 2012.

[17] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[18] Hoang Vu Nguyen, Klemens Böhm, Florian Becker, Bertrand Goldman, Georg Hinkel, and Emmanuel Müller. Identifying user interests within the data space-a case study with skyserver. In *EDBT*, pages 641–652, 2015.

[19] M Jordan Raddick, Ani R Thakar, Alexander S Szalay, and Rafael DC Santos. Ten years of skyserver i: Tracking web and sql e-science usage. *Computing in Science & Engineering*, 16(4):22–31, 2014.

[20] Information removed for double-blind review. Submitted paper, 2017.

[21] Vik Singh, Jim Gray, Ani Thakar, Alexander S Szalay, Jordan Raddick, Bill Boroski, Svetlana Lebedeva, and Brian Yanny. Skyserver traffic report-the first five years. *arXiv preprint cs/0701173*, 2007.

[22] Saúl Vargas and Pablo Castells. Improving sales diversity by recommending users to items. In *RecSys*, 2014.

[23] Roy Villafane, Kien A Hua, Duc Tran, and Basab Maulik. Mining interval time series. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 318–330. Springer, 1999.

[24] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. Maximum margin matrix factorization for collaborative ranking. *Advances in neural information processing systems*, pages 1–8, 2007.

[25] Jian Xu and Rachel Pottinger. Integrating domain heterogeneous data sources using decomposition aggregation queries. *Information Systems*, 39(0), 2014.

[26] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. Challenging the long tail recommendation. *PVLDB*, 5(9):896–907, 2012.

[27] Weinan Zhang, Jun Wang, Bowei Chen, and Xiaoxue Zhao. To personalize or not: a risk management perspective. In *RecSys*, pages 229–236, 2013.

[28] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *RecSys*, pages 249–256, 2013.

[29] Sedigheh Zolaktaf and Gail C Murphy. What to learn next: recommending commands in a feature-rich environment. In *ICMLA*, pages 1038–1044. IEEE, 2015.

[30] Zainab Zolaktaf, Jian Xu, and Rachel Pottinger. Extracting aggregate answer statistics for integration. *EDBT*, 2015.

Processing Moving Object Data Streams with Data Stream Management Systems

Tobias Brandt
Supervised by Marco Grawunder
University of Oldenburg, Germany
tobias.leo.brandt@uol.de

ABSTRACT

With the wide spread of cheap and mobile GPS sensors as well as mobile data connections, live streams from moving objects are becoming a huge data source. The services based on these data streams, for example, for connected cars, vessels or smartphone users, need real-time results for queries based on the current or even near-future positions of the moving objects. Spatio-temporal data from moving objects cannot just be treated as a crowd of points with timestamps, but must be seen as points in a trajectory with non-measured points in between. In this paper I present my work on the management of such real-time trajectories within a Data Stream Management System (DSMS) to enable simple, flexible and efficient in-memory moving object query processing.

1. INTRODUCTION

Moving objects in the real world are everywhere and have been there for a while: pedestrians and cars on the streets, vessels on the oceans and airplanes in the sky. Comparably new is the huge amount of data these objects are producing. Many of these send their location regularly to a central server or other facility, may it be the smartphone user with a Location-based Service (LBS) or a vessel with an Automatic Identification System (AIS) sender.

This data can be used to answer questions and solve real-world problems. For example, vessels can be warned about congested or currently dangerous areas based on their own position as well as the positions of other vessels. As more data can be shared via live-streams and as the results of such queries are required with minimal delay, traditional systems that first store and then query the data streams are not ideal. They typically run short-term queries on static data sets that are stored on the hard drive.

Data Stream Management Systems (DSMSs) especially target data streams. They offer solutions for many data stream related challenges, provide query languages to define queries without the need to write code in a general purpose programming language and simplify the connection to typical data sources. Maintenance of queries on data streams is made simple due to the ease to change and update the query text. Hence, queries can be adapted to new requirements quickly. These features make them a useful tool to easily create and change queries for many different use cases and are therefore a good choice for rapid prototyping systems in the field of data stream processing and analysis.

To cope with the requirements of data streams, DSMSs support continuous queries and use a data-driven approach. New results are incrementally calculated when new data arrives at the system. This increases the demand for quick calculations, wherefore data is typically kept in-memory. Unfortunately, a potentially infinite data stream cannot be hold in-memory. A typical solution DSMSs provide are windows. These reduce the amount of data hold in-memory to a smaller part of the data streams, e. g., all elements from the last hour or the last 100 elements.

Even though DSMSs already tackle lots of the challenges that occur with data stream processing, they lack features necessary to work with moving objects data. Two of these features are (1) continuous location interpolation and near-future prediction and (2) fast moving objects index structures for windows with high fluctuation. In this work, I concentrate on point data, hence, moving regions are not in the scope. That is because most moving object data of interest today, such as vessels and pedestrians, can be simplified to point objects without loosing too much precision in the queries. Moving or evolving regions in contrast introduce a whole new palette of challenges.

One important feature of moving objects data streams is that the objects move continuously but are only measured once in a while. Therefore, the objects have unknown locations in between the measurements, which can and sometimes need to be used for querying. Imagine a query where a vessel needs to know all vessels around it. Another vessel, which last known location is (temporal and spatial) far away but will probably be within that range on querytime, should be included in the answer, hence, its location needs to be automatically predicted to the future by the query. This scenario is particularly important for satellite AIS where it is normal to have hours between location updates [3]. Streaming data differs from static time-series data: in static data, all measured locations of a moving object are known. In contrast, in a streaming environment, it is not possible to know if and when the next location update of a moving object will arrive.

Challenges with this approach are that interpolation and

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

prediction depends on the use case and always comes with an uncertainty. When new data about an interpolated object is available, old query results may need to be updated. When and how to update results by more precise ones is a non-trivial question within a DSMS. To my best knowledge, there is no work that tackles these challenges in the field of data streams for moving objects.

The second feature mentioned above are moving objects windows and suitable index structures. As not all data can be stored, it needs to be decided which data is still needed for processing and which is not and how and when old data can be wiped. Typical window concepts such as time-based windows can be extended by windows especially for moving objects. A possible window type could be a distance-based window. It would store all data within a certain distance of a single moving object, e. g., the last kilometer of every object. The requirements for the underlying index structures differ both from pure temporal as well as pure spatial index structures.

2. OBJECTIVES AND CHALLENGES

The overall goal of this work is to create and implement concepts to allow DSMSs to process spatio-temporal data from moving objects. To reach this goal, I am tackling the challenges of including location interpolation and prediction as well as window concepts for moving object data streams.

2.1 Objectives

- **Location Interpolation** Moving objects naturally move continuously but are only measured once in a while. Within a data stream, they seem to be hopping from point to point, resulting in delayed and possibly wrong results. I aim to introduce location interpolation into the data stream processing to that the objects move continuously in time and space. The interpolation should also be used for short-term location prediction.
- **Result Uncertainty** As prediction always comes with a degree of uncertainty, the results are uncertain as well and can change when new data from a moving objects arrives. The accuracy or uncertainty of a result should be transparent to the user and updates of results should be possible.
- **Moving Object Windows** Window definitions that are especially useful for moving objects data should be introduced.
- **Spatio-temporal Indexes** The data within the windows need to be hold in spatio-temporal index structures optimized for high data fluctuation.

The concepts created to solve the goals should be evaluated by implementing them into an open source DSMS. Scenarios with AIS data from vessels should show that the concepts work with real-world data and queries.

2.2 Challenges for Location Interpolation

Typical spatio-temporal queries include neighborhood and range queries. An example query could be "Continuously report all vessels within a range of 10 km around vessel X". Such a query is depicted in Figure 1. The orange vessel in the middle sent its last location at 12 o'clock and wants to know which vessels are in its range at that point in time. The trajectory of the other vessels one to three are visualized with the arrows and circles. The circles are the measurements where the correct location of the vessel is known. The arrows in between visualize a simple interpolation. It is assumed that the path between the measurements is straight. That is not necessarily the case but it is a reasonable and simple approximation. The dashed lines with the striped circles are predictions of the future trajectory.

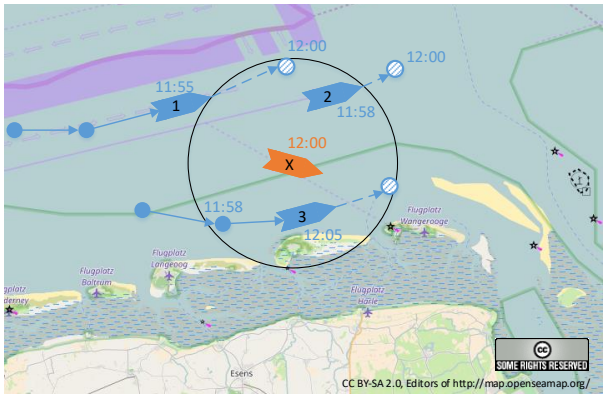


Figure 1: Range query for the orange vessel.

The need to interpolate and predict locations arises from the fact that the moving objects do not measure and send their location synchronized with each other, but at different time intervals. In the figure, the last known location of the orange vessel in the middle was captured at 12 o'clock. For this point in time, the locations of all other vessels are needed to answer the query correctly. Unfortunately, the known, i. e., measured, locations of the other vessels are not at 12 o'clock, but slightly before or after this point in time. If only the last known location would be used to answer the query, the result would be wrong: Vessel 1 would not be within the result but should be, Vessel 2 would be within the result but should not and the location of Vessel 3 would be wrong.

Hence, interpolation and prediction is necessary to answer the query approximately correct. When doing so, a few challenges arise. The interpolation has to work in an incremental manner with limited knowledge about the data, as not all past data can be stored in a streaming environment. The accuracy of a query result needs to be known to the user or further processing steps. When the prediction was wrong, old results for a query may need to be updated (e. g., if Vessel 1 takes a different path than predicted). In a streaming scenario, the approximate result may already be used for further processing or a following result, for example, for 12:03, has already been processed. The questions on how to integrate uncertain results and updates to them within a DSMS need to be tackled.

2.3 Challenges for Moving Object Windows

Windows are necessary to reduce the infinite data stream to a finite set of data. The data within a window can be kept in-memory and never needs to be permanently written to a hard drive. Next to the performance improvements,

the concept of windows has more benefits. They are based on the assumption that in many cases, queries are only or mostly interested in the current data and not in the data from the distant past. To define a window according to the use case, the domain needs to be known. A typical window could, for example, hold all data from the last hour.

In the domain of moving objects, the requirements for windows can differ depending on the scenario. The definition of a window only by the time and not by the space dimension is often not enough. Imagine, for example, a window where the speed of all objects within a data stream is diverse and variable (e. g., slow vessels and fast planes within one stream). It could be necessary to have from each object at least the last kilometer within the window. With a time-based window this would be difficult to achieve. Additionally, compression could be introduced to moving objects windows. Patroumpas et al. [7] show that trajectory data can be compressed without loosing much accuracy. Hence, new window concepts for moving objects are useful.

Windows reduce the amount of the required (in-memory) storage space. Nevertheless, it opens up new questions about how to clean up the memory, for example, when to delete old data. This question gets more complicated as window indexes have to be shared between multiple queries. Imagine a data stream of all vessels on the North Sea. As spatial queries can be heavily accelerated when using an index, the data in the windows are indexed. Thereby, due to memory limitations, creating multiple nearly identical indexes must be avoided. Subsequently, one index is shared between multiple queries. Nevertheless, such a sharing makes the decision when to delete old data more complicated, as the window requirements from the queries can be different.

Additionally, not every index structure is suitable for a spatio-temporal data stream index. In contrast to more traditional Geographic Information System (GIS) applications, the fluctuation in the data is very high. New data needs to be inserted and at the same time old data needs to be removed on a high frequency. It is possible that the whole set of data within a window can be swapped within minutes or even seconds, which, for example, distinguishes the requirements from static time-series data. Traditional index-structures that require heavy reorganization when data gets changed are probably not suitable for this environment. In this work it needs to be evaluated if index structures for this purpose are useful as it is only an improvement if the indexing needs less time than it saves while querying the data.

In this PhD project I aim to create suitable window concepts, implement them and choosing an efficient index structure for this very dynamic environment.

3. RESEARCH PLAN

In this section, the main approaches to overcome the challenges from above are described.

3.1 Approach

Query processing on data streams is typically done with operator graphs. The elements of the data stream are send from one operator to the next, each operator doing a specific task. Joins, projections and selections are typical examples for such operators.

When adding support for moving objects data, this modular architecture should be exploited. New operators can implement the spatio-temporal operations. While doing so, they have to behave like normal operators to the outside so that other operators can seamlessly use the output. Two example operators that are needed are a range and a k-nearest neighbors (kNN) operator. Both search for other moving objects close to a certain object. The external behavior of these operators is similar to other operators. They receive stream elements, process them and send their results as stream elements to the next operator.

Internally, these operators need to use location interpolation and prediction to compute correct results and annotate these results with the level of (un)certainty in the meta data of a streaming object. The interpolation should be done by a framework within the DSMS that allows to interchange algorithms, as the interpolation algorithm can change from case to case.

3.2 Current and Future Work

Currently, prototype versions for moving object range and kNN queries are available. A prototype implementation within a DSMS was developed. For spatial querying with moving object windows, an index structure based on GeoHashes [6] was developed. It showed better performance than an implementation based on QuadTrees. This could be due to better insertion and deletion performance with the GeoHash implementation. However, these results are very preliminary, as the test setup needs to be better described and results need to be analyzed further to find out the reasons for the differences.

The correct integration of those queries as well as moving object windows is ongoing work. The results are currently only partly usable for other query operators. Interpolating and predicting locations are in the concept phase. Development and implementation of these will be a major part of the future work.

3.3 Planned Evaluation

The concepts that are created in this PhD project will be implemented into the open source DSMS *Odysseus*[1] [1]. Odysseus offers a rich set of operators and a query language. For the purpose of this work it already supports protocols used in the maritime domain such as AIS and can talk to common data sources such as RabbitMQ out of the box. In contrast to streaming frameworks such as Apache Flink[2] or Heron[3], it is not necessary to program in a general programming language such as Java to create new queries.

With that implementation, the feasibility of the concepts will be evaluated. Using an iterative approach, the concepts can be adjusted if uncovered challenges occur while evaluating. The implementation will be used with scenarios in the maritime context, especially with AIS data. An example query could be to continuously query if a vessel is heading to an area that will be congested during its transit.

For the given scenarios with moving objects, timely query results are necessary, wherefore the performance of the solutions will be measured. The latency and throughput of the queries will be used to compare different implementations, e. g., different approaches for spatio-temporal indexes.

[1] `http://odysseus.offis.uni-oldenburg.de/`, last accessed on 03/21/2017

[2] `https://flink.apache.org/`, last accessed on 05/24/2017

[3] `https://twitter.github.io/heron/`, last accessed on 05/24/2017

4. RELATED WORK

General purpose and open source streaming systems such as Apache Flink and Apache Storm[4] as well as commercial systems such as IBM InfoSphere Streams[5] (short: IBM Streams) offer high performance and distributed stream processing, but have only limited support for moving objects. While spatio-temporal data is supported in some systems (e. g., with IBM Streams [2]), location interpolation and moving object windows are, as of my knowledge, not.

Zhang et al. [9] use Apache Storm to process fast data streams from moving objects. They focus on a distributed spatial index which speeds up range and kNN queries as well as spatial joins. One main difference to this work is that they do not interpolate and predict locations to have temporal correct results.

The open source project GeoMesa[6] works with spatio-temporal data, e. g., from moving objects [6]. The project develops indexes based on space-filling curves. These allow quick access to spatial or spatio-temporal data within sorted key-value stores such as Apache Accumolo[7]. GeoMesa does not specifically address streaming data, data stream management capabilities or location interpolation.

The RxSpatial library [8] is an extension for the Microsoft SQL Server Spatial Library and adds support for moving objects. It adds the RUM-tree, which is an extension of the R-tree for frequent updates. Additionally, RxSpatial allows continuous spatial queries, e. g., to observe if a moving object is close to another. As of my best knowledge, the library does not take into account the time of the updates but uses the newest updates of every moving object. Interpolation and prediction are not used.

Interpolation for moving objects is a difficult challenge for moving regions (e. g., described by Heinz et al. [5]). This is especially complex as these regions can change their shape over time. In this work I want to concentrate on moving points, which is a way simpler version of that problem. Nevertheless, the perfect interpolation method is not the goal of this work but the integration of interpolation and prediction into the stream processing of moving objects data.

Secondo [4] is a database system especially for moving objects. It has a spatial and temporal algebra with which queries for moving objects can be formulated. As it is a database, it is not optimized for data streams, e. g., it does not support windows, does not run mainly in-memory and hence does not have to solve the problem of cleaning up old data. Nevertheless, it gives useful insights on the handling of moving objects data.

Patroumpas et al. [7] use AIS data in a streaming environment to detect complex events such as unexpected stops of vessels. They compress the data to important points in the trajectory of the vessels without loosing much accuracy. They also address errors in the AIS data by removing wrong measurements. In contrast to this work, they do not use location prediction for vessels that did not send an update for a while, which is for example the case for satellite AIS.

[4] `https://storm.apache.org/`, last accessed on 03/21/2017
[5] `https://www.ibm.com/analytics/us/en/technology/stream-computing/`, last accessed on 03/21/2017
[6] `http://www.geomesa.org`, last accessed on 03/17/2017
[7] `https://accumulo.apache.org/`, last accessed on 03/21/2017

5. CONCLUSION

This paper describes the motivation, challenges and approaches of processing data from moving objects in DSMSs. A major challenge are asynchronous updates of locations of multiple moving objects. To serve timely query results, e. g., for a range query, locations of objects need to be interpolated and predicted. The integration of interpolated values into a DSMS provides some challenges that are tackled with this PhD project. First approaches to solve these are explained. The planned evaluation uses AIS data from vessels for continuous queries.

6. ACKNOWLEDGMENTS

We thank the Ministry of Science and Culture of Lower Saxony, Germany for supporting us with the graduate school *Safe Automation of Maritime Systems (SAMS)*.

7. REFERENCES

[1] H.-J. Appelrath, D. Geesen, M. Grawunder, T. Michelsen, and D. Nicklas. Odysseus: A highly customizable framework for creating efficient event stream management systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 367–368, New York, NY, USA, 2012. ACM.

[2] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1093–1104, New York, NY, USA, 2010. ACM.

[3] M. A. Cervera, A. Ginesi, and K. Eckstein. Satellite-based vessel automatic identification system: A feasibility and performance analysis. *International Journal of Satellite Communications and Networking*, 29(2):117–142, 2011.

[4] R. H. Güting, T. Behr, and C. Düntgen. *Secondo: A platform for moving objects database research and for publishing and integrating research implementations.* Fernuniv., Fak. für Mathematik u. Informatik, 2010.

[5] F. Heinz and R. H. Güting. Robust high-quality interpolation of regions to moving regions. *Geoinformatica*, 20(3):385–413, July 2016.

[6] H. V. Le. Distributed moving objects database based on key-value stores. In *Proceedings of the VLDB 2016 PhD Workshop co-located with the 42nd International Conference on Very Large Databases (VLDB 2016), New Delhi, India, September 9, 2016.*, 2016.

[7] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2):389–427, 2017.

[8] Y. Shi, A. M. Hendawi, H. Fattah, and M. Ali. Rxspatial: Reactive spatial library for real-time location tracking and processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2165–2168. ACM, 2016.

[9] F. Zhang, Y. Zheng, D. Xu, Z. Du, Y. Wang, R. Liu, and X. Ye. Real-time spatial queries for moving objects using storm topology. *ISPRS International Journal of Geo-Information*, 5(10), 2016.

Symmetric and Asymmetric Aggregate Function in Massively Parallel Computing

ZHANG Chao
Supervised by Prof. Farouk Toumani, Prof. Emmanuel GANGLER

LIMOS, Université Clermont Auvergne, Aubière, France
{zhangch, ftoumani}@isima.fr

ABSTRACT

Applications of aggregation for information summary have great meanings in various fields. In big data era, processing aggregate function in parallel is drawing researchers' attention. The aim of our work is to propose a generic framework enabling to map an arbitrary aggregation into a generic algorithm and identify when it can be efficiently executed on modern large-scale data-processing systems. We describe our preliminary results regarding classes of symmetric and asymmetric aggregation that can be mapped, in a systematic way, into efficient MapReduce-style algorithms.

1. INTRODUCTION

The ability to summarize information is drawing increasing attention for information analysis [11, 6]. Simultaneously, under the progress of data explosive growth processing aggregate function has to experience a transition to massively distributed and parallel platforms, e.g. Hadoop MapReduce, Spark, Flink etc. Therefore aggregation function requires a decomposition approach in order to execute in parallel due to its inherent property of taking several values as input and generating a single value based on certain criteria. Decomposable aggregation function can be processed in a way that computing partial aggregation and then merging them at last to obtain final results.

Decomposition of aggregation function is a long-standing research problem due to its benefits in various fields. In distributed computing platforms, decomposability of aggregate function can push aggregation before shuffle phase [17, 3]. This is usually called initial reduce, with which the size of data transmission on a network can be substantially reduced. For wireless sensor network, the need to reduce data transmission is more necessary because of limitation of power supply [15]. In online analytical processing (OLAP), decomposability of aggregate function enables aggregation across multi-dimensions, such that aggregate queries can be executed on pre-computation results instead of base data to accelerate query answering [8]. An important point of query optimization in relational databases is to reduce table size for join [10], and decomposable aggregation brings interests [4].

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

When an arbitrary aggregation function is decomposable, how to decompose it and when a decomposition is 'efficient' is a hard nut to crack. Previous works identify interesting properties for decomposing aggregation. A very relevant classification of aggregation functions, introduced in [11], is based on the size of sub-aggregation (i.e., partial aggregation). This classification distinguishes between distributive and algebraic aggregation having sub-aggregation with fixed sizes, and holistic functions where there is no constant bound on the storage size of sub-aggregation. Some algebraic properties, such as associativity and commutativity, are identified as sufficient conditions for decomposing aggregation [17, 3]. Compared to these works, our work provides a generic framework to identify the decomposability of any symmetric aggregation and generate generic algorithms to process it in parallel. Moreover, all but few researches in the literature consider symmetric functions. Asymmetric aggregation is inherently non-commutative functions and this makes their processing in parallel and distributed environment far from being easy. In [16], a symbolic parallel engine (SYMPLE) is proposed in order to automatically parallelize User Defined Aggregations (UDAs) that are not necessarily commutative. Although interesting, the proposed framework lacks guarantees for efficiency and accuracy in the sense that it is up to users to encode a function as SYMPLE UDA. Moreover, symbolic execution may have path explosion problem.

My research focuses on designing generic framework that enables to map symmetric and asymmetric aggregation functions into efficient massively parallel algorithms. To achieve this goal, we firstly identify a computation model, and an associated cost model to design and evaluate parallel algorithms. We consider MapReduce-style (MR) framework and use the MRC [12] cost model to define *'efficient'* MR algorithms. We rest on the notion of well-formed aggregation [4] as a canonical form to write symmetric aggregation and provide a simple and systematic way to map well-formed aggregation function α into an MR algorithm, noted by $MR(\alpha)$. Moreover, we provide reducible properties to identify when the generated $MR(\alpha)$ is efficient (when $MR(\alpha)$ is an MRC algorithm). Then we extend our framework to a class of asymmetric aggregation function, position-based aggregation, and propose extractable property to have generic MRC algorithms. Our main results are Theorem 1 and Theorem 2, of which proofs are provided in an extended report[2].

2. MRC ALGORITHM

Several research works concentrate on the complexity of parallel algorithms. MUD[7] algorithm was proposed to

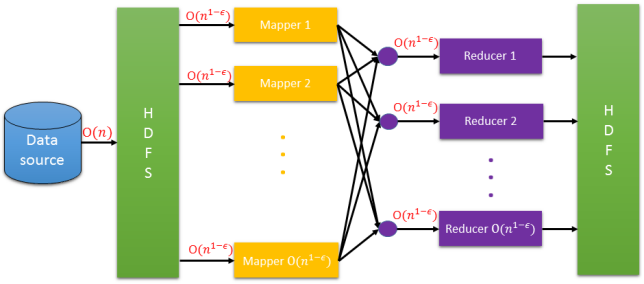


Figure 1: MapReduce flowchart with MRC constraints

transform symmetric streaming algorithms to parallel algorithms with nice bounds in terms of communication and space complexity, but without any bound on time complexity. This disqualifies MUD as a possible candidate cost model to be used in our context. MRC[12] is another popular model that has been used to evaluate whether a MapeReduce algorithm is efficient. The constraints enforced by MRC w.r.t. total input data size can be summarized as following: sublinear number of total computing nodes, sublinear space for any mapper or reducer, polynomial time for any mapper or reducer, and logarithm round number. We illustrate these constraints besides round number in a simplified MapReduce flowchart in figure 1 where $\epsilon > 0$.

Hence, the MRC model considers necessary parameters for parallel computing, communication time, computation space and computing time, and makes more realistic assumptions. A MapReduce algorithm satisfying these constraints is considered as an efficient parallel algorithm and will be called hereafter an MRC algorithm.

3. SYMMETRIC AGGREGATION WITH MRC

Let I be a doamin, an *n-ary aggregation* α is a function[9]: $I^n \rightarrow I$. α is symmetric or commutative[9] if $\alpha(X) = \alpha(\sigma(X))$ for any $X \in I$ and any permutation σ, where $\sigma(X) = (x_{\sigma(1)}, ..., x_{\sigma(n)})$. Symmetric aggregation result does not depend on the order of input data, therefore input is considered as a *multiset*. In this section, we define a generic framework to map symmetric aggregation into an MRC algorithm.

3.1 A Generic Form for Symmetric Aggregation

To define our generic aggregation framework, we rest on the notion of well-formed aggregation [4]. A symmetric aggregation α defined on a multiset $X = \{d_1, \ldots, d_n\}$ can be written in well-formed aggregation as following:

$$\alpha(X) = T(F(d_1) \oplus \ldots \oplus F(d_n)),$$

where F is translating function(tuple at a time), $\oplus$ is a commutative and associative binary operation, and T is terminating function. For instance, *average* can be easily transformed into well-formed aggregation: $F(d) = (d, 1)$, $(d, k) \oplus (d', k') = (d + d', k + k')$ and $T((d, n)) = \frac{d}{n}$. In fact, any symmetric aggregation can be rewritten into well-formed aggregation with a flexible choice of $\oplus$, e.g $\oplus = \cup$.

Well-formed aggregation provides a generic plan for processing aggregate function in distributed architecture based on the associative and commutative property of $\oplus$: processing F and $\oplus$ at mapper, $\oplus$ and T at reducer. Table 1 depicts the corresponding generic MapReduce(MR) algorithm(the case of one key and trivially extending to any number of keys), noted by $MR(\alpha)$, where mapper input is a submultiset X_i of X and mapper output is o_i, and $\sum_{\oplus}$ is the concatenation of $\oplus$.

Table 1: $MR(\alpha)$: a generic MR aggregation algorithm

	operation
mapper	$\sum_{\oplus, d_j \in X_i} F(d_j)$
reducer	$T(\sum_{\oplus, i} o_i)$

However, the obtained $MR(\alpha)$ are not necessarily an efficient MapReduce algorithm. We identify when $MR(\alpha)$ is a MRC algorithm using *reducibility* property.

Definition 1. A symmetric aggregation function α defined on domain I is reducible if the well-formed aggregation $(F, \oplus, T)$ of α satisfies

$$\forall d_i, d_j \in I : |F(d_i) \oplus F(d_j)| = O(1).$$

With this reducible property, we provide a theorem identifying when $MR(\alpha)$ of a symmetric aggregation is a MRC algorithm.

THEOREM 1. *Let α be a symmetric well-formed aggregation and $MR(\alpha)$ be the generic algorithm for α, then $MR(\alpha)$ is an MRC algorithm if and only if α is reducible.*

3.2 Deriving MRC Algorithm from Algebraic Properties

In this section, we investigate several **symmetric aggregation** properties satisfying Theorem 1. If an aggregation α is in one of the following classes, then α has an $MRC(\alpha)$ algorithm illustrated in table 1.

An aggregate function α is *associative* [9] if for multiset $X = X_1 \cup X_2$, $\alpha(X) = \alpha(\alpha(X_1), \alpha(X_2))$. **Associative and symmetric** aggregation function can be transformed in well-formed aggregation $(F, \oplus, T)$ as following,

$$F = \alpha, \ , \oplus = \alpha, \ T = id \tag{1}$$

where id denotes identity function. α is reducible because it is an aggregation. Therefore $MR(\alpha)$ of associative and symmetric aggregation α is an MRC algorithm.□

An aggregation α is *distributive* [11] if there exists a combining function C such that $\alpha(X, Y) = C(\alpha(X), \alpha(Y))$. **Distributive and symmetric** aggregation can be rewritten in well-formed aggregation $(F, \oplus, T)$ as following,

$$F = \alpha, \ \oplus = C, \ T = id. \tag{2}$$

Similarly, α is reducible and corresponding $MR(\alpha)$ is an MRC algorithm.□

Another kind of aggregate function having the same behavior as symmetric and distributive aggregation is **commutative semigroup aggregate function** [5]. An aggregation α is in this class if there exists a commutative semigroup $(H, \otimes)$, such that $\alpha(X) = \bigotimes_{x_i \in X} \alpha(x_i)$. The corresponding well-formed aggregation $(F, \oplus, T)$ is illustrated as following,

$$F = \alpha, \ \oplus = \otimes, \ T = id. \tag{3}$$

It is clearly that α is reducible and $MR(\alpha)$ is an MRC algorithm.□

A more general property than commutative semi-group aggregation is symmetric and preassociative aggregate function. An aggregation α is *preassociative* [13] if it satisfies $\alpha(Y) = \alpha(Y') \implies \alpha(XYZ) = \alpha(XY'Z)$. According to [13], some **symmetric and preassociative**(unarily quasi-range-idempotent and continuous) aggregation functions can be constructed as $\alpha(\mathbf{X}) = \psi\left(\sum_{i=1}^{n}\varphi(x_i)\right), n \geq 1$, where ψ and φ are continuous and strictly monotonic function. For instance, $\alpha(X) = \sum_{i=1}^{n} 2 \cdot x_i$, where $\psi = id$ and $\varphi(x_i) = 2 \cdot x_i$. The well-formed aggregation $(F, \oplus, T)$ for this kind of preassociative aggregation is illustrated as following

$$F = \varphi,\ \oplus = +,\ T = \psi. \tag{4}$$

The corresponding $MR(\alpha)$ is also an MRC algorithm.□

An aggregate function α is barycentrically associative [14] if it satisfies $\alpha(XYZ) = \alpha(X\alpha(Y)^{|Y|}Z)$, where $|Y|$ denotes the number of elements contained in multiset Y and $\alpha(Y)^{|Y|}$ denotes $|Y|$ occurrences of $\alpha(Y)$. A well-known class of symmetric and barycentrically associative aggregation is **quasi-arithmetic mean** : $\alpha(\mathbf{X}) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(x_i)\right), n \geq 1$, where f is an unary function and f^{-1} is a quasi-inverse of f. With different choices of f, α can be different kinds of mean functions, e.g arithmetic mean, quadratic mean, harmonic mean etc. It is trivial to rewrite this kind of aggregation into well-formed aggregation $(F, \oplus, T)$ and the $MR(\alpha)$ is also an MRC algorithm,

$$F = (f, 1),\ \oplus = (+, +),\ T = f^{-1}(\frac{\sum_{i=1}^{n} f(x_i)}{n}). \tag{5}$$

4. ASYMMETRIC AGGREGATION

Many commonly used aggregation function is symmetric(commutative) such that the order of input data can be ignored, while *asymmetric aggregation considers the order*. Two common asymmetric cases could be weighted aggregation and cumulative aggregation, where aggregated result will be changed if data order is changed, e.g. WMA(weighted moving average) and EMA(exponential moving average)[1], which are used to highlight trends.

4.1 A Generic Form for Asymmetric Aggregation

In contrast to symmetric aggregation, asymmetric function is impossible to rewrite into well-formed aggregation, because translating function F is a tuple at a time function and $\oplus$ is commutative and hence both of them are insensitive to the order. For this reason, we propose an extended form based on well-formed aggregation which is more suitable for asymmetric aggregation.

Definition 2. An asymmetric aggregation α defined on an ordered sequence $\bar{X}$ is an asymmetric well-formed aggregation if α can be rewritten as following,

$$\alpha(\bar{X}) = T(F^o(\bar{X}, x_1) \oplus ... \oplus F^o(\bar{X}, x_n)), \tag{6}$$

where F^o is order-influenced translating function, $\oplus$ is a commutative and associative binary operation, and T is terminating function.

For instance, $\alpha(\bar{X}) = \sum_{x_i \in \bar{X}}(1-z)^{i-1}x_i$[14] with a constant z can be rewritten as $F^o(\bar{X}, x_i) = (1-z)^{i-1}x_i,\ \oplus = +,\ T = id$, where i is the position of x_i in the sequence $\bar{X}$.

Asymmetric well-formed aggregation can rewrite any asymmetric aggregation α, and with the associative property of $\oplus$, α also has a generic MR algorithm $MR(\alpha)$: processing F^o and $\oplus$ at mapper, $\oplus$ and T at reducer. Similar to the behavior of symmetric well-formed aggregation, reducible property is needed to ensure MRC constraints. The reducible property for asymmetric well-formed aggregation is

$$\forall x_i, x_{i+1} \in \bar{X} :\ |F^o(\bar{X}, x_i) \oplus F^o(\bar{X}, x_{i+1})| = O(1).$$

However, in order to have a correct generic MRC algorithm for asymmetric aggregation, reducible property is not enough, because asymmetric function considers data order such that operations for combining mapper outputs are more than $\oplus$. We illustrate this problem and identify properties to have correct MRC algorithm for a class of asymmetric well-formed aggregation in the following.

4.2 Position-based Aggregation with MRC

We deal with a kind of asymmetric aggregation α called position-based aggregation, for which F^o is $F^o(\bar{X}, x_i) = h(i) \odot f(x_i)$, where $h()$ and $f()$ are unary functions, and $\odot$ is a binary operation. The corresponding asymmetric well-formed framework is $\alpha(\bar{X}) = T(\sum_{\oplus, x_i \in \bar{X}} h(i) \odot f(x_i))$, where $\sum_{\oplus}$ is the concatenation of $\oplus$.

Let $\bar{X}$ be an ordered sequence $\bar{X} = \bar{S_1} \circ ... \circ \bar{S_m}$, where $\bar{S_l}$ is a subsequence of $\bar{X}$, $l \in \{1, ..., m\}$ and $\circ$ is the concatenation of subsequence, and i be the holistic position of x_i in $\bar{X}$ and j be the relative position of x_j in subsequence $\bar{S_l}$. Then $\sum_{\oplus} F^o(\bar{X}, x_i)$ of α on any subsequence S_l is

$$\sum_{\oplus, x_i \in \bar{S_l}} F^o(\bar{X}, x_i) = \sum_{\oplus, x_j \in \bar{S_l}} h(j+k) \odot f(x_i),$$

where $j + k$ $(j + k = i)$ is the holistic position of the jth element x_j in $\bar{S_l}$. In order to process α in parallel on these subsequences, the first requirement is to have l, which means in distributed and parallel computing data set is split into ordered chunks and chunk indexes can be stored. It can be trivially implemented in Hadoop[16]. Secondly, k is needed, the number of elements before $\bar{S_l}$. Sequential distributing subsequence count values then starting aggregation is costly due to too many times of data transferring on network. If k can be extracted out of $\sum_{\oplus, x_j \in \bar{S_l}} h(j+k) \odot f(x_i)$, then α can be processed without distributing counts because operations relating to count can be pushed to reducer. We identify conditions to extract k which we call *extractable* property.

LEMMA 1. *Given an ordered sequence $\bar{X}$, a position-based asymmetric well-formed aggregation α defined in $(F^o, \oplus, T)$ and $F^o(\bar{X}, x_i) = h(i) \odot f(x_i)$ for any $x_i \in \bar{X}$, where $h()$ and $f()$ are unary functions, is **extractable** if there exists a binary operation $\otimes$ making $h()$ satisfy $h(i + k) = h(i) \otimes h(k + c)$ with a constant c, and $\oplus$, $\otimes$ and $\odot$ satisfy one of the following conditions,*

- *$\otimes$, $\odot$ and $\oplus$ are same,*
- *$\otimes$ and $\odot$ are same and they are distributive over $\oplus$,*
- *$\otimes$ is distributive over $\odot$ which is same as $\oplus$.*

The behavior of $h()$ is similar to group homomorphism however they are not exactly same, and our intention is to extract k instead of preserving exact operations.

THEOREM 2. *Let α be a position-based well-formed aggregation and $MR(\alpha)$ be the generic algorithm for α, then $MR(\alpha)$ is an MRC algorithm if α is reducible and extractable.*

Extractable property of position-based aggregation α allows previous subsequences count value 'k' to be extracted out of mapper operation, then α can be correctly processed by $\sum_{\oplus} F^o$ or $(\sum_{\oplus} f(x_i), \sum_{\oplus} h(i))$ at mapper phase. To combine mapper outputs, more than $\oplus$ and T are needed and specific combining operation depends on the three different extractable conditions (provided in our extended report[2]).

For instance, given an input sequence $\bar{X} = (x_1, ..., x_n)$, then $EMA(\bar{X}) = \frac{\sum_{i=1}^{n}(1-a)^{i-1} \cdot x_i}{\sum_{i=1}^{n}(1-a)^{i-1}}$, where a is a constant between 0 and 1. We give below the asymmetric well-formed aggregation of EMA, where $h(i) = (1-a)^{i-1}$,

$$F^o : F^o(\bar{X}, x_i) = \Big(h(i) \cdot x_i,\ h(i)\Big),$$

$$\oplus : \Big(h(i) \cdot x_i,\ h(i)\Big) \oplus \Big(h(i+1) \cdot x_{i+1},\ h(i+1)\Big)$$
$$= \Big(h(i) \cdot x_i + h(i+1) \cdot x_{i+1},\ h(i) + h(i+1)\Big),$$

$$T : T(\sum_{i=1}^{n} h(i) \cdot x_i, \sum_{i=1}^{n} h(i)) = \frac{\sum_{i=1}^{n} h(i) \cdot x_i}{\sum_{i=1}^{n} h(i)}.$$

It is clearly that EMA is a position-based aggregation, and EMA is reducible because $\oplus$ is a pair of addition. Moreover $h()$ satisfies $h(i+k) = h(i) \cdot h(k+1)$, and the corresponding three binary operations $\otimes = \cdot$, $\odot = \cdot$, $\oplus = +$ satisfy the second extractable condition. Therefore EMA has a MRC algorithm(the generic MRC algorithm for the second extractable condition) illustrated as following, where we assume input sequence $\bar{X} = \bar{S_1} \circ ... \circ \bar{S_m}$ and mapper input is $S_l,\ l \in \{1, ..., m\}$, and $count(S_0) = 0$,

- mapper: $\Big(OM_l^{'} = \sum_{x_j \in S_l} h(j) \cdot x_j,\ OM_l^{''} = \sum_{x_j \in S_l} h(j),$ $OM_l^{'''} = count(S_l)\Big)$,
- reducer: $\frac{\sum_{l=1}^{m} OM_l{'} \cdot (1-a)^{\sum_{j=0}^{l-1} OM_j^{'''}}}{\sum_{l=1}^{m} OM_l^{''} \cdot (1-a)^{\sum_{j=0}^{l-1} OM_j{'''}}}$.

5. CONCLUSION AND FUTURE WORK

In this work, we studied how to map aggregation functions, in a systematic way, into generic MRC algorithms and we identified properties that enable to efficiently execute symmetric and asymmetric aggregations using MapReduce-style platforms. For symmetric aggregation, we proposed the reducible property within well-formed aggregation framework to satisfy space and time complexity of MRC. Several algebraic properties of symmetric aggregation leading to a generic MRC algorithm have been identified. Moreover, we extended the notion of well-formed aggregation to asymmetric aggregation and showed how it can be exploited to deal with position-based asymmetric aggregation. Through identifying the problem for parallelizing it, we proposed extractable property and merged it with the reducible property of asymmetric well-formed aggregation to have MRC algorithms.

Our future work will be devoted to the implementation and experimentation. We will study the extension of our framework to mainstream parallel computing platforms (e.g. Apache Spark). Moreover, we also plan to extend our framework to cover additional classes of asymmetric aggregations. Finally, we plan to investigate how to generalize our approach to nested aggregation functions (i.e., functions defined as a complex composition of aggregation functions).

6. REFERENCES

[1] Moving average. `https://en.wikipedia.org/wiki/Moving_average`.

[2] Symmetric and asymmetric aggregate function in massively parallel computing(extened version). `https://hal-clermont-univ.archives-ouvertes.fr/hal-01533675`.

[3] C.Liu, J.Zhang, H.Zhou, Z. S.McDirmid, and T.Moscibroda. Automating distributed partial aggregation. In *SOCC'14*, pages 1–12, 2014.

[4] S. Cohen. User-defined aggregate functions: bridging theory and practice. In *SIGMOD'06*, pages 49–60, 2006.

[5] S. COHEN, W.NUTT, and Y.SAGIV. Rewriting queries with arbitrary aggregation functions using views. *ACM TODS*, 31(2):672–715, June 2006.

[6] A. Cuzzocrea. Aggregation and multidimensional analysis of big data for large-scale scientific applications: models, issues, analytics, and beyond. In *SSDBM'15*, 2015.

[7] J. Feldman, S.muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On distributing symmetric streaming computations. *ACM TALG*, 6(4), August 2010.

[8] M. Franklin. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[9] M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap. Aggregation function: Means. *Information Sciences*, 181(1):1–22, January 2011.

[10] H.Garcia-Molina, J.D.Ullman, and J.Widom. *Database System Implementation*. Prentice-Hall, New Jersey, 2000.

[11] J.Gray, A.Bosworth, A.Layman, and H.Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, Janaury 1997.

[12] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *SODA'10*, pages 938–948, 2010.

[13] M.Jean-Luc and T.Bruno. Preassociative aggregation functions. *Fuzzy Sets and Systems*, 268:15–26, June 2015.

[14] M.Jean-Luc and T.Bruno. Strongly barycentrically associative and preassociative functions. *Fuzzy Sets and Systems*, 437(1):181–193, May 2016.

[15] S.Madden, M.J.Franklin, J.M.Hellerstein, and W.Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI'02*, pages 131–146, 2002.

[16] V.Raychev, M.Musuvathi, and T.Mytkowicz. Parallelizing user-defined aggregaions using symbolic execution. In *SOSP'15*, pages 153–167, 2015.

[17] Y.Yu, M.Isard, and P. Gunda. Distributed aggregation for data-parallel computing: Interfaces and implementations. In *SOSP'09*, pages 247–260, 2009.

Practical Verification of Hierarchical Artifact Systems

Yuliang Li
Co-Advised by Alin Deutsch and Victor Vianu
UC San Diego
La Jolla, California
yul206@eng.ucsd.edu

ABSTRACT

Data-driven workflows, of which IBM's Business Artifacts are a prime exponent, have been successfully deployed in practice, adopted in industrial standards, and have spawned a rich body of research in academia, focused primarily on static analysis. The present research bridges the gap between the theory and practice of artifact verification by studying the implementation of a full-fledged and efficient artifact verifier for a variant of the Hierarchical Artifact System (HAS) model presented in [9]. With a family of specialized optimizations to the classic Karp-Miller algorithm, our verifier performs >10x faster than a nontrivial Spin-based baseline on real-world workflows and is scalable to large synthetic workflows.

1. INTRODUCTION

The past decade has witnessed the evolution of workflow specification frameworks from the traditional process-centric approach towards data-awareness. Process-centric formalisms focus on control flow while under-specifying the underlying data and its manipulations by the process tasks, often abstracting them away completely. In contrast, data-aware formalisms treat data as first-class citizens. A notable exponent of this class is IBM's *business artifact model* pioneered in [14], successfully deployed in practice [3, 5, 18] and adopted in industrial standards.

In a nutshell, business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks, specified declaratively by pre-and-post conditions. A collection of artifacts and services is called an *artifact system*. IBM has developed several variants of artifacts, of which the most recent is Guard-Stage-Milestone (GSM) [7, 11]. The GSM approach provides rich structuring mechanisms for services, including parallelism, concurrency and hierarchy, and has been incorporated in the OMG standard for Case Management Model and Notation (CMMN) [15, 12].

Artifact systems deployed in industrial settings typically specify complex workflows prone to costly bugs, whence the need for verification of critical properties. Over the past few years, the verification problem for artifact systems was intensively studied. Rather than relying on general-purpose software verification tools suffering from well-known limitations, the focus of the research community has been to identify practically relevant classes of artifact systems and properties for which *fully automatic* verification is possible. This is an ambitious goal, since artifacts are infinite-state systems due to the presence of unbounded data. Along this line, decidability and complexity results were shown for different versions of the verification problem with various expressiveness of the artifact models, as reviewed in the next section.

The project described in this paper bridges the gap between the theory and practice of artifact verification by providing the first implementation of a full-fledged and efficient artifact verifier. The artifact model we use is a variant of the *Hierarchical Artifact System* (HAS) model of [9], which captures core elements of IBM's GSM model. Rather than building on top of an existing program verification tool such as Spin, which we have shown to have strong limitations, we implemented our verifier from scratch. The implementation is based on the classic Karp-Miller algorithm [16], with a family of specialized optimizations to boost performance. The experimental results show that our verifier performs an order of magnitude faster compared to a baseline implementation using Spin [10] on specifications based on real-world BPMN workflows [2], and scales well on large synthetic workflows. To the best of our knowledge, our artifact verifier is the first implementation with full support of unbounded data.

2. BACKGROUND AND RELATED WORK

[8, 6] studied the verification problem for a bare-bones variant of artifact systems, in which each artifact consists of a flat tuple of evolving values and the services are specified by simple pre-and-post conditions on the artifact and database. The verification problem was to check statically whether all runs of an artifact system satisfy desirable properties expressed in LTL-FO, an extension of linear-time temporal logic where propositions are interpreted as existential first-order logic sentences on the database and current artifact tuple. In order to deal with the resulting infinite-state system, a symbolic approach was developed in [8] to allow a reduction to finite-state model checking and yielding a PSPACE verification algorithm for the simplest variant of the model (no database dependencies and uninterpreted data domain). In [6] the approach was extended to allow for database dependencies and numeric data testable by arithmetic constraints.

In our previous work [9], we made significant progress on several fronts. We introduced the HAS model, a much richer and more realistic model abstracting the core elements of

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.

the GSM model. The model features task hierarchy, concurrency, and richer artifact data (including updatable artifact relations). In more detail, a HAS consists of a database and a hierarchy (rooted tree) of *tasks*. Each task has associated to it local evolving data consisting of a tuple of artifact variables and an updatable artifact relation. It also has an associated set of *services*. Each application of a service is guarded by a pre-condition on the database and local data and causes an update of the data, specified by a post-condition (constraining the next artifact tuple) and an insertion or retrieval of a tuple from the artifact relation. In addition, a task may invoke a child task with a tuple of input parameters, and receive back a result if the child task completes. To express properties of HAS we introduce *hierarchical* LTL-FO (HLTL-FO), which is similar to LTL-FO but adapted to the hierarchy. The main results of [9] establish the complexity of checking HLTL-FO properties for various classes of HAS, highlighting the impact of various features on verification.

3. MODEL AND EXAMPLE

The artifact model used in our implementation is a variant of the HAS model of [9] denoted HAS*, which differs from the HAS model used in [9] in two respects. On one hand, it *restricts* HAS by disallowing arithmetic in service pre-and-post conditions, and requires the underlying database schema to use an *acyclic* set of foreign keys, as in the widely used Star (or Snowflake) schemas [17]. On the other hand, HAS* *extends* HAS by allowing an arbitrary number of artifact relations in each task, arbitrary variable propagation, and more flexible interactions between tasks. As shown by our real-life examples, HAS* is powerful enough to model a wide variety of business processes, and so is a good vehicle for studying the implementation of a verifier. Moreover, despite the extensions, the complexity of verifying HLTL-FO properties of HAS* can be shown to remain EXPSPACE, by adapting the techniques of [9].

We illustrate the HAS* model with a simplified example of order fulfillment business process based on a real-world BPMN workflow. The workflow allows customers to place orders and suppliers to process the orders. It has the following database schema:

- CUSTOMERS(id, name, address, record)
  ITEMS(id, item_name, price, in_stock)
  CREDIT_RECORD(id, status)

In the schema, the id's are key attributes, and record is a foreign key referencing CREDIT_RECORD. The CUSTOMERS table contains basic customer information and CREDIT_RECORD provides each customer's credit rating. The ITEMS table contains information on all the items. The artifact system has 4 tasks: T_1:**ProcessOrders**, T_2:**TakeOrder**, T_3:**CheckCredit** and T_4:**ShipItem**, which form the hierarchy in Figure 1.

Intuitively, the root task **ProcessOrders** serves as a global coordinator which manages a pool of orders and the child tasks **TakeOrder**, **CheckCredit** and **ShipItem** implement the 3 processing stages of an order. At each point in a run, **ProcessOrders** nondeterministically picks an order from its pool, triggers one processing stage, and places it back into the pool upon completion.

ProcessOrders: The task has the artifact variables: cust_id, item_id, status which store basic information of an order. It also has an artifact relation ORDERS(cust_id, item_id, status) storing the orders to be processed.

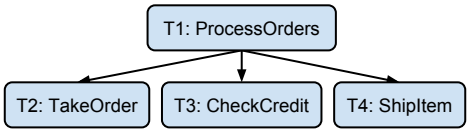


Figure 1: Tasks Hierarchy

The task has 3 internal services: *Initialize*, *StoreOrder* and *RetrieveOrder*. Intuitively, *Initialize* creates a new order with cust_id = item_id = null. When *RetrieveOrder* is called, an order is non-deterministically chosen and removed from ORDERS for processing, and (cust_id, item_id, status) is set to be the chosen tuple. When *StoreOrder* is called, the current order (cust_id, item_id, status) is inserted into ORDERS. The latter two services are specified as follows (the specification consists of a pre-condition, a post-condition, and an update to the ORDERS artifact relation):

StoreOrder:
Pre: cust_id $\neq$ null $\wedge$ item_id $\neq$ null $\wedge$ status $\neq$ "Failed"
Post: cust_id = null $\wedge$ item_id = null $\wedge$ status = "Init"
Update: {+ORDERS(cust_id, item_id, status)}

RetrieveOrder:
Pre: cust_id = null $\wedge$ item_id = null // Post: True
Update: {−ORDERS(cust_id, item_id, status)}

TakeOrder: When this task is called, the customer enters the information of the order (cust_id and item_id) and the status of the order is initialized to "OrderPlaced". The task contains cust_id, record and status as variables and all are return variables to the parent task. There are two services called *EnterCustomer* and *EnterItem*, that allow the customer to enter her and the item's information. The CUSTOMERS and ITEMS tables are queried to obtain the customer ID and item ID. These two services can be called multiple times to allow the customer to modify previously entered data. The task's termination condition is cust_id $\neq$ null $\wedge$ item_id $\neq$ null, at which time its variables are returned to its parent task **ProcessOrders**.

CheckCredit: This task checks the financial record of a customer and decides whether the supplier will go ahead with the sale. It is called when status = "OrderPlaced". It has artifact variables cust_id (input variable), record and status. When the credit record is good, status is set to "Passed", and otherwise to "Failed". After status is set, the task terminates and returns status to the parent task. The task has a single service *Check* performing the credit check.

Check:
Pre: true // Post:

$\exists n \exists a$ CUSTOMERS(cust_id, n, a, record) $\wedge$
(CREDIT_RECORD(record, "Good") $\rightarrow$ status = "Passed") $\wedge$
($\neg$CREDIT_RECORD(record, "Good") $\rightarrow$ status = "Failed")

Note that in a service we can also specify a set of propagated variables whose values stay unchanged when the service is applied. In *Check*, only cust_id is a propagated variable and others will be assigned new values.

ShipItem: This task checks whether the desired item is in stock by looking up the item_id in the ITEMS table to see whether the in_stock attribute equals "Yes". If so, the item is shipped to the customer (status is set to "Shipped") otherwise the order fails (status is set to "Failed"). This task is specified similarly to **CheckCredit** (details omitted).

Properties of HAS* can be specified in LTL-FO. In the

above workflow, we can specify a temporal property saying *"If an order is taken and the ordered item is out of stock, then the item must be restocked before it is shipped."* It can be written in LTL-FO as:

$\forall i\ \mathbf{G}(\texttt{EnterItem} \wedge \texttt{item\_id} = i \wedge \texttt{instock} = \text{"No"}) \rightarrow$
$(\neg(\texttt{ShipItem} \wedge \texttt{item\_id} = i)\ \mathbf{U}\ (\texttt{Restock} \wedge \texttt{item\_id} = i))$

4. VERIFIER IMPLEMENTATION

Although decidability of verification was shown in [9], a naive implementation of the EXPSPACE algorithm outlined there would be wholly impractical. Instead, our implementation brings to bear a battery of optimization techniques crucial to performance. This approach of [9] is based on developing a symbolic representation of the runs of a HAS*. In the representation, each snapshot is summarized by:

(i) the *isomorphism type* of the artifact variables, describing symbolically the structure of the portion of the database reachable from the variables by navigating foreign keys
(ii) for each artifact relation and isomorphism type, the number of tuples in the relation that share that isomorphism type

The heart of the proof in [9] is showing that it is sufficient to verify symbolic runs rather than actual runs. Observe that because of (ii), the symbolic representation is not finite state. Indeed, (ii) requires maintaining a set of counters, which can grow unboundedly. Therefore, the verification algorithm relies on a reduction to state reachability in Vector Addition Systems with States (VASS) [4]. A VASS is a finite-state machine augmented with positive counters that can be incremented and decremented (but not tested for zero). This is essentially equivalent to a Petri Net.

A direct implementation of the above algorithm is impractical because the resulting VASS can have exponentially many states and counters in the input size, and state-of-the-art VASS tools can only handle a small number of counters (<100) [1]. To mitigate the inefficiency, our implementation never generates the whole VASS but instead lazily computes the symbolic representations on-the-fly. Thus, it only generates *reachable* symbolic states, whose number is usually much smaller. In addition, isomorphism types in the symbolic representation are replaced by *partial isomorphism types*, which store only the subset of constraints on the variables imposed by the current run, leaving the rest unspecified. This representation is not only more compact, but also results in an exponentially smaller search space. Then our verifier performs the (repeated) state reachability search using the classic Karp-Miller algorithm [16] with three specialized optimizations to further accelerate the search. We discuss these next.

State Pruning The classic Karp-Miler algorithm is well-known to be inefficient and pruning is a standard way to improve its performance [16]. We introduce a new pruning technique which can be viewed as a generalization of the strategies in [16]. The high-level idea is that when a new state I is found, if there exists a reached state I' such that all states reachable from I are also reachable from I', then we can stop exploring I immediately. In this case, we call I' a *superstate* of I and I a *substate*. Similarly, if there exists a reached state I' which is a substate of I, then we can prune I' and its successors. Compared to [16], our pruning is more aggresive, resulting in a much smaller search space. As shown in Section 5, the performance of the verifier is significantly improved.

Data Structure Support When the above optimization is applied, a frequent operation during the search is to find substates and superstates of a given candidate state in the current set of reached symbolic states. This operation becomes the performance bottleneck when there is a large number of reached states. We accelerate the superstate and substate queries with a Trie index and an Inverted-Lists index, respectively.

Static Analysis The verifier statically analyzes and simplifies the input workflow with a preprocessing step. We notice that in real workflows, some contraints in the specification can never be violated in a symbolic run, and thus can be removed. For example, for a constraint $x = y$ in the specification, where x, y are variables, if $x \neq y$ does not appear anywhere in the specification and is not implied by other constraints, then $x = y$ can be safely removed from the specification without affecting the result of the verification algorithm.

5. EXPERIMENTAL RESULTS

We evaluated the performance of our verifier using both real-world and synthetic artifact specifications.

The Real Set As the artifact approach is still new to the industry, real-world processes available for evaluation are limited. We therefore built an artifact system benchmark specific to business processes, by rewriting the more widely available process-centric BPMN workflows as HAS* specifications. There are numerous sources of BPMN workflows, including the official BPMN website [2], that provides 36 workflows of non-trivial size. To rewrite these workflows into the HAS*, we manually added the database schema, artifact variables/relations, and services for updating the data. Among the 36 real-world BPMN workflows collected from the official BPMN website `bpmn.org`, our model is sufficiently expressive to specify 32 of them in HAS* and can thus be used for performance evaluation. The remaining ones cannot be expressed in HAS* because they involve computing aggregate functions or updating the artifact relations in ways that are not supported in the current model. We will consider these features in our future work.

The Synthetic Set The second benchmark we used for evaluation is a set of randomly generated HAS specifications. All components of each specification, including DB schema, task hierarchy and services, are generated fully at random of a certain size. The ones with empty search space due to unsatisfiable conditions are removed from the benchmark. Table 1 shows some statistics of the benchmarks.

Dataset	Size	#Relations	#Tasks	#Variables	#Services
Real	32	3.563	3.219	20.63	11.59
Synthetic	120	5	5	75	75

Table 1: Statistics of the Two Sets of Workflows

Baseline and Setup We compare our verifier with a simpler implementation built on top of Spin, a widely used software verification tool [10]. Building such a verifier is by itself a challenging task since Spin is incapable of handling data of unbounded size, present in the HAS* model. We managed to build a Spin-based verifier supporting a restricted version of out model, without updatable artifact relations.

As the read-only database can still have unbounded size and domain, the verifier requires a set of nontrivial translations and optimizations. The details will be discussed in a separate paper.

We implemented both verifiers in `C++` with Spin version 6.4.6 for the Spin-based verifier. All experiments were performed on a Linux server with a quad-core Intel i7-2600 CPU and 16G memory. For each workflow in each dataset, we ran our verifiers to test a randomly generated liveness property. The time limit of each run was set to 10 minutes. For fair comparison, since the Spin-based verifier (Spin-Opt) cannot handle artifact relations, we ran both the full Karp-Miller-based verifier (KM), and the Karp-Miller-based verifier with artifact relations ignored (KM-NoSet).

Performance Table 2 shows the results on both sets of workflows. The Spin-based verifier achieves acceptable performance on the real set with an average elapsed time of few seconds and only 1 timeout instance. However, it failed on most runs (109/120) in the synthetic set of workflows. On the other hand, both KM and KM-NoSet achieve average running times below 1 second and with no timeout on the real set, and the average running time is in seconds on the synthetic set, with only 3 timeouts. The presence of artifact relations introduced only a negligible amount of overhead in the running time. Compared with the Spin-based verifier, the KM-based approach is >10x faster in the average running time and more scalable to large workflows.

Mode	Real		Synthetic	
	Avg(Time)	#Timeout	Avg(Time)	#Timeout
Spin-Opt	3.111s	1	67.01s	109
KM-NoSet	**.2635s**	0	**3.214s**	3
KM	**.2926s**	0	**3.355s**	2

Table 2: Performance of the two Verifiers

Cyclomatic Complexity To better understand the scalability of the Karp-Miller-based approach, we measured the difficulty of verifying each workflow using a metric called the cyclomatic complexity [13], which is widely used in software engineering to measure the complexity of program modules. Figure 2 shows that the elapsed time increases exponentially with the cyclomatic complexity. According to [13], it is recommended that any well-designed program should have cyclomatic complexity at most 15 in order to be readable and testable. Our verifier successfully handled all workflows in both benchmarks with cyclomatic complexity less than or equal to 17, which is above the recommended level. For instances with cyclomatic complexity above 15, our verifier only timed out in 2/24 instances (8.33%).

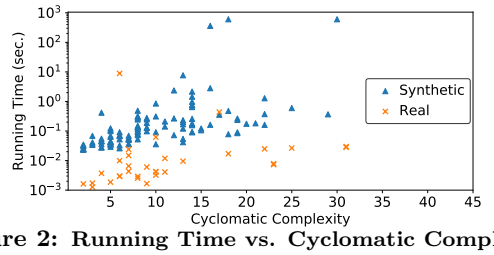


Figure 2: Running Time vs. Cyclomatic Complexity

Comparing Different Optimizations We show next the effect of our 3 optimization techniques: state pruning (SP), static analysis (SA) and data structure support (DSS), by rerunning the experiment with the optimization turned off, and comparing the difference. Table 3 shows the average speedups of each optimization in both datasets. We also present the trimmed averages of the speedups (i.e. removing the top/bottom 5% speedups before averaging) which is less sensitive to extreme values.

Dataset	SP		SA		DSS	
	Mean	Trim.	Mean	Trim.	Mean	Trim.
Real	2943.58x	**55.31x**	1.80x	**1.66x**	1.90x	1.24x
Synthetic	494.57x	**180.82x**	**17.92x**	0.92x	1.45x	**1.27x**

Table 3: Mean and Trimmed Mean (5%) of Speedups

Table 3 shows that the effect of state pruning is the most significant in both sets of workflows, with an average (trimmed) speedup of 55x and 180x in the real and synthetic set, respectively. The static analysis optimization is more effective in the real set (1.6x improvement) but its effect in the synthetic set is less obvious. It creates a small amount (8%) of overhead in most cases, but significantly improves the running time of a single instance, resulting in the huge gap between the normal average speedup and the trimmed average speedup. Finally, the data-structure support provides a consistent ∼1.2x average speedup in both datasets.

Acknowledgement This work was supported in part by the National Science Foundation under award IIS-1422375.

6. REFERENCES

[1] MIST - a safety checker for petri nets and extensions. `https://github.com/pierreganty/mist/wiki`.

[2] Object management group business process model and notation. `http://www.bpmn.org/`. Accessed: 2017-03-01.

[3] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[4] M. Blockelet and S. Schmitz. Model checking coverability graphs of vector addition systems. In *Mathematical Foundations of Computer Science 2011*, pages 108–119. Springer, 2011.

[5] T. Chao et al. Artifact-based transformation of IBM Global Financing: A case study. In *BPM*, 2009.

[6] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. TODS, 37(3):22, 2012.

[7] E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*, 38:561–584, 2013.

[8] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[9] A. Deutsch, Y. Li, and V. Vianu. Verification of hierarchical artifact systems. In *PODS*, pages 179–194, 2016.

[10] G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.

[11] R. Hull et al. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *ACM DEBS*, 2011.

[12] M. Marin, R. Hull, and R. Vaculín. Data centric bpm and the emerging case management standard: A short survey. In *BPM Workshops*, 2012.

[13] T. J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[14] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[15] Object Management Group. Case Management Model and Notation (CMMN), 2014.

[16] P.-A. Reynier and F. Servais. Minimal coverability set for petri nets: Karp and miller algorithm with pruning. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 69–88. Springer, 2011.

[17] P. Vassiliadis and T. Sellis. A survey of logical models for olap databases. *ACM Sigmod Record*, 28(4):64–69, 1999.

[18] W.-D. Zhu et al. *Advanced Case Management with IBM Case Manager*. IBM Redbooks, 2015.