# Proceedings

## 25. GI-Workshop „Grundlagen von Datenbanken“

28.05.2013 – 31.05.2013

Ilmenau, Deutschland

Kai-Uwe Sattler
Stephan Baumann
Felix Beier
Heiko Betz
Francis Gropengießer
Stefan Hagedorn
(Hrsg.)

# Vorwort

Liebe Teilnehmerinnen und Teilnehmer,

mittlerweile zum 25. Mal fand vom 28.5. bis 31.5.2013 der Workshop „Grundlagen von Datenbanken“ des GI-Arbeitskreises „Grundlagen von Informationssystemen“ im Fachbereich Datenbanken und Informationssysteme (DBIS) statt. Nach Österreich im Jahr 2011 und dem Spreewald im Jahr 2012 war bereits zum dritten Mal Thüringen der Austragungsort – diesmal die kleine Gemeinde Elgersburg am Fuße der Hohen Warte im Ilm-Kreis. Organisiert wurde der Workshop vom Fachgebiet Datenbanken und Informationssysteme der TU Ilmenau.

Die Workshop-Reihe, die 1989 in Volkse bei Braunschweig vom Braunschweiger Datenbanklehrstuhl ins Leben gerufen wurde und die ersten 3 Jahre auch in Volkse blieb, hat sich inzwischen als eine Institution für den Gedankenaustausch gerade für Nachwuchswissenschaftler/-innen aus dem deutschsprachigen Raum im Bereich Datenbanken und Informationssysteme etabliert. Längst sind dabei die Beschränkungen auf Deutsch als Vortragssprache und reine theorie- und grundlagenorientierte Themen gefallen – auch wenn die offene Atmosphäre an abgeschiedenen Tagungsorten (und Elgersburg stellte hier keine Ausnahme dar) mit viel Zeit für intensive Diskussionen während der Sitzungen und an den Abenden geblieben sind.

Für den diesjährigen Workshop wurden 15 Beiträge eingereicht und von jeweils drei Mitgliedern des 13-köpfigen Programmkomitees begutachtet. Aus allen eingereichten Beiträgen wurden 13 für die Präsentation auf dem Workshop ausgewählt. Die Bandbreite der Themen reichte dabei von fast schon klassischen Datenbankthemen wie Anfrageverarbeitung (mit XQuery), konzeptueller Modellierung (für XML Schemaevolution), Indexstrukturen (für Muster auf bewegten Objekten) und dem Auffinden von Spaltenkorrelationen über aktuelle Themen wie MapReduce und Cloud-Datenbanken bis hin zu Anwendungen im Bereich Image Retrieval, Informationsextraktion, Complex Event Processing sowie Sicherheitsaspekten.

Vervollständigt wurde das viertägige Programm durch zwei Keynotes von namhaften Datenbankforschern: Theo Härder stellte das WattDB-Projekt eines energieproportionalen Datenbanksystems vor und Peter Boncz diskutierte die Herausforderungen an die Optimierung von Datenbanksysteme durch moderne Hardwarearchitekturen - untersetzt mit praktischen Vorführungen. Beiden sei an dieser Stelle für ihr Kommen und ihre interessanten Vorträge gedankt. In zwei weiteren Vorträgen nutzten die Sponsoren des diesjährigen Workshops, SAP AG und Objectivity Inc., die Gelegenheit, die Datenbanktechnologien hinter HANA (SAP AG) und InfiniteGraph (Objectivity Inc.) vorzustellen. Hannes Rauhe und Timo Wagner als Vortragenden möchten wir daher genauso wie den beiden Unternehmen für die finanzielle Unterstützung des Workshops und damit der Arbeit des GI-Arbeitskreises danken.

Gedankt sei an dieser Stelle auch allen, die an der Organisation und Durchführung beteiligt waren: den Autoren für ihre Beiträge und Vorträge, den Mitgliedern des Programmkomitees für ihre konstruktive und pünktliche Begutachtung der Einreichungen, den Mitarbeitern vom Hotel am Wald in Elgersburg, dem Leitungsgremium des Arbeitskreises in Person von Günther Specht und Stefan Conrad, die es sich nicht nehmen ließen, persönlich am Workshop teilzunehmen, sowie Eike Schallehn, der im Hintergrund mit Rat und Tat zur Seite stand. Der größte Dank gilt aber meinem Fachgebietsteam, das den Großteil der Organisationsarbeit geleistet hat: Stephan Baumann, Francis Gropengießer, Heiko Betz, Stefan Hagedorn und Felix Beier. Ohne ihr Engagement wäre der Workshop nicht möglich gewesen. Herzlichen Dank!

Kai-Uwe Sattler

Ilmenau am 28.5.2013

# Komitee

## Programm-Komitee

- Andreas Heuer, Universität Rostock
- Eike Schallehn, Universität Magdeburg
- Erik Buchmann, Karlsruher Institut für Technologie
- Friederike Klan, Universität Jena
- Gunter Saake, Universität Magdeburg
- Günther Specht, Universität Innsbruck
- Holger Schwarz, Universität Stuttgart
- Ingo Schmitt, Brandenburgische Technische Universität Cottbus
- Kai-Uwe Sattler, Technische Universität Ilmenau
- Katja Hose, Aalborg University
- Klaus Meyer-Wegener, Universität Erlangen
- Stefan Conrad, Universität Düsseldorf
- Torsten Grust, Universität Tübingen

## Organisations-Komitee

- Kai-Uwe Sattler, TU Ilmenau
- Stephan Baumann, TU Ilmenau
- Felix Beier, TU Ilmenau
- Heiko Betz, TU Ilmenau
- Francis Gropengießer, TU Ilmenau
- Stefan Hagedorn, TU Ilmenau

# Inhaltsverzeichnis

# WattDB—a Rocky Road to Energy Proportionality

Theo Härder

Databases and Information Systems Group
University of Kaiserslautern, Germany
haerder@cs.uni-kl.de

## Extended Abstract

Energy efficiency is becoming more important in database design, i. e., the work delivered by a database server should be accomplished by minimal energy consumption. So far, a substantial number of research papers examined and optimized the energy consumption of database servers or single components. In this way, our first efforts were exclusively focused on the use of flash memory or SSDs in a DBMS context to identify their performance potential for typical DB operations. In particular, we developed tailor-made algorithms to support caching for flash-based databases [3], however with limited success concerning the energy efficiency of the entire database server.

A key observation made by Tsirogiannis et al. [5] concerning the energy efficiency of single servers, the best performing configuration is also the most energy-efficient one, because power use is not proportional to system utilization and, for this reason, runtime needed for accomplishing a computing task essentially determines energy consumption. Based on our caching experiments for flash-based databases, we came to the same conclusion [2]. Hence, the server system must be fully utilized to be most energy efficient. However, real-world workloads do not stress servers continuously. Typically, their average utilization ranges between 20 and 50% of peak performance [1]. Therefore, traditional single-server DBMSs are chronically underutilized and operate below their optimal energy-consumption-per-query ratio. As a result, there is a big optimization opportunity to decrease energy consumption during off-peak times.

Because the energy use of single-server systems is far from being *energy proportional*, we came up with the hypothesis that better energy efficiency may be achieved by a cluster of nodes whose size is dynamically adjusted to the current workload demand. For this reason, we shifted our research focus from inflexible single-server DBMSs to distributed clusters running on lightweight nodes. Although distributed systems impose some performance degradation compared to a single, brawny server, they offer higher energy saving potential in turn.

Current hardware is not energy proportional, because a single server consumes, even when idle, a substantial fraction of its peak power [1]. Because typical usage patterns lead to a server utilization far less than its maximum, energy efficiency of a server aside from peak performance is reduced [4]. In order to achieve energy proportionality using commodity hardware, we have chosen a clustered approach, where each node can be powered independently. By turning on/off whole nodes, the overall performance and energy consumption can be fitted to the current workload. Unused servers could be either shut down or made available to other processes. If present in a cloud, those servers could be leased to other applications.

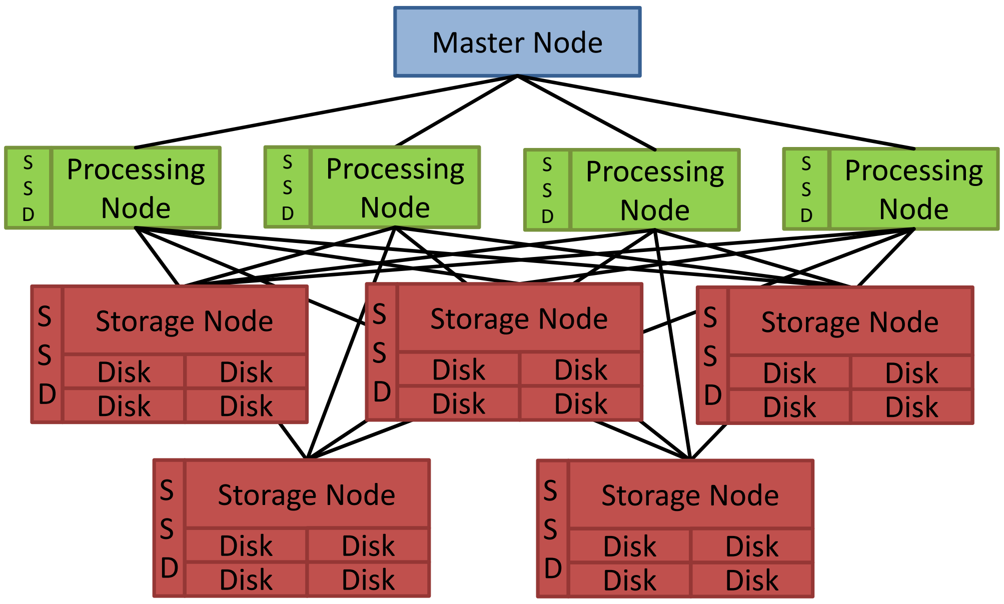
We have developed a research prototype of a distributed DBMS called *WattDB* on a scale-out architecture, consisting of $n$ wimpy computing nodes, interconnected by an 1GBit/s Ethernet switch. The cluster currently consists of 10 identical nodes, composed of an Intel Atom D510 CPU, 2 GB DRAM and an SSD. The configuration is considered Amdahl-balanced, i. e., balanced between I/O and network throughput on one hand and processing power on the other.

Compared to InfiniBand, the bandwidth of the interconnecting network is limited but sufficient to supply the lightweight nodes with data. More expensive, yet faster connections would have required more powerful processors and more sophisticated I/O subsystems. Such a design would have pushed the cost beyond limits, especially because we would not have been able to use commodity hardware. Furthermore, by choosing lightweight components, the overall energy footprint is low and the smallest configuration, i. e., the one with the fewest number of nodes, exhibits low power consumption. Moreover, experiments running on a small cluster can easily be repeated on a cluster with more powerful nodes.

A dedicated node is the *master node*, handling incoming queries and coordinating the cluster. Some of the nodes have each four hard disks attached and act as *storage nodes*, providing persistent data storage to the cluster. The remaining nodes (without hard disks drives) are called *processing nodes*. Due to the lack of directly accessible storage, they can only operate on data provided by other nodes (see Figure 1).

All nodes can evaluate (partial) query plans and execute DB operators, e. g., sorting, aggregation, etc., but only the *storage nodes* can access the DB storage structures, i. e., tables and indexes. Each storage node maintains a DB buffer

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2012, Ilmenau, Germany.



Master Node
SSD Processing Node
SSD Processing Node
SSD Processing Node
SSD Processing Node
SSD Storage Node Disk Disk Disk Disk
SSD Storage Node Disk Disk Disk Disk
SSD Storage Node Disk Disk Disk Disk
SSD Storage Node Disk Disk Disk Disk
SSD Storage Node Disk Disk Disk Disk

**Figure 1: Overview of the WattDB cluster**

to keep recently referenced pages in main memory, whereas a processing node does not cache intermediate results. As a consequence, each query needs to always fetch the qualified records from the corresponding storage nodes.

Hence, our cluster design results in a *shared-nothing architecture* where the nodes only differentiate to those which have or have not direct access to DB data on external storage. Each of the nodes is additionally equipped with a 128GB Solid-State Disk (Samsung 830 SSD). The SSDs do not store the DB data, they provide swap space to support external sorting and to provide persistent storage for configuration files. We have chosen SSDs, because their access latency is much lower compared to traditional hard disks; hence, they are better suited for temp storage.

In WattDB, a dedicated component, running on the master node, controls the energy consumption, called *Energy-Controller*. This component monitors the performance of all nodes in the cluster. Depending on the current query workload and node utilization, the *EnergyController* activates and suspends nodes to guarantee a sufficiently high node utilization depending on the workload demand. Suspended nodes do only consume a fraction of the idle power, but can be brought back online in a matter of a few seconds. It also modifies query plans to dynamically distribute the current workload on all running nodes thereby achieving balanced utilization of the active processing nodes.

As data-intensive workloads, we submit specific TPC-H queries against a distributed shared-nothing DBMS, where time and energy use are captured by specific monitoring and measurement devices. We configure various static clusters of varying sizes and show their influence on energy efficiency and performance. Further, using an *EnergyController* and a load-aware scheduler, we verify the hypothesis that energy proportionality for database management tasks can be well approximated by dynamic clusters of wimpy computing nodes.

## 1. REFERENCES

[1] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, 2007.

[2] T. Härder, V. Hudlet, Y. Ou, and D. Schall. Energy efficiency is not enough, energy proportionality is needed! In *DASFAA Workshops, 1st Int. Workshop on FlashDB, LNCS 6637*, pages 226–239, 2011.

[3] Y. Ou, T. Härder, and D. Schall. Performance and Power Evaluation of Flash-Aware Buffer Algorithms. In *DEXA, LNCS 6261*, pages 183–197, 2010.

[4] D. Schall, V. Höfner, and M. Kern. Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In *TPCTC, LNCS 7144*, pages 31–45, 2012.

[5] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the Energy Efficiency of a Database Server. In *SIGMOD Conference*, pages 231–242, 2010.

# Optimizing database architecture for machine architecture: is there still hope?

Peter Boncz

CWI
p.boncz@cwi.nl

## Extended Abstract

In the keynote, I will give some examples of how computer architecture has strongly evolved in the past decennia and how this influences the performance, and therefore the design, of algorithms and data structure for data management. One the one hand, these changes in hardware architecture have caused the (continuing) need for new data management research. i.e. hardware-conscious database research. Here, I will draw examples from hardware-conscious research performed on the CWI systems MonetDB and Vectorwise.

This diversification trend in computer architectural characteristics of the various solutions in the market seems to be intensifying. This is seen in quite different architectural options, such as CPU vs GPU vs FPGA, but also even restricting oneself to just CPUs there seems to be increasing design variation in architecture and platform behavior. This poses a challenge to hardware-conscious database research.

In particular, there is the all too present danger to over-optimize of one particular architecture; or to propose techniques that will have only a very short span of utility. The question thus is not only to find specific ways to optimize for certain hardware features, but do so in a way that works across the full spectrum of architectural, i.e. robust techniques.

I will close the talk by recent work at CWI and Vectorwise on robustness of query evaluator performance, describing a project called "Micro-Adaptivity" where database systems are made self-adaptive and react immediately to observed performance, self-optimizing to the combination of current query workload, observed data distributions, and hardware characteristics.

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2012, Ilmenau, Germany.

# Adaptive Prejoin Approach for Performance Optimization in MapReduce-based Warehouses

Weiping Qu
Heterogeneous Information Systems Group
University of Kaiserslautern
qu@informatik.uni-kl.de

Michael Rappold[*]
Department of Computer Science
University of Kaiserslautern
m_rappol@cs.uni-kl.de

Stefan Dessloch
Heterogeneous Information Systems Group
University of Kaiserslautern
dessloch@informatik.uni-kl.de

## ABSTRACT

MapReduce-based warehousing solutions (e.g. Hive) for big data analytics with the capabilities of storing and analyzing high volume of both structured and unstructured data in a scalable file system have emerged recently. Their efficient data loading features enable a so-called near real-time warehousing solution in contrast to those offered by conventional data warehouses with complex, long-running ETL processes.

However, there are still many opportunities for performance improvements in MapReduce systems. The performance of analyzing structured data in them cannot cope with the one in traditional data warehouses. For example, join operations are generally regarded as a bottleneck of performing generic complex analytics over structured data with MapReduce jobs.

In this paper, we present one approach for improving performance in MapReduce-based warehouses by pre-joining frequently used dimension columns with fact table redundantly during data transfer and adapting queries to this join-friendly schema automatically at runtime using a rewrite component. This approach is driven by the statistics information derived from previous executed workloads in terms of join operations.

The results show that the execution performance is improved by getting rid of join operations in a set of future workloads whose join exactly fits the pre-joined fact table schema while the performance still remains the same for other workloads.

## 1. INTRODUCTION

By packaging complex custom imperative programs (text mining, machine learning, etc.) into simple `map` and `reduce` functions and executing them in parallel on files in a large scalable file system, MapReduce/Hadoop[1] systems enable analytics on large amounts of unstructured data or structured data in acceptable response time.

With the continuous growth of data, scalable data stores based on Hadoop/HDFS[2] have achieved more and more attention for big data analytics. In addition, by means of simply pulling data into the file system of MapReduce-based systems, unstructured data without schema information is directly analyzed with parallelizable custom programs, whereas data can only be queried in traditional data warehouses after it has been loaded by ETL tools (cleansing, normalization, etc.), which normally takes a long period of time.

Consequently, many web or business companies add MapReduce systems to their analytical architecture. For example, Fatma Özcan et al. [12] integrate their DB2 warehouse with the Hadoop-based analysis tool - IBM Infosphere BigInsights with connectors between these two platforms. An analytical synthesis is provided, where unstructured data is initially placed in a Hadoop-based system and analyzed by MapReduce programs. Once its schema can be defined, it is further loaded into a DB2 warehouse with more efficient analysis execution capabilities.

Another example is the data warehousing infrastructure at Facebook which involves a web-based tier, a federated MySQL tier and a Hadoop-based analytical cluster - Hive.

Such orchestration of various analytical platforms forms a heterogeneous environment where each platform has a different interface, data model, computational capability, storage system, etc.

Pursuing a global optimization in such a heterogeneous environment is always challenging, since it is generally hard to estimate the computational capability or operational cost concisely on each autonomous platform. The internal query engine and storage system do not tend to be exposed to outside and are not designed for data integration.

In our case, relational databases and Hadoop will be integrated together to deliver an analytical cluster. Simply transferring data from relational databases to Hadoop without considering the computational capabilities in Hadoop can lead to lower performance.

As an example, performing complex analytical workloads over multiple small/large tables (loaded from relational data-

[*]finished his work during his master study at university of kaiserslautern

[1]one open-source implementation of MapReduce framework from Apache community, see http://hadoop.apache.org
[2]Hadoop Distributed File System - is used to store the data in Hadoop for analysis

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

bases) in Hadoop leads to a number of join operations which slows down the whole processing. The reason is that the join performance is normally weak in MapReduce systems as compared to relational databases [15]. Performance limitations have been shown due to several reasons such as the inherent unary feature of `map` and `reduce` functions.

To achieve better global performance in such an analytical synthesis with multiple platforms from a global perspective of view, several strategies can be applied.

One would be simply improving the join implementation on single MapReduce platform. There have been several existing works trying to improve join performance in MapReduce systems [3, 1].

Another one would be using heuristics for global performance optimization. In this paper, we will take a look at the second one. In order to validate our general idea of improving global performance on multiple platforms, we deliver our adaptive approach in terms of join performance. We take the data flow architecture at Facebook as a starting point and the contributions are summarized as follows:

1. Adaptively pre-joining tables during data transfer for better performance in Hadoop/Hive.

2. Rewriting incoming queries according to changing table schema.

The remainder of this paper is structured as follows: Section 2 describes the background of this paper. Section 3 gives a naïve approach of fully pre-joining related tables. Based on the performance observation of this naïve approach, more considerations have been taken into account and an adaptive pre-join approach is proposed in Section 4, followed by the implementation and experimental evaluation shown in Section 5. Section 6 shows some related works. Section 7 concludes with a summary and future work.

## 2. BACKGROUND

In this section, we will introduce our starting point, i.e. the analytical data flow architecture at Facebook and its MapReduce-based analytical platform - Hive. In addition, the performance issue in terms of join is also stated subsequently.

### 2.1 Facebook Data Flow Architecture

Instead of using a traditional data warehouse, Facebook uses Hive - a MapReduce-based analytical platform - to perform analytics on information describing advertisement. The MapReduce/Hadoop system offers high scalability which enables Facebook to perform data analytics over 15PB of data and load 60TB of new data every day [17]. The architecture of data flow at Facebook is described as follows.

As depicted in Figure 1, data is extracted from two types of data sources: a federated MySQL tier and a web-based tier. The former offers the category, the name and corresponding information of the advertisements as dimension data while the actions such as viewing an advertisement, clicking on it, fanning a Facebook page are extracted as fact data from the latter.

There are two types of analytical cluster: production Hive cluster and ad hoc Hive cluster. Periodic queries are performed on the production Hive cluster while the ad hoc queries are executed on the ad hoc Hive cluster.

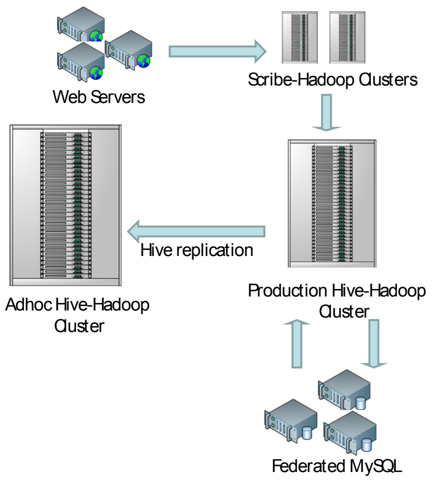


**Figure 1: Facebook Data Flow Architecture[17]**

### 2.2 Hive

Hive [16] is an open source data warehousing solution built on top of MapReduce/Hadoop. Analytics is essentially done by MapReduce jobs and data is still stored and managed in Hadoop/HDFS.

Hive supports a higher-level SQL-like language called HiveQL for users who are familiar with SQL for accessing files in Hadoop/HDFS, which highly increases the productivity of using MapReduce systems. When a HiveQL query comes in, it will be automatically translated into corresponding MapReduce jobs with the same analytical semantics. For this purpose, Hive has its own meta-data store which maps the HDFS files to the relational data model. Files are logically interpreted as relational tables during HiveQL query execution.

Furthermore, in contrast to high data loading cost (using ETL jobs) in traditional data warehouses, Hive benefits from its efficient loading process which pulls raw files directly into Hadoop/HDFS and further publishes them as tables. This feature makes Hive much more suitable for dealing with large volumes of data (i.e. big data).

### 2.3 Join in Hadoop/Hive

There has been an ongoing debate comparing parallel database systems and MapReduce/Hadoop. In [13], experiments showed that performance of selection, aggregation and join tasks in Hadoop could not reach parallel databases (Vertica & DBMS-X). Several reasons of the performance difference have been also explained by Stonebraker et al. in [15] such as repetitive record parsing, and high I/O cost due to non-compression & non-indexing.

Moreover, as MapReduce was not originally designed to combine information from two or more data sources, join implementations are always cumbersome [3]. The join performance relies heavily on the implementation of MapReduce jobs which have been considered as not straightforward.

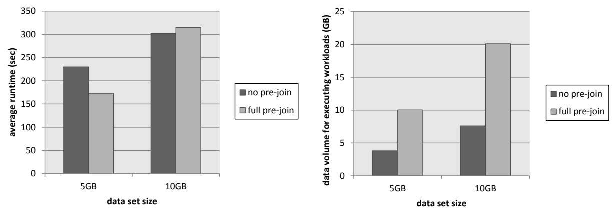
As Hive is built on top of MapReduce/Hadoop, the join operation is essentially done by corresponding MapReduce jobs. Thus, Hive suffers from these issues even though there have been efforts [5] to improve join performance in MapReduce systems or in Hive.

## 3. FULL PRE-JOIN APPROACH

Due to the fact that the join performance is a performance bottleneck in Hive with its inherent MapReduce feature, one naïve thinking for improving total workload performance would be to simply eliminate the join task from the workload by performing a rewritten workload with the same analytical semantics over pre-joined tables created in the data load phase. A performance gain would be expected by performing large table scan with high parallelism of increasing working nodes in Hadoop instead of join. In addition, the scalable storage system allows us to create redundant pre-joined tables for some workloads with specific join patterns.

In an experiment, we tried to validate this strategy. An analytical workload (TPC-H Query 3) was executed over two data sets of TPC-H benchmark (with scale factor 5 & 10) of the original table schema (with join at runtime) and a fully pre-joined table schema (without join) which fully joins all the related dimension tables with the fact table during the load phase, respectively. In this case, we trade storage overhead for better total performance.

As shown on the left side of the Figure 2(a), the performance gain of the total workload (including the join) over the data set with SF 5 can be seen with 6GB storage overhead introduced by fully pre-joining the related tables into one redundant table (shown in Figure 2(b)). The overall



(a) Average Runtimes (b) Accessed Data Volume

**Figure 2: Running TPC-H Query-3 on Original and Full Pre-joined Table Schema**

performance can be significantly increased if workloads with the same join pattern later frequently occur, especially for periodic queries over production Hive-Hadoop cluster in the Facebook example.

However, the result of performing the same query on the data set with SF 10 size is disappointing as there is no performance gain while paying 12.5GB storage for redundancy (shown in Figure 2(b)), which is not what we expected. The reason could be that the overhead of scanning such redundant fully pre-joined tables and the high I/O cost as well offset the performance gain as the accessed data volume grows.

## 4. ADAPTIVE PRE-JOIN APPROACH

Taking the lessons learned from the full pre-join approach above, we propose an adaptive pre-join approach in this paper.

Instead of pre-joining full dimension tables with the fact table, we try to identify the dimension columns which occurred frequently in the `select`, `where`, etc. clauses of previous executed queries for filtering, aggregation and so on. We refer to these columns as additional columns as compared to the join columns in the join predicates. By collecting a list of additional column sets from previous queries, for example,

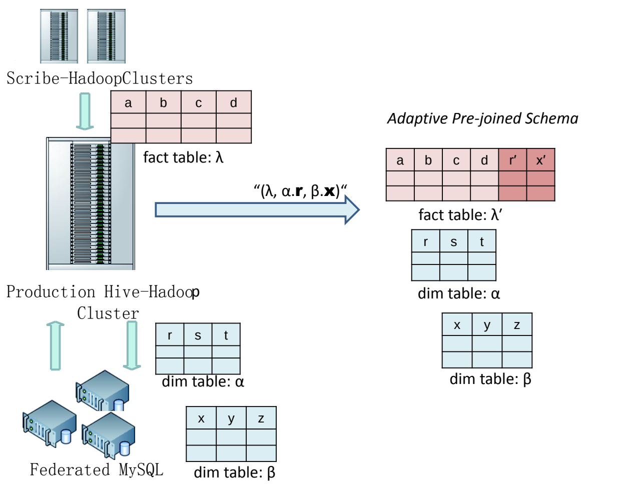


**Figure 3: Adaptive Pre-joined Schema in Facebook Example**

the periodic queries on production Hive-Hadoop cluster, a frequent column set could be extracted.

One example is illustrated in Figure 3. The frequent set of additional columns has been extracted. The column `r` in dimension table $\alpha$ is frequently joined with fact table in company in the previous workloads as a filter or aggregate column, as the same for the column `x` in dimension table $\beta$. During next load phase, the fact table is expanded by redundantly pre-joining these two additional columns `r` and `x` with it.

Depending on the statistics information of previous queries, different frequent sets of additional columns could be found in diverse time intervals. Thus, the fact table is pre-joined in an adaptive manner.

Assume that the additional columns identified in previous queries will also frequently occur in the future ones (as in the Facebook example), the benefits of adaptive pre-join approach are two-fold:

First, when all the columns (including dimension columns) in a certain incoming query which requires a join operation have been contained in the pre-joined fact table, this query could be directly performed on the pre-joined fact table without join.

Second, the adaptive pre-join approach leads to a smaller table size in contrast to the full pre-join approach, as only subsets of the dimension tables are pre-joined. Thus, the resulting storage overhead is reduced, which plays a significant role especially in big data scenarios (i.e. terabytes, petabytes of data).

To automatically accomplish the adaptive pre-join approach, three sub-steps are developed: frequent column set extraction, pre-join and query rewrite.

### 4.1 Frequent Column Set Extraction

In the first phase, the statistics collected for extracting frequent set of additional columns is formated as a list of entries each which has the following form:

$$Set: \ \{Fact, \ Dim\_X.Col\_i, \ Dim\_X.Col\_j \ ... \ \ Dim\_Y.Col\_k\}$$

The join set always starts with the involved fact table while the joint dimension columns are identified and captured from the `select`, `where`, etc. clauses or from the sub-queries.

The frequent set of additional columns could be extracted using a set of frequent itemset mining approaches [2, 7, 11]

### 4.2 Query Rewrite

As the table schema is changed in our case (i.e. newly generated fact table schema), initial queries need to be rewritten for successful execution. Since the fact table is pre-joined with a set of dedicated redundant dimension columns, the tables which are involved in the `from` clause of the original query can be replaced with this new fact table once all the columns have been covered in it.

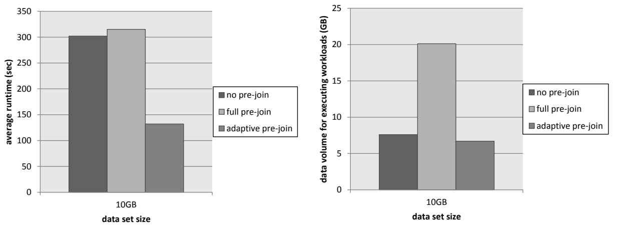
By storing the mapping from newly generated fact table schema to the old schema in the catalog, the query rewrite process can be easily applied. Note that the common issue of handling complex sub-queries for Hive can thereby be facilitated if the columns in the sub-query have been pre-joined with the fact table.

## 5. IMPLEMENTATION AND EVALUATION

We use Sqoop[3] as the basis to implement our approach. The TPC-H benchmark data set with SF 10 is adaptively pre-joined according to the workload statistics and transferred from MySQL to Hive. First, the extracted join pattern information is sent to Sqoop as additional transformation logic embedded in the data transfer jobs for generating the adaptive pre-joined table schema on the original data sources. Furthermore, the generated schema is stored in Hive to enable automatic query rewrite at runtime.

We tested the adaptive pre-join approach on a six-node cluster (Xeon Quadcore CPU at 2.53GHz, 4GB RAM, 1TB SATA-II disk, Gigabit Ethernet) running Hadoop and Hive.

After running the same TPC-H Query 3 over the adaptive pre-joined table schema, the result in the Figure 4(a) shows that the average runtime is significantly reduced. The join



(a) Average Runtimes (b) Accessed Data Volume

**Figure 4: Running TPC-H Query-3 on Original, Full Pre-joined and Adaptive Pre-joined Table Schema**

task has been eliminated for this query and the additional overheads (record parsing, I/O cost) have been relieved due to the smaller size of redundancy as shown in Figure 4(b).

## 6. RELATED WORK

An adaptively pre-joined fact table is essentially a materialized view in Hive. Creating materialized views in data warehouses is nothing new but a technique used for query optimization. Since 1990s, a substantial effort [6, 8] has been to answer queries using views in data warehouses. Furthermore, several subsequent works [14, 10] have focuses on dynamic view management based on runtime statistics (e.g. reference frequency, result data size, execution cost) and measured profits for better query performance. In our work, we reviewed these sophisticated techniques in a MapReduce-based environment.

Cheetah [4] is a high performance, custom data warehouse on top of MapReduce. It is very similar to the MapReduce-based warehouse Hive introduced in this paper. The performance issue of join implementation has also been addressed in Cheetah. To reduce the network overhead for joining big dimension table with fact table at query runtime, big dimension tables are denormalized and all the dimension attributes are directly stored into the fact table. In contrast, we choose to only denormalize the frequently used dimension attributes with the fact table since we believe that less I/O cost can be achieved in this way.

## 7. CONCLUSION AND FUTURE WORK

We propose a schema adaption approach for global optimization in an analytical synthesis of relational databases and a MapReduce-based warehouse - Hive. As MapReduce systems have weak join performance, frequently used columns of dimension tables are pre-joined with the fact table according to useful workload statistics in an adaptive manner before being transfered to Hive. Besides, a rewrite component enables the execution of incoming workloads with join operations over such pre-joined tables transparently. In this way, better performance can be achieved in Hive. Note that this approach is not restricted to any specific platform like Hive. Any MapReduce-based warehouse can benefit from it, as generic complex join operations occur in almost every analytical platform.

However, the experimental results also show that the performance improvement is not stable while the data volume grows continuously. For example, when the query is executed on one larger pre-joined table, the performance gain from eliminating joins is offset by the impact caused by the record parsing overhead and high I/O cost during the scan, which results in worse performance. This concludes that the total performance of complex data analytics is effected by multiple metrics rather than a unique consideration, e.g. join.

With the continuous growth of data, diverse frameworks and platforms (e.g. Hive, Pig) are built for large-scale data analytics and business intelligent applications. Data transfer between different platforms generally takes place in the absence of key information such as operational cost model, resource consumption, computational capability etc. within platforms which are autonomous and inherently not designed for data integration. Therefore, we are looking at a generic description of the operational semantics with their computational capabilities on different platforms and a cost model for performance optimization from a global perspective of view. The granularity we are observing is a single operator in the execution engines. Thus, a global operator model with generic cost model is expected for performance improvement in several use cases, e.g. federated systems.

Moreover, as an adaptively pre-joined fact table is regarded as a materialized view in a MapReduce-based warehouse, another open problem left is how to handle the view maintanence issue. The work from [9] introduced an incre-

[3]an open source tool for data transfer between Hadoop and relational database, see http://sqoop.apache.org/

mental loading approach to achieve near real-time datawarehousing by using change data capture and change propagation techniques. Ideas from this work could be taken further to improve the performance of total workload including the pre-join task.

## 8. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 99–110, New York, NY, USA, 2010. ACM.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 975–986, New York, NY, USA, 2010. ACM.

[4] S. Chen. Cheetah: a high performance, custom data warehouse on top of mapreduce. *Proc. VLDB Endow.*, 3(1-2):1459–1468, Sept. 2010.

[5] A. Gruenheid, E. Omiecinski, and L. Mark. Query optimization using column statistics in hive. In *Proceedings of the 15th Symposium on International Database Engineering & Applications*, IDEAS '11, pages 97–105, New York, NY, USA, 2011. ACM.

[6] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, Dec. 2001.

[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[8] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 205–216, New York, NY, USA, 1996. ACM.

[9] T. Jörg and S. Deßloch. Towards generating etl processes for incremental loading. In *Proceedings of the 2008 international symposium on Database engineering & applications*, IDEAS '08, pages 101–110, New York, NY, USA, 2008. ACM.

[10] Y. Kotidis and N. Roussopoulos. Dynamat: a dynamic view management system for data warehouses. *SIGMOD Rec.*, 28(2):371–382, June 1999.

[11] H. Mannila, H. Toivonen, and I. Verkamo. Efficient algorithms for discovering association rules. pages 181–192. AAAI Press, 1994.

[12] F. Özcan, D. Hoa, K. S. Beyer, A. Balmin, C. J. Liu, and Y. Li. Emerging trends in the enterprise data analytics: connecting hadoop and db2 warehouse. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 1161–1164, New York, NY, USA, 2011. ACM.

[13] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[14] P. Scheuermann, J. Shim, and R. Vingralek. Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 51–62, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[15] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? *Commun. ACM*, 53(1):64–71, Jan. 2010.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE '10: Proceedings of the 26th International Conference on Data Engineering*, pages 996–1005. IEEE, Mar. 2010.

[17] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1013–1020, New York, NY, USA, 2010. ACM.

# Ein Cloud-basiertes räumliches Decision Support System für die Herausforderungen der Energiewende

Golo Klossek
Hochschule Regensburg
golo.klossek
@stud.hs-regensburg.de

Stefanie Scherzinger
Hochschule Regensburg
stefanie.scherzinger
@hs-regensburg.de

Michael Sterner
Hochschule Regensburg
michael.sterner
@hs-regensburg.de

## KURZFASSUNG

Die Energiewende in Deutschland wirft sehr konkrete Fragestellungen auf: Welche Standorte eignen sich für Windkraftwerke, wo können Solaranlagen wirtschaftlich betrieben werden? Dahinter verbergen sich rechenintensive Datenverarbeitungsschritte, auszuführen auf *Big Data* aus mehreren Datenquellen, in entsprechend heterogenen Formaten. Diese Arbeit stellt exemplarisch eine konkrete Fragestellung und ihre Beantwortung als MapReduce Algorithmus vor. Wir konzipieren eine geeignete, Cluster-basierte Infrastruktur für ein neues *Spatial Decision Support System* und legen die Notwendigkeit einer deklarativen, domänenspezifischen Anfragesprache dar.

## Allgemeine Begriffe

Measurement, Performance, Languages.

## Stichworte

Cloud-Computing, MapReduce, Energiewende.

## 1. EINLEITUNG

Der Beschluss der Bundesregierung, zum Jahr 2022 aus der Kernenergie auszusteigen und deren Anteil am Strom-Mix durch erneuerbare Energien zu ersetzen, fordert einen rasanten Ausbau der erneuerbaren Energien. Entscheidend für den Bau neuer Windkraft- und Solaranlagen sind vor allem die zu erzielenden Gewinne und die Sicherheit der Investitionen. Somit sind präzise Ertragsprognosen von großer Bedeutung. Unterschiedliche Standorte sind zu vergleichen, die Ausrichtung der Windkraftanlagen zueinander in den Windparks ist sorgfältig zu planen. Als Entscheidungsgrundlage dienen hierzu vor allem historische Wetterdaten. Für die Kalkulation des Ertrags von Windkraftanlagen muss effizient auf die Datenbasis zugegriffen werden können. Diese erstreckt sich über große Zeiträume, da das Windaufkommen nicht nur jährlich schwankt, sondern auch dekadenweise variiert [3, 9].

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

Die Standortfindung etwa für Bankfilialen und die Zonierung, also das Ausweisen geographischer Flächen für die Landwirtschaft, sind klassische Fragestellungen für räumliche Entscheidungsunterstützungssysteme [6].

Die Herausforderungen an solch ein *Spatial Decision Support System* im Kontext der Energiewende sind vielfältig:

1. Verarbeitung heterogener Datenformate.
2. Skalierbare Anfragebearbeitung auf *Big Data*.
3. Eine elastische Infrastruktur, die mit der Erschließung neuer Datenquellen ausgebaut werden kann.
4. Eine deklarative, domänenspezifische Anfragesprache für komplexe *ad-hoc* Anfragen.

Wir begründen kurz die Eckpunkte dieses Anforderungsprofils im Einzelnen. Dabei vertreten wir den Standpunkt, dass existierende Entscheidungsunterstützungssysteme auf Basis relationaler Datenbanken diese nicht in allen Punkten erfüllen können.

(1) Historische Wetterdaten sind zum Teil öffentlich zugänglich, werden aber auch von kommerziellen Anbietern bezogen. Prominente Vertreter sind das *National Center for Atmospheric Research* [12] in Boulder Colorado, der *Deutsche Wetterdienst* [7] und die *Satel-Light* [14] Datenbank der Europäischen Union. Hinzu kommen Messwerte der hochschuleigenen experimentellen Windkraft- und Solaranlagen. Die Vielzahl der Quellen und somit der Formate führen zu den klassischen Problemen der Datenintegration.

(2) Daten in hoher zeitlicher Auflösung, die über Jahrzehnte hinweg erhoben werden, verursachen Datenvolumina im *Big Data* Bereich. Der Deutsche Wetterdienst allein verwaltet ein Datenarchiv von 5 Petabyte [7]. Bei solchen Größenordnung haben sich NoSQL Datenbanken gegenüber relationalen Datenbanken bewährt [4].

(3) Wir stellen die Infrastruktur für ein interdisziplinäres Team der *Regensburg School of Energy and Resources* mit mehreren im Aufbau befindlichen Projekten bereit. Um den wachsenden Anforderungen unserer Nutzer gerecht werden zu können, muss das System elastisch auf neue Datenquellen und neue Nutzergruppen angepasst werden können.

(4) Unsere Nutzer sind überwiegend IT-affin, doch nicht erfahren in der Entwicklung komplexer verteilter Systeme. Mit einer domänenspezifischen Anfragesprache wollen die Autoren dieses Artikels die intuitive Nutzbarkeit des Systems gewährleisten.

Unter diesen Gesichtspunkten konzipieren wir unser System als Hadoop-Rechencluster [1, 5]. Damit sind die Skalierbarkeit auf große Datenmengen (2) und die horizontale Skalierbarkeit der Hardware gegeben (3). Da auf historische Daten ausschließlich lesend zugegriffen wird, bietet sich der MapReduce Ansatz geradezu an. Zudem erlaubt Hadoop das Verarbeiten unstrukturierter, heterogener Daten (1). Der Entwurf einer eigenen Anfragesprache (4) stellt dabei eine spannende und konzeptionelle Herausforderung dar, weil hierfür ein tiefes Verständnis für die Fragestellungen der Nutzer erforderlich ist.

*Struktur.* Die folgenden Kapitel liefern Details zu unserem Vorhaben. In Kapitel 2 beschreiben wir eine konkrete Fragestellung bei der Standortfindung von Windkraftwerken. In Kapitel 3 stellen wir unsere Lösung als MapReduce Algorithmus dar. Kapitel 4 skizziert unsere Infrastruktur. Im 6. Kapitel wird auf verwandte Arbeiten eingegangen. Das letzte Kapitel gibt eine Zusammenfassung unserer Arbeit und zeigt deren Perspektive auf.

## 2. WINDPOTENTIALANALYSE

Ein aktuelles Forschungsprojekt der Hochschule Regensburg beschäftigt sich mit der Potentialanalyse von Windkraftanlagen. Hier werden die wirtschaftlichen Aspekte, die für das Errichten neuer Windkraftanlagen entscheidend sind, untersucht. Mithilfe der prognostizierten Volllaststunden einer Windkraftanlage kann eine Aussage über die Rentabilität getroffen werden. Diese ist bestimmt durch die Leistungskennlinie der Windkraftanlage und letztlich durch die zu erwartenden Windgeschwindigkeiten.

Abbildung 1 (aus [9]) skizziert die spezifische Leistungskennlinie einer Windkraftanlage in vier Phasen:

I) Erst ab einer gewissen Windgeschwindigkeit beginnt die Anlage Strom zu produzieren.

II) Die Leistung steigt über den wichtigsten Arbeitsbereich in der dritten Potenz zur Windgeschwindigkeit an, bis die Nennleistung der Anlage erreicht ist.

III) Die Ausgangsleistung wird auf die Nennleistung der Anlage begrenzt. Ausschlaggebend für die Höhe der Nennleistung ist die Auslegungsgröße des Generators.

IV) Die Windkraftanlage schaltet sich bei zu hohen Windgeschwindigkeiten ab, um eine mechanische Überbelastung zu verhindern.

Wie Abbildung 1 verdeutlicht, ist zum Errechnen der abgegeben Arbeit einer Windkraftanlage eine genaue Kenntnis der stochastischen Verteilung der Windgeschwindigkeit [1] notwendig. Mithilfe entsprechender Histogramme können somit potentielle Standorte für neue Windkraftanlagen verglichen, und Anlagen mit geeigneter Leistungskennlinie passend für den spezifischen Standort ausgewählt werden.

Als Datenbasis eignen sich etwa die Wetterdaten des Forschungsinstitut des *National Center for Atmospheric Research* [12] und die des Deutschen Wetterdienstes [7].

Insbesondere im Binnenland ist eine hohe räumliche Auflösung der meteorologischen Daten wichtig. Aufgrund der

[1]Wir verwenden die Begriffe Windgeschwindigkeit und Windstärke synonym. Streng genommen wird die Windgeschwindigkeit als Vektor dargestellt, während die Windstärke als skalare Größe erfasst wird. Dabei kann die Windstärke aus der Windgeschwindigkeit errechnet werden.

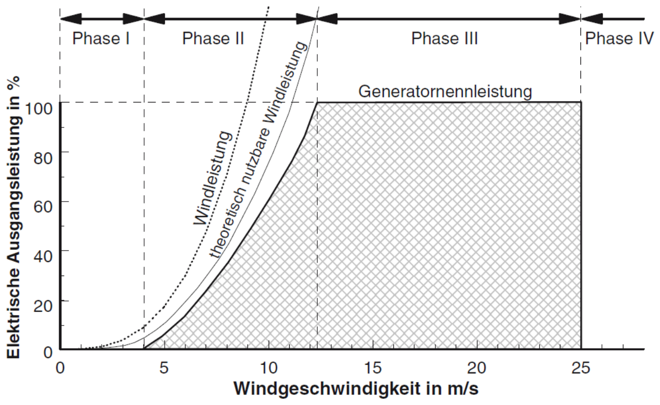


**Abbildung 1: Aussagen über die Leistung in Abhängigkeit zur Windgeschwindigkeit (aus [9]).**

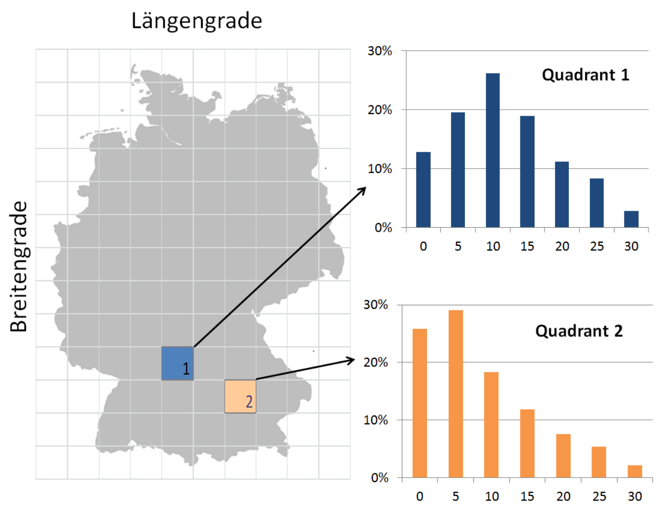


**Abbildung 2: Histogramme über die Windstärkeverteilung.**

Orographie variieren die Windgeschwindigkeiten schon bei kurzen Distanzen stark.

Abbildung 2 skizziert die resultierende Aufgabenstellung: Geographische Flächen werden kleinräumig unterteilt, was die Abbildung aus Gründen der Anschaulichkeit stark vereinfacht darstellt. Für jeden Quadranten, bestimmt durch Längen- und Breitengrad, interessiert die Häufigkeitsverteilung der Windstärken (dargestellt als Histogramm).

Je nach Fragestellung wird von unterschiedlichen Zeiträumen und unterschiedlicher Granularität der Quadranten ausgegangen. Aufgrund der schieren Größe der Datenbasis ist hier ein massiv paralleler Rechenansatz gefordert, wenn über eine Vielzahl von Quadranten hinweg Histogramme berechnet werden sollen.

## 3. MASSIV PARALLELE HISTOGRAMMBERECHNUNG

Im Folgenden stellen wir einen MapReduce Algorithmus zur parallelen Berechnung von Windgeschwindigkeitsverteilungen vor. Wir betrachten dabei die Plattform Apache Ha-

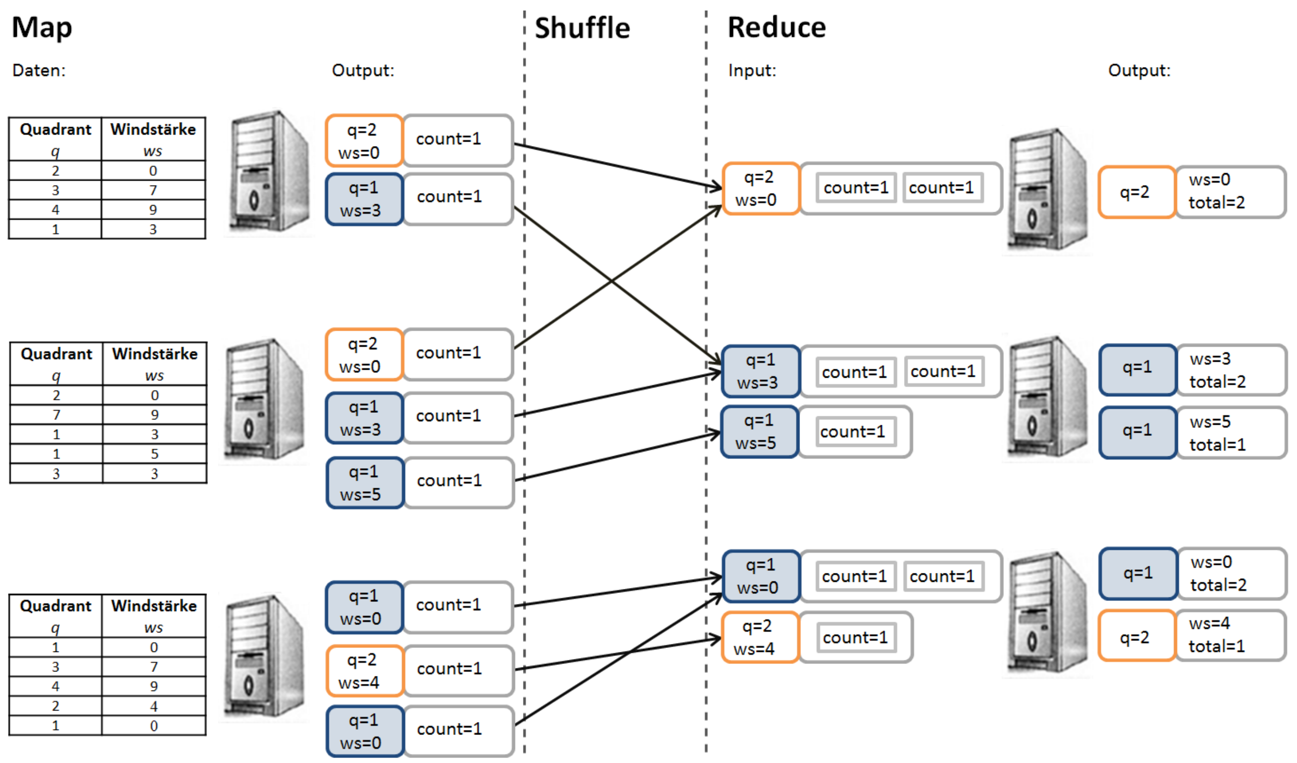


**Abbildung 3: Erste MapReduce-Sequenz zur Berechnung der absoluten Häufigkeiten.**

doop [1], eine quelloffene MapReduce Implementierung [5].

Hadoop ist dafür ausgelegt, mit großen Datenmengen umzugehen. Ein intuitives Programmierparadigma erlaubt es, massiv parallele Datenverarbeitungsschritte zu spezifizieren. Die Plattform partitioniert die Eingabe in kleinere Datenblöcke und verteilt diese redundant auf dem *Hadoop Distributed File System* [15]. Dadurch wird eine hohe Datensicherheit gewährleistet. Als logische Basiseinheit arbeitet Hadoop mit einfachen Schlüssel/Werte Paaren. Somit können selbst unstrukturierte oder nur schwach strukturierte Daten *ad hoc* verarbeitet werden.

MapReduce Programme werden in drei Phasen ausgeführt.

1. In der ersten Phase wird auf den partitionierten Eingabedaten eine Map-Funktion parallel ausgeführt. Diese Map-Funktion transformiert einfache Schlüssel/Werte Paare in eine Liste von neuen Schlüssel/Werte Paaren.

2. Die anschließende Shuffle-Phase verteilt die entstandenen Tupel so um, dass nun alle Paare mit dem gleichen Schlüssel an demselben Rechner vorliegen.

3. Die Reduce-Phase berechnet meist eine Aggregatfunktion auf allen Tupeln mit demselben Schlüssel.

Die Signaturen der Map- und Reduce-Funktion werden üblicherweise wie folgt beschrieben [11]:

| | | | |
|---|---|---|---|
| **Map:** | (k1, v1) | $\rightarrow$ | list(k2, v2) |
| **Reduce:** | (k2, list(v2)) | $\rightarrow$ | list(k3, v3) |

Wir erläutern nun unseren MapReduce Algorithmus zum Erstellen von Histogrammen der Windgeschwindigkeitsverteilungen. Im Sinne einer anschaulichen Darstellung abstrahieren wir von dem tatsächlichen Eingabeformat und beschränken uns auf nur eine Datenquelle. Die Eingabetupel enthalten einen Zeitstempel, den Längen- und Breitengrad als Ortsangabe und diverse Messwerte.

Wir nehmen vereinfachend an, dass die Ortsangabe bereits in eine Quadranten-ID übersetzt ist. Diese Vereinfachung erlaubt eine übersichtlichere Darstellung, gleichzeitig ist die Klassifikation der Datensätze nach Quadranten einfach umzusetzen. Zudem ignorieren wir alle Messwerte bis auf die Windstärke. Tabelle 1 zeigt exemplarisch einige Datensätze, die wir in unserem laufenden Beispiel verarbeiten.

Wir betonen an dieser Stelle, dass diese vereinfachenden Annahmen nur der Anschaulichkeit dienen und keine Einschränkung unseres Systems darstellen.

| Quadrant<br>*q* | Windstärke<br>*ws* |
|---|---|
| 2 | 0 |
| 3 | 7 |
| 4 | 9 |
| 1 | 3 |
| ... | ... |

**Tabelle 1: Tabellarisch dargestellte Eingabedaten.**

Wir schalten zwei MapReduce-Sequenzen in Reihe:

- Die erste Sequenz ermittelt, wie oft in einem Quadranten eine konkrete Windstärke aufgetreten ist.
- Die zweite Sequenz fasst die berechneten Tupel zu Histogrammen zusammen.

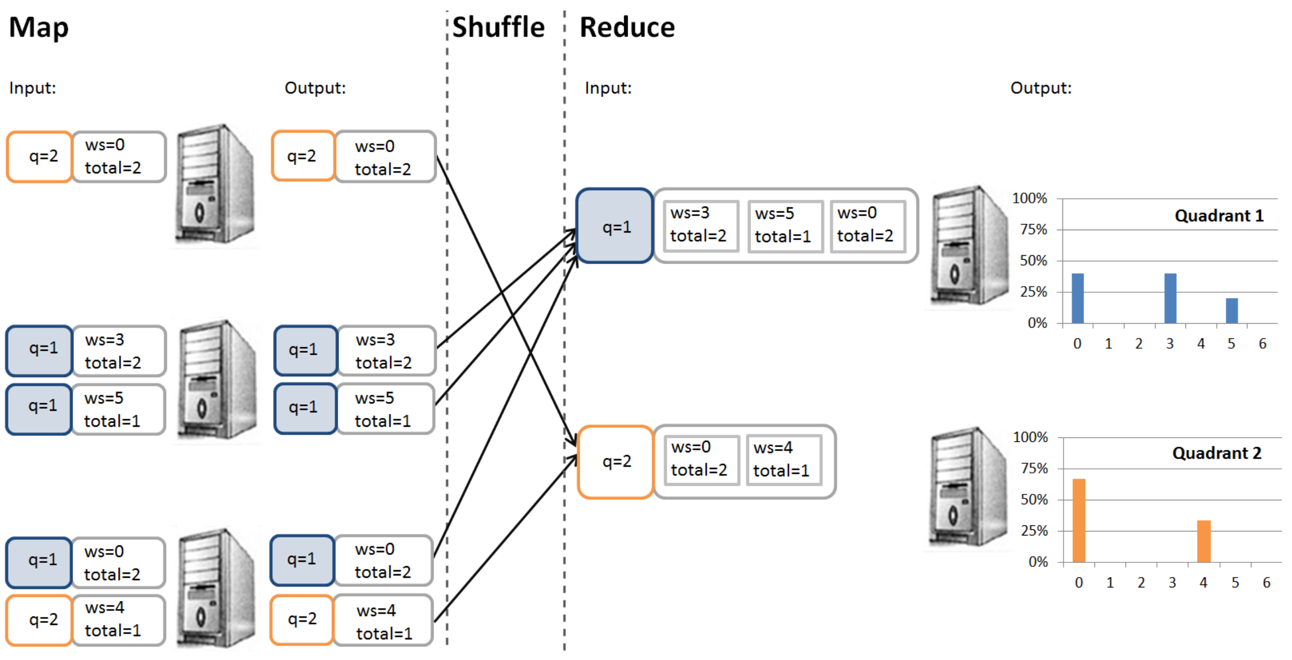


**Abbildung 4: Zweite MapReduce-Sequenz zur Berechnung der Histogramme.**

```
def map(Datei d, Liste<Quadrant, Windstärke> L) :
    foreach (q, ws) in L do
        if ($q \in Q$)
            int count = 1;
            emit ((q, ws), count);
        fi
    od
```

**Abbildung 5: Map-Funktion der ersten Sequenz.**

```
def reduce((Quadrant q, Windstärke ws), Liste<Integer> L) :
    int total = 0;
    foreach count in L do
        total += count;
    od
    emit (q, (ws, total));
```

**Abbildung 6: Reduce-Funktion der ersten Sequenz.**

Durch das Aneinanderreihen von MapReduce-Sequenzen werden ganz im Sinne des Prinzips „teile und herrsche“ mit sehr einfachen und gut parallelisierbaren Rechenschritten komplexe Transformationen spezifiziert. Wir betrachten nun beide Sequenzen im Detail.

## 3.1 Sequenz 1: Absolute Häufigkeiten

Die erste Sequenz erinnert an das „WordCount“ -Beispiel, das klassische Einsteigerbeispiel für MapReduce Programmierung [11]. Die Eingabe der Map-Funktion ist der Name einer Datei und deren Inhalt, nämlich eine Liste der Quadranten und der darin gemessenen Windstärke. Wir nehmen an, dass nur eine ausgewählte Menge von Quadranten Q interessiert, etwa um mögliche Standorte von Windkraftanlagen im Regensburger Raum zu untersuchen.

Abbildung 5 zeigt die Map-Funktion in Pseudocode. Die Anweisung *emit* produziert ein neues Ausgabetupel. In der Shuffle-Phase werden die produzierten Tupel nach der Schlüsselkomponente aus Quadrant und Windstärke umverteilt. Die Reduce-Funktion in Abbildung 6 berechnet nun die Häufigkeit der einzelnen Windstärkewerte pro Quadrant.

Beispiel 1. Abbildung 3 visualisiert die erste Sequenz anhand konkreter Eingabedaten. Die Map-Funktion selektiert nur Tupel aus den Quadranten 1 und 2 (d.h. Q = {1, 2}). Die Shuffle-Phase reorganisiert die Tupel so, dass anschließend alle Tupel mit den gleichen Werten für Quadrant und Windstärke bei demselben Rechner vorliegen. Hadoop fasst dabei die count-Werte bereits zu einer Liste zusammen.

Die Reduce-Funktion produziert daraus Tupel mit dem Quadranten als Schlüssel. Der Wert setzt sich aus der Windstärke und ihrer absoluten Häufigkeit zusammen. □

## 3.2 Sequenz 2: Histogramm-Berechnung

Die Ausgabe der ersten Sequenz wird nun weiter verarbeitet. Die Map-Funktion der zweiten Sequenz ist schlicht die Identitätsfunktion. Die Shuffle-Phase gruppiert die Tupel nach dem Quadranten. Somit findet die finale Erstellung der Histogramme in der Reduce-Funktion statt.

Beispiel 2. Abbildung 4 zeigt für das laufende Beispiel die Verarbeitungsschritte der zweiten Sequenz. □

## 4. ARCHITEKTURBESCHREIBUNG

Unsere Vision eines Cloud-basierten *Spatial Decision Support Systems* für die Fragestellungen der Energiewende fußt fest auf MapReduce Technologie.

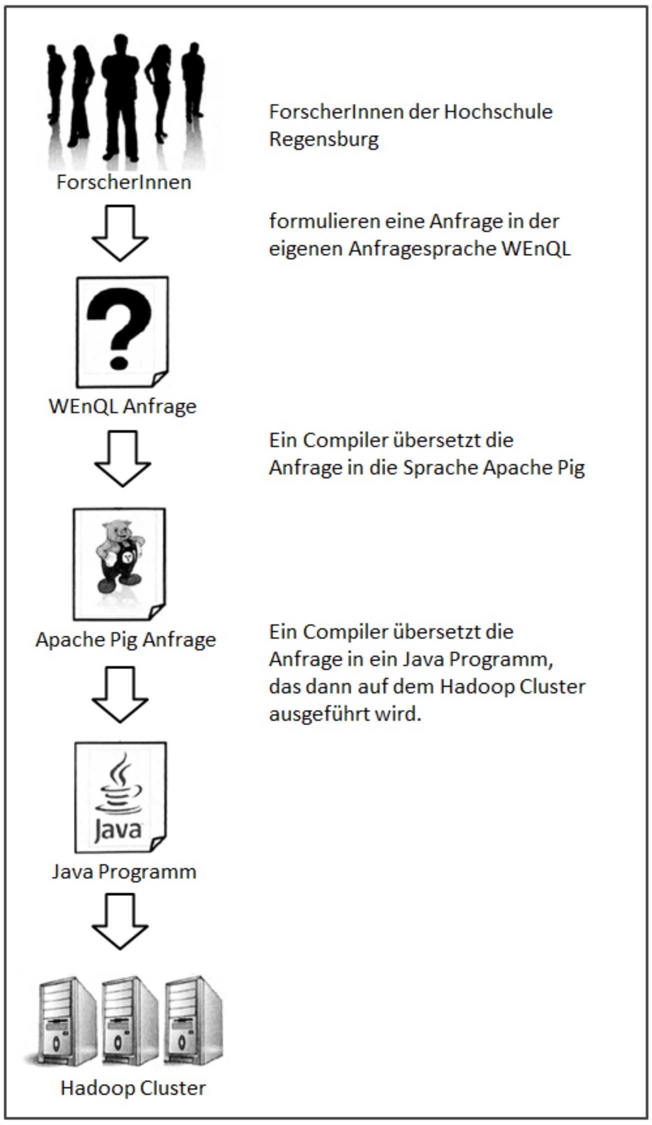

ForscherInnen
ForscherInnen der Hochschule Regensburg
formulieren eine Anfrage in der eigenen Anfragesprache WEnQL
WEnQL Anfrage
Ein Compiler übersetzt die Anfrage in die Sprache Apache Pig
Apache Pig Anfrage
Ein Compiler übersetzt die Anfrage in ein Java Programm, das dann auf dem Hadoop Cluster ausgeführt wird.
Java
Java Programm
Hadoop Cluster

**Abbildung 7: Architektur des Cloud-basierten Spatial Decision Support Systems.**

Das Projektvorhaben geht dabei über den bloßen Einsatz von Cluster-Computing hinaus. Das Ziel ist der Entwurf einer domänenspezifischen Anfragesprache *WEnQL*, die *„Wetter und Energie Query Language“*. Diese wird in interdisziplinärer Zusammenarbeit mit dem Forschungsinstitut *Regensburg School of Energy and Resources* entwickelt. Mit ihr sollen auch MapReduce Laien in der Lage sein, Algorithmen auf dem Cluster laufen zu lassen.

Abbildung 7 illustriert die Vision: Unsere Nutzer formulieren eine deklarative Anfrage in WenQL, etwa um die Wind geschwindigkeits-Histogramme einer Region zu berechnen. Ein eigener Compiler übersetzt die WenQL Anfrage in das gängige Anfrageformat Apache Pig [8, 13], das wiederum nach Java übersetzt wird. Das Java Programm wird anschließend kompiliert und auf dem Hadoop Cluster ausgeführt.

Die Übersetzung in zwei Schritten hat den Vorteil, dass das Programm in der Zwischensprache Pig von Experten hinsichtlich Performanz und Korrektheit analysiert werden kann. Hadoop Laien hingegen brauchen sich mit diesen Interna nicht zu belasten. Aktuell erarbeiten wir einen Katalog konkreter Fragestellungen der Energiewirtschaft, um gängige Query-Bausteine für WenQL zu identifizieren.

## 5. VERWANDTE ARBEITEN

In diversen Forschungsgebieten der Informatik finden sich Berührungspunkte mit unserer Arbeit. Viele Forschungsprojekte, die sich mit der *Smart Grid* Technologie beschäftigen, setzten auf distributive Systeme zum Bearbeiten ihrer Daten [4, 17]. Ähnlich wie in unserem Projekt wird diese Entscheidung aufgrund der großen Datenmengen getroffen, welche aus unterschiedlichen Quellen stammen. Wettereinflüsse auf Kraftwerke, Börsenstrompreise, Auslastung von Stromnetzen und das Stromverbrauchsverhalten von Millionen von Nutzern müssen verglichen werden. Smart Grid Analysen unterscheiden sich von unserem Projekt darin, dass wir nur auf historische Daten zurückgreifen und somit keine Echtzeitanforderungen an das System stellen.

Fragestellungen wie Standortbestimmung und Zonierung haben eine lange Tradition in der Entwicklung dedizierter *Spatial Decision Support* Systeme [6]. Deren Architekturen fußen üblicherweise auf relationalen Datenbanken zur Datenhaltung. Mit der Herausforderung, auf *Big Data* zu skalieren, und mit heterogenen Datenquellen zu arbeiten, besitzen *NoSQL* Systeme wie Hadoop und das Hadoop Dateisystem hingegen eindeutige Vorteile in der Datenhaltung und Anfragebearbeitung.

Die Definition deklarativer Anfragesprachen für MapReduce Algorithmen ist ein sehr aktives Forschungsgebiet. Am relevantesten für uns sind SQL-ähnliche Anfragesprachen, wie etwa im Hive Projekt umgesetzt [2, 16]. Allerdings wird SQL von unseren Anwendern in der Regel nicht beherrscht. Daher planen wir, eine eigene Anfragesprache zu definieren, die möglichst intuitiv für unsere Anwender zu erlernen ist.

## 6. ZUSAMMENFASSUNG UND AUSBLICK

Der Bedarf für eine neue, *BigData*-fähige Generation von räumlichen Entscheidungsunterstützungssystemen für diverse Fragestellungen der Energiewende ist gegeben.

In dieser Arbeit stellen wir unsere Vision eines Cloud-basierten *Spatial Decision Support Systems* vor. Anhand des Beispiels der Windpotentialanalyse zeigen wir die Einsatzfähigkeit von MapReduce Algorithmen für strategische Fragestellungen in der Energieforschung.

Wir sind zuversichtlich, ein breites Spektrum essentieller Entscheidungen unterstützen zu können. Eine Weiterführung unserer Fallstudie ist die Ausrichtung von Windkrafträdern innerhalb eines Windparks. Dafür ist die dominierende Windrichtung entscheidend, um Windkrafträder günstig zueinander und zur Hauptwindrichtung auszurichten. Ein einzelnes Windkraftwerk kann zwar die Gondel um 360° drehen, um die Rotoren gegen den Wind auszurichten. Die Anordnung der Türme zueinander im Park ist allerdings fixiert. Bei einem ungünstigen Layout der Türme können somit Windschatteneffekte die Rendite nachhaltig schmälern. Abbildung 8 (aus [10]) visualisiert die Windstärke und Windrichtung als Entscheidungsgrundlage. Unser MapReduce Algorithmus aus Kapitel 3 kann dementsprechend erweitert werden.

Darüber hinaus eruieren wir derzeit die Standortbestim-

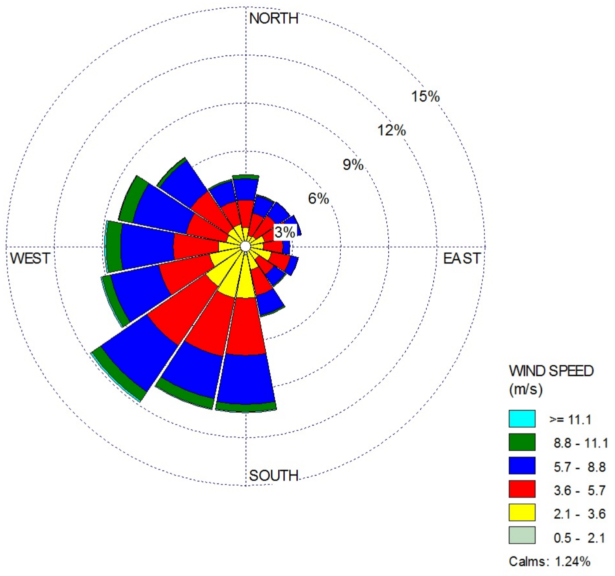


**Abbildung 8: Aussagen über Windstärke und Windrichtung (aus [10]).**

mung von Solaranlagen, und noch komplexer, den strategischen Einsatz von Energiespeichern, um etwa Windstillen oder Nachtphasen ausgleichen zu können.

Mit den Fähigkeiten unseres künftigen *Decision Support Systems*, der Skalierbarkeit auf sehr großen Datenmengen, dem flexible Umgang mit heterogenen Datenformaten und nicht zuletzt mit einer domänenspezifischen Anfragesprache wollen wir unseren Beitrag zu einer klimafreundlichen und nachhaltigen Energieversorgung leisten.

## 7. DANKSAGUNG

Diese Arbeit ist ein Projekt der *Regensburg School of Energy and Resources*, eine interdisziplinäre Einrichtung der Hochschule Regensburg und des *Technologie- und Wissenschaftsnetzwerkes Oberpfalz*.

## 8. LITERATUR

[1] *Apache Hadoop*. http://hadoop.apache.org/, 2013.

[2] *Apache Hive*. http://hive.apache.org/, 2013.

[3] O. Brückl. *Meteorologische Grundlagen der Windenergienutzung*. Vorlesungsskript: Windenergie. Hochschule Regensburg, 2012.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[6] P. J. Densham. Spatial decision support systems. *Geographical information systems: Principles and applications*, 1:403–412, 1991.

[7] *Deutscher Wetterdienst*. http://www.dwd.de/, 2013.

[8] A. Gates. *Programming Pig*. O'Reilly Media, 2011.

[9] M. Kaltschmitt, W. Streicher, and A. Wiese. *Erneuerbare Energien Systemtechnik, Wirtschaftlichkeit, Umweltaspekte*. Springer, 2006.

[10] *Lakes Environmental Software*. http://www.weblakes.com/, 2013.

[11] C. Lam. *Hadoop in Action*. Manning Publications, 2010.

[12] *National Center for Atmospheric Research (NCAR)*. http://ncar.ucar.edu/,, 2013.

[13] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[14] *Satel-Light*. http://www.satel-light.com/, 2013.

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, 2010.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[17] D. Wang and L. Xiao. Storage and Query of Condition Monitoring Data in Smart Grid Based on Hadoop. In *Computational and Information Sciences (ICCIS), 2012 Fourth International Conference*, pages 377–380. IEEE, 2012.

# Consistency Models for Cloud-based Online Games: the Storage System's Perspective

Ziqiang Diao
Otto-von-Guericke University Magdeburg
39106 Magdeburg, Germany
diao@iti.cs.uni-magdeburg.de

## ABSTRACT

The existing architecture for massively multiplayer online role-playing games (MMORPG) based on RDBMS limits the availability and scalability. With increasing numbers of players, the storage systems become bottlenecks. Although a Cloud-based architecture has the ability to solve these specific issues, the support for data consistency becomes a new open issue. In this paper, we will analyze the data consistency requirements in MMORPGs from the storage system point of view, and highlight the drawbacks of Cassandra to support of game consistency. A timestamp-based solution will be proposed to address this issue. Accordingly, we will present data replication strategies, concurrency control, and system reliability as well.

## 1. INTRODUCTION

In massively multiplayer online role-playing games (MMORPG) thousands of players can cooperate with other players in a virtual game world. To support such a huge game world following often complex application logic and specific requirements. Additionally, we have to bear the burden of managing large amounts of data. The root of the issue is that the existing architectures of MMORPGs use RDBMS to manage data, which limits the availability and scalability.

Cloud data storage systems are designed for internet applications, and are complementary to RDBMS. For example, Cloud systems are able to support system availability and scalability well, but not data consistency. In order to take advantages of these two types of storage systems, we have classified data in MMORPGs into four data sets according to typical data management requirements (e.g., data consistency, system availability, system scalability, data model, security, and real-time processing) in [4]: account data, game data, state data, and log data. Then, we have proposed to apply multiple data management systems (or services) in one MMORPG, and manage diverse data sets accordingly. Data with strong requirements for data consistency and security (e.g., account data) is still managed by RDBMS, while data

25rd GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2011, Ilmenau, Deutschland.

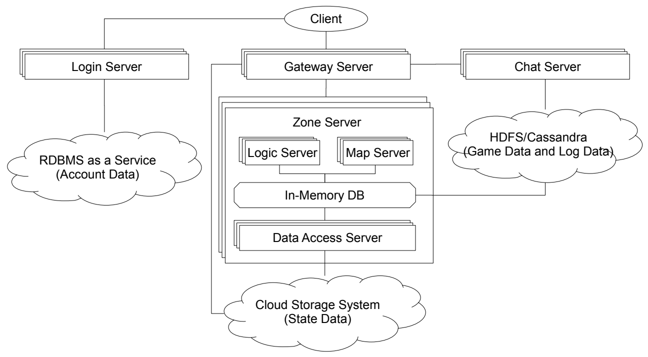


**Figure 1: Cloud-based Architecture of MMORPGs [4]**

(e.g., log data and state data) that requires scalability and availability is stored in a Cloud data storage system (Cassandra, in this paper). Figure 1 shows the new architecture.

Unfortunately, there are still some open issues, such as the support of data consistency. According to the CAP theorem, in a partition tolerant distributed system (e.g., an MMORPG), we have to sacrifice one of the two properties: consistency or availability [5]. If an online game does not guarantee availability, players' requests may fail. If data is inconsistent, players may get data not conforming to game logic, which affects their operations. For this reason, we must analyze the data consistency requirements of MMORPGs so as to find a balance between data consistency and system availability.

Although there has been some research work focused on the data consistency model of online games, the researchers generally discussed it from players' or servers' point of view [15, 9, 11], which actually are only related to data synchronization among players. Another existing research work did not process diverse data accordingly [3], or just handled this issue based on a rough classification of data [16]. However, we believe the only efficient way to solve this issue is to analyze the consistency requirements of each data set from the storage system's perspective. Hence, we organize the rest of this paper as follows: in Section 2, we highlight data consistency requirements of the four data sets. In Section 3, we discuss the data consistency issue of our Cloud-based architecture. We explain our timestamp-based solution in detail from Section 4 to Section 6. Then, we point out some optimization programs and our future work in Section 7. Finally, we summarize this paper in Section 8.

## 2. CONSISTENCY REQUIREMENTS OF DIVERSE DATA IN MMORPGS

Due to different application scenarios, the four data sets have distinct data consistency requirements. For this reason, we need to apply different consistency models to fulfill them.

**Account data:** is stored on the server side, and is created, accessed as well as deleted when players log in to or log out of a game. It includes player's private data and some other sensitive information (e.g., user ID, password, and recharge records). The inconsistency of account data might bring troubles to a player as well as the game provider, or even lead to an economic or legal dispute. Imagine the following two scenarios: a player has changed the password successfully. However, when this player log in to the game again, the new password is not effective; a player has transferred to the game account, or the player has consumed in the game, but the account balance is somehow not properly presented in the game system. Both cases would influence on the player's experience, and might result in the customer or the economic loss of a game company. Hence, we need to access account data under strong consistency guarantees, and manage it with transactions. In a distributed database system, it means that each copy should hold the same view on the data value.

**Game data:** such as world appearance, metadata (name, race, appearance, etc.) of NPC (Non Player Character), system configuration files, and game rules, is used by players and game engine in the entire game, which can only be modified by game developers. Players are not as sensitive to game data as to account data. For example, the change of an NPC's appearance or name, the duration of a bird animation, and the game interface may not catch the players' attention and have no influence on players' operations. As a result, it seems that strong consistency for game data is not so necessary. On the other hand, some changes of the game data must be propagated to all online players synchronously, for instance, the change of the game world's appearance, the power of a weapon or an NPC, game rules as well as scripts, and the occurrence frequency of an object during the game. The inconsistency of these data will lead to errors on the game display and logic, unfair competition among players, or even a server failure. For this reason, we also need to treat data consistency of game data seriously. Game data could be stored on both the server side and the client side, so we have to deal with it accordingly.

Game data on the client side could only synchronize with servers when a player logs in to or starts a game. For this reason, causal consistency is required [8, 13]. In this paper, it means when player A uses client software or browser to connect with the game server, the game server will then transmit the latest game data in the form of data packets to the client side of player A. In this case, the subsequent local access by player A is able to return the updated value. Player B that has not communicated with the game server will still retain the outdated game data.

Although both client side and server side store the game data, only the game server maintains the authority of it. Furthermore, players in different game worlds cannot communicate to each other. Therefore, we only need to ensure that the game data is consistent in one zone server so that players in the same game world could be treated equally. It is noteworthy that a zone server accesses data generally from one data center. Hence, we guarantee strong consistency within one data center, and causal consistency among data centers. In other words, when game developers modify the game data, the updated value should be submitted synchronously to all replicas within the same data center, and then propagated asynchronously across data centers.

**State data:** for instance, metadata of PCs (Player Characters) and state (e.g., position, task, or inventory) of characters, is modified by players frequently during the game. The change of state data must be perceived by all relevant players synchronously, so that players and NPCs can respond correctly and timely. An example for the necessity of data synchronization is that players cannot tolerate that a dead character can continue to attack other characters. Note that players only access data from the in-memory database during the game. Hence, we need to ensure strong consistency in the in-memory database.

Another point about managing state data is that updated values must be backed up to the disk-resident database asynchronously. Similarly, game developers also need to take care of data consistency and durability in the disk-resident database, for instance, it is intolerable for a player to find that her/his last game record is lost when she/he starts the game again. In contrast to that in the in-memory database, we do not recommend ensuring strong consistency to state data. The reason is as follows: according to the CAP theorem, a distributed database system can only simultaneously satisfy two of three the following desirable properties: consistency, availability, and partition tolerance. Certainly, we hope to satisfy both consistency and availability guarantees. However, in the case of network partition or under high network latency, we have to sacrifice one of them. Obviously, we do not want all update operations to be blocked until the system recovery, which may lead to data loss. Consequently, the level of data consistency should be reduced. We propose to ensure read-your-writes consistency guarantee [13]. In this paper, it describes that once state data of player A has been persisted in the Cloud, the subsequent read request of player A will receive the updated values, yet other players (or the game engine) may only obtain an outdated version of it. From the storage system's perspective, as long as a quorum of replicas has been updated successfully, the commit operation is considered complete. In this case, the storage system needs to provide a solution to return the up-to-date data to player A. We will discuss it in the next section.

**Log data:** (e.g., player chat history and operation logs) is created by players, but used by data analysts for the purpose of data mining. This data will be sorted and cached on the server side during the game, and then bulk stored into the database, thereby reducing the conflict rate as well as the I/O workload, and increasing the total simultaneous throughput [2]. The management of log data has three features: log data will be appended continually, and its value will not be modified once it is written to the database; The replication of log data from thousands of players to multiple nodes will significantly increase the network traffic and even block the network; Moreover, log data is generally organized and analyzed after a long time. Data analysts are only concerned about the continuous sequence of the data, rather than the timeliness of the data. Hence, data inconsistency is accepted in a period of time. For these three reasons,

| | **Account data** | **Game data** | | | **State data** | | **Log data** |
|---|---|---|---|---|---|---|---|
| **Modified by** | Players | Game developers | | | Players | | Players |
| **Utilized by** | Players & Game engine | Players & Game engine | | | Players & Game engine | | Data analysts |
| **Stored in** | Cloud | Client side | Cloud | | In-memory DB | Cloud | Cloud |
| **Data center** | Across | — | Single | Across | Single | Across | Across |
| **Consistency model** | Strong consistency | Causal consistency | Strong consistency | Causal consistency | Strong consistency | Read-your-writes consistency | Timed consistency |

**Table 1: Consistency requirements**

a deadline-based consistency model, such as timed consistency, is more suitable for log data[12, 10]. In this paper, timed consistency specifically means that update operations are performed on a quorum of replicas instantaneously at time t, and then the updated values will be propagated to all the other replicas within a time bounded by t + $\triangle$ [10]. Additionally, to maintain the linear order of the log data, the new value needs to be sorted with original values before being appended to a replica. In other words, we execute a sort-merge join by the timestamp when two replicas are asynchronous. Under timed consistency guarantee, data analysts can at time t + $\triangle$ obtain a continuously sequential log data until time t.

## 3. OPPORTUNITIES AND CHALLENGES

In our previous work, we have already presented the capability of the given Cloud-based architecture to support the corresponding consistency model for each data set in MMORPGs [4]. However, we also have pointed out that to ensure read-your-writes consistency to state data and timed consistency to log data efficiently in Cassandra is an open issue. In this section, we aim at discussing it in detail.

Through customizing the quorum of replicas in response to read and write operations, Cassandra provides tunable consistency, which is an inherent advantage to support MMORPGs [7, 4]. There are two reasons: first, as long as a write request receives a quorum of responses, it completes successfully. In this case, although data in Cassandra is inconsistent, it reduces the response time of write operations, and ensures availability as well as fault tolerance of the system; Additionally, a read request will be sent to the closest replica, or routed to a quorum or all replicas according to the consistency requirement of the client. For example, if a write request is accepted by three (N, N> 0) of all five (M, M>= N) replicas, at least three replicas (M-N+1) need to respond to the subsequent read request, so that the up-to-date data can be returned. At this case, Cassandra can guarantee read-your-writes consistency or strong consistency. Otherwise, it can only guarantee timed consistency or eventual consistency [7, 13]. Due to the support of tunable consistency, Cassandra has the potential to manage state data and log data of MMORPGs simultaneously, and is more suitable than some other Cloud storage systems that only provide either strong or eventual consistency guarantees.

On the other hand, Cassandra fails to implement tunable consistency efficiently according to MMORPG requirements. For example, M-N+1 replicas of state data have to be compared so as to guarantee read-your-writes consistency. However, state data has typically hundreds of attributes, the transmission and comparison of which affect the read performance. Opposite to update a quorum of replicas, we update all replicas while executing write operations. In this case, data in Cassandra is consistent, and we can obtain the up-to-date data from the closest replica directly. Unfortunately, this replication strategy significantly increases the network traffic as well as the response time of write operations, and sacrifices system availability. As a result, to implement read-your-writes consistency efficiently becomes an open issue.

Another drawback is that Cassandra makes all replicas eventually consistent, which sometimes does not match the application scenarios of MMORPG, and reduce the efficiency of the system. The reasons are as follows.

- Unnecessary for State data: state data of a PC is read by a player from the Cloud storage system only once during the game. The subsequent write operations do not depend on values in the Cloud any more. Hence, after obtaining the up-to-date data from the Cloud, there is no necessity to ensure that all replicas reach a consensus on these values.
- Increase network traffic: Cassandra utilizes Read Repair functionality to guarantee eventual consistency [1]. It means that all replicas have to be compared in the background while executing a write operation in order to return the up-to-date data to players, detect the outdated data versions, and fix them. In MMORPGs, both state data and log data have a large scale, and are distributed in multiple data centers. Hence, transmission of these data across replicas will significantly increase the network traffic and affect the system performance.

## 4. A TIMESTAMP-BASED CONSISTENCY SOLUTION

A common method for solving the consistency problem of Cloud storage system is to build an extra transaction layer on top of the system [6, 3, 14]. Similarly, we have proposed a timestamp-based solution especially for MMORPG, which is designed based on the features of Cassandra [4]. Cassandra records timestamps in each column, and utilizes it as a version identification (ID). Therefore, we record the timestamps from a global server in both server side and in the Cloud storage system. When we read state data from the Cloud, the timestamps recorded on the server side will be sent with the read request. In this way, we can find out the most recent data easily. In the following sections, we will introduce this solution in detail.

### 4.1 Data Access Server

Data access servers are responsible for data exchange between the in-memory database and the Cloud storage sys-

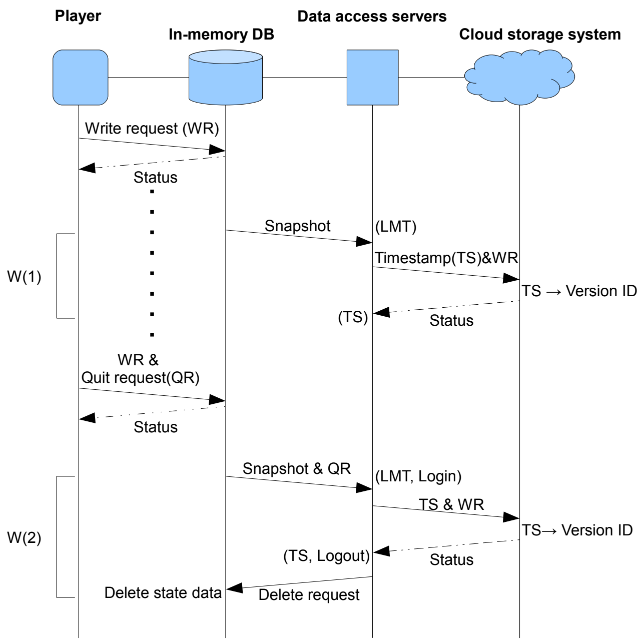


**Figure 2: Executions of Write Operations: W(1) describes a general backup operation; W(2) shows the process of data persistence when a player quits the game.**

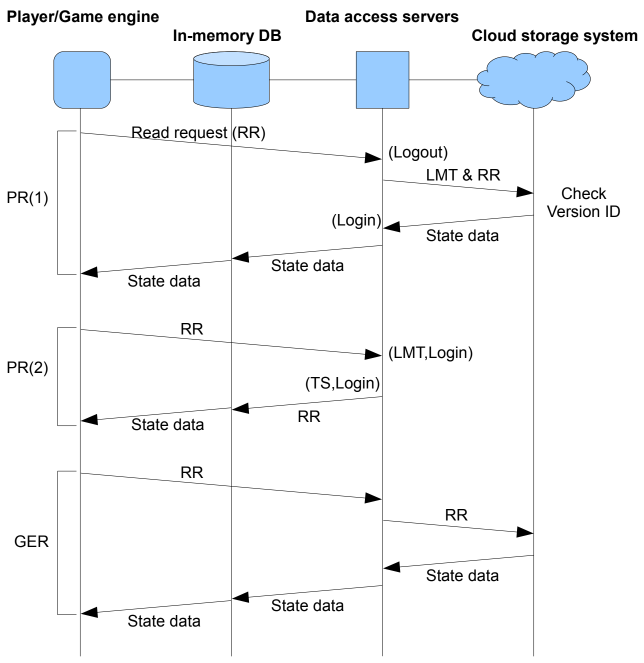


**Figure 3: Executions of Read Operations: PR(1) shows a general read request from the player; In the case of PR(2), the backup operation is not yet completed when the read request arrives; GER presents the execution of a read operation from the game engine.**

tem. They ensure the consistency of state data, maintain timestamp tables, and play the role of global counters as well. In order to balance the workload and prevent server failures, several data access servers run in one zone server in parallel. Data access servers need to synchronize their system clock with each other automatically. However, a complete synchronization is not required. A time difference less than the frequency of data backup is acceptable.

An important component in data access servers is the timestamp table, which stores the ID as well as the last modified time (LMT) of state data, and the log status (LS). If a character or an object in the game is active, its value of LS is "login". Otherwise, the value of LS is "logout". We utilize a hash function to map IDs of state data to distinct timestamp tables, which are distributed and partitioned in data access servers. It is noteworthy that timestamp tables are partitioned and managed by data access servers in parallel and data processing is simple, so that accessing timestamp tables will not become a bottleneck of the game system.

Note that players can only interact with each other in the same game world, which is managed by one zone server. Moreover, a player cannot switch the zone server freely. Therefore, data access servers as well as timestamp tables across zone servers are independent.

## 4.2 Data Access

In this subsection, we discuss the data access without considering data replication and concurrency conflicts.

In Figure 2, we show the storage process of state data in the new Cloud-based architecture: the in-memory database takes a consistent snapshot periodically. Though using the same hash function employed by timestamp tables, each data access server obtains the corresponding state data from the snapshot periodically. In order to reduce the I/O workload of the Cloud, a data access server generates one message including all its responsible state data as well as a new timestamp TS, and then sends it to the Cloud storage system. In the Cloud, this message is divided based on the ID of state data into several messages, each of which still includes TS. In this way, the update failure of one state data won't block the submission of other state data. Then, these messages are routed to appropriate nodes. When a node receives a message, it writes changes immediately into the commit log, updates data, and records TS as version ID in each column. If an update is successful and TS is higher than the existing LMT of this state data, then the data access server uses TS to replace the LMT. Note that if a player has quit the game and the state data of the relevant PC has backed up into the Cloud storage system, the LS of this PC needs to be modified form "login" to "logout", and the relevant state data in the in-memory database needs to be deleted.

Data access servers obtain log data not from the in-memory database, but from the client side. Log data also updates in batch, and gets timestamp from a data access server. When a node in the Cloud receives log data, it inserts log data into its value list according to the timestamp. However, timestamp tables are not modified when the update is complete.

Figure 3 presents executions of read operations. When a player starts the game, a data access server firstly obtains the LS information from the timestamp table. If the value is "login", that means the previous backup operation is not completed and the state data is still stored in the in-memory database. In this case, the player gets the state date from the in-memory database directly, and the data access server needs to generate a new timestamp to replace the LMT of the relevant state data; if the value is "logout", the data access server then gets the LMT, and sends it with a read request to the Cloud storage system. When the relevant node receives the request, it compares the LMT with its local version ID. If they match, the replica responds the read request immediately. If not match, this read request will be sent to other replicas (we will discuss it in detail in the section 5). When the data access server receives the state data, it sends it to the in-memory database as well as the relevant client sides, and modifies the LS from "logout" to "login" in the timestamp table. Note that state data may also be read by the game engine for the purpose of statistics. In this case, the up-to-date data is not necessary, so that we do not need to compare the LMT with the Version ID.

Data analysts read data also through data access servers. If a read request contains a timestamp T, the cloud storage system only returns log data until T-△ because it only guarantees log data timed consistency.

## 4.3 Concurrency Control

Concurrency conflicts appear rarely in the storage layer of MMORPGs: the probability of read-write conflicts is low because only state data with a specific version ID (the same as its LMT) will be read by players during the game, and a read request to log data does not return the up-to-date data. Certain data is periodically updated by only one data access server simultaneously. Therefore, write-write conflicts occur only when the per-update is not completed for some reason, for example, serious network latency, or a node failure. Fortunately, we can solve these conflicts easily by comparing timestamps. If two processes attempt to update the same state data, the process with higher timestamp wins, and another process should be canceled because it is out of date. If two processes intend to update the same log data, the process with lower timestamp wins, and another process enters the wait queue. The reason is that values contained in both processes must be stored in correct order.

## 5. DATA REPLICATION

Data in the Cloud typically has multiple replicas for the purpose of increasing data reliability as well as system availability, and balancing the node workload. On the other hand, data replication increases the response time and the network traffic as well, which cannot be handled well by Cassandra. For most of this section, we focus on resolving this contradiction according to access features of state data and log data.

## 5.1 Replication Strategies

Although state data is backed up periodically into the Cloud, only the last updated values will be read when players start the game again. It is noteworthy that the data loss in the server layer occurs infrequently. Therefore, we propose to synchronize only a quorum of replicas during the game, so that an update can complete effectively and won't block the subsequent updates. In addition, players usually start a game again after a period of time, so the system has enough time to store state data. For this reason, we propose to update all replicas synchronously when players quit the game. As a result, the subsequent read operation can obtain the updated values quickly.

While using our replication strategies, a replica may contain outdated data when it receives a read request. Though comparing LMT held by the read request with the Version ID in a replica, this case can be detected easily. Contrary to the existing approach of Cassandra (compares M-N+1 replicas and utilizes Read Repair), only the read request will be sent to other replicas until the lasted values was found. In this way, the network traffic will not be increased significantly, and the up-to-date data can also be found easily. However, if the read request comes from the game engine, the replica will respond immediately. These strategies ensure that this Cloud-based architecture can manage state data under read-your-writes consistency guarantees.

Similar to state data, a write request to log data is also accepted by a quorum of replicas at first. However, the updated values then must be propagated to other replicas asynchronously when the Cloud storage system is not busy, and arranged in order of timestamp within a predetermined time (△), which can be done with the help of Anti-Entropy functionality in Cassandra [1]. In this way, this Cloud storage system guarantees log data timed consistency.

## 5.2 Version Conflict Reconciliation

When the Cloud storage system detected a version conflict between two replicas: if it is state data, the replica with higher version ID wins, and values of another replica will be replaced by new values; if it is log data, these two replicas perform a sort-merge join by timestamps for the purpose of synchronization.

## 6. SYSTEM RELIABILITY

Our Cloud-based architecture for MMORPGs requires a mutual cooperation of multiple components. Unfortunately, each component has the possibility of failure. In the following, we discuss measures to deal with different failures.

Cloud storage system failure: the new architecture for MMORPGs is built based on Cassandra, which has the ability to deal with its own failure. For example, Cassandra applies comment logs to recover nodes. It is noteworthy that by using our timestamp-based solution, when a failed node comes back up, it could be regarded as an asynchronous node. Therefore, the node recovery as well as response to write and read requests can perform simultaneously.

In-memory database failure: similarly, we can also apply comment logs to handle this kind of failure so that there is no data loss. However, writing logs affects the real-time response. Moreover, logs are useless when changes are persisted in the Cloud. Hence, we have to find a solution in our future work.

Data access server failure: If all data access servers crash, the game can still keep running, whereas data cannot be backed up to the Cloud until servers restart, and only players already in the game can continue to play; Data access servers have the same functionality and their system clocks are relatively synchronized, so if one server is down, any other servers can replace it.

Timestamp table failure: We utilize the primary/secondary model and the synchronous replication mechanism to maintain the reliability of timestamp tables. In the case of all replicas failure, we have to apply the original feature of Cassandra to obtain the up-to-date data. In other words, M-

N+1 replicas need to be compared. In this way, we can rebuild timestamp tables as well.

## 7. OPTIMIZATION AND FUTURE WORK

When a data access server updates state data in the Cloud, it propagates a snapshot of state data to multiple replicas. Note that state data has hundreds of attributes, so the transmission of a large volume of state data may block the network. Therefore, we proposed two optimization strategies in our previous work [4]: if only some less important attributes of the state (e.g., the position or orientation of a character) are modified, the backup can be skipped; Only the timestamp, ID, and the modified values are sent as messages to the Cloud. However, in order to achieve the second optimization strategy, our proposed data access approach, data replication strategies, and concurrency control mechanism have to be changed. For example, even during the game, updated values must be accepted by all replicas, so that the subsequent read request does not need to compare M-N+1 replicas. We will detail the adjustment program in our future work.

It is noteworthy that a data access server stores a timestamp repeatedly into the timestamp table, which increases the workload. A possible optimization program is as follows: If a batch write is successful, data access server caches the timestamp (TS) of this write request. Accordingly, in the timestamp table, we add a new column to each row to maintain a pointer. If a row is active (the value of LS is "login"), the pointer refers to the memory location of TS; if not, it refers to its own LMT. When a row becomes inactive, it uses TS to replace its LMT. In this way, the workload of a timestamp table will reduce significantly. However, LMT and Version ID of state data may be inconsistent due to the failure of the Cloud storage system or the data access server.

## 8. CONCLUSIONS

Our Cloud-based architecture of MMORPGs can cope with data management requirements regarding availability and scalability successfully, while supporting data consistency becomes an open issue. In this paper, we detailed our timestamp-based solution in theory, which will guide the implementation work in the future. We analyzed the data consistency requirements of each data set from the storage system's perspective, and studied methods of Cassandra to guarantee tunable consistency. We found that Cassandra cannot ensure read-your-writes consistency for state data and timed consistency for log data efficiently. Hence, we proposed a timestamp-based solution to improve it, and explained our idea for concurrency control, data replication strategies, and fault handling in detail. In our future work, we will implement our proposals and the optimization strategies.

## 9. ACKNOWLEDGEMENTS

Thanks to Eike Schallehn for his comments.

## 10. REFERENCES

[1] Apache. Cassandra, January 2013. `http://cassandra.apache.org/`.

[2] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Lt'eon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *Conference on Innovative Data Systems Research(CIDR)*, pages 223–234, Asilomar, California, USA, 2011.

[3] S. Das, D. Agrawal, and A. E. Abbadi. G-store: a scalable data store for transactional multi key access in the cloud. In *Symposium on Cloud Computing(SoCC)*, pages 163–174, Indianapolis, Indiana, USA, 2010.

[4] Z. Diao and E. Schallehn. Cloud Data Management for Online Games : Potentials and Open Issues. In *Data Management in the Cloud (DMC)*, Magdeburg, Germany, 2013. Accepted for publication.

[5] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM Special Interest Group on Algorithms and Computation Theory (SIGACT)*, 33(2):51–59, 2002.

[6] F. Gropengieß er, S. Baumann, and K.-U. Sattler. Cloudy transactions cooperative xml authoring on amazon s3. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 307–326, Kaiserslautern, Germany, 2011.

[7] A. Lakshman. Cassandra - A Decentralized Structured Storage System. *Operating Systems Review*, 44(2):35–40, 2010.

[8] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[9] F. W. Li, L. W. Li, and R. W. Lau. Supporting continuous consistency in multiplayer online games. In *12. ACM Multimedia 2004*, pages 388–391, New York, New York, USA, 2004.

[10] H. Liu, M. Bowman, and F. Chang. Survey of state melding in virtual worlds. *ACM Computing Surveys*, 44(4):1–25, 2012.

[11] W. Palant, C. Griwodz, and P. l. Halvorsen. Consistency requirements in multiplayer online games. In *Proceedings of the 5th Workshop on Network and System Support for Games, NETGAMES 2006*, page 51, Singapore, 2006.

[12] F. J. Torres-Rojas, M. Ahamad, and M. Raynal. Timed consistency for shared distributed objects. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing - PODC '99*, pages 163–172, Atlanta, Georgia, USA, 1999.

[13] W. Vogels. Eventually consistent. *ACM Queue*, 6(6):14–19, 2008.

[14] Z. Wei, G. Pierre, and C.-H. Chi. Scalable Transactions for Web Applications in the Cloud. In *15th International Euro-Par Conference*, pages 442–453, Delft, The Netherlands, 2009.

[15] K. Zhang and B. Kemme. Transaction Models for Massively Multiplayer Online Games. In *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011)*, pages 31–40, Madrid, Spain, 2011.

[16] K. Zhang, B. Kemme, and A. Denault. Persistence in massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NETGAMES 2008*, pages 53–58, Worcester, Massachusetts, USA, 2008.

# Analysis of DDoS Detection Systems

Michael Singhof
Heinrich-Heine-Universität
Institut für Informatik
Universitätsstraße 1
40225 Düsseldorf, Deutschland
singhof@cs.uni-duesseldorf.de

## ABSTRACT

While there are plenty of papers describing algorithms for detecting distributed denial of service (DDoS) attacks, here an introduction to the considerations preceding such an implementation is given. Therefore, a brief history of and introduction to DDoS attacks is given, showing that these kind of attacks are nearly two decades old. It is also depicted that most algorithms used for the detection of DDoS attacks are outlier detection algorithms, such that intrusion detection can be seen as a part of the KDD research field.

It is then pointed out that no well known and up-to-date test cases for DDoS detection system are known. To overcome this problem in a way that allows to test algorithms as well as making results reproducible for others we advice using a simulator for DDoS attacks.

The challenge of detecting denial of service attacks in real time is addressed by presenting two recently published methods that try to solve the performance problem in very different ways. We compare both approaches and finally summarise the conclusions drawn from this, especially that methods concentrating on one network traffic parameter only are not able to detect all kinds of distributed denial of service attacks.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering, Information filtering*

## Keywords

DDoS, Intrusion Detection, KDD, Security

## 1. INTRODUCTION

Denial of service (DoS) attacks are attacks that have the goal of making a network service unusable for its legitimate users. This can be achieved in different ways, either by targeting specific weaknesses in that service or by brute force approaches. A particularly well-known and dangerous kind of DoS attack are distributed denial of service attacks. These kinds of attacks are more or less brute force bandwidth DoS attacks carried out by multiple attackers simultaneously.

In general, there are two ways to detect any kind of network attacks: Signature based approaches in which the intrusion detection software compares network input to known attacks and anomaly detection methods. Here, the software is either trained with examples for normal traffic or not previously trained at all. Obviously, the latter method is more variable since normal network traffic does not change as quickly as attack methods. The algorithms used in this field are, essentially, known KDD methods for outlier detection such as clustering algorithms, classification algorithms or novelty detection algorithms on time series. However, in contrast to many other related tasks such as credit card fraud detection, network attack detection is highly time critical since attacks have to be detected in near real time. This makes finding suitable methods especially hard because high precision is necessary, too, in order for an intrusion detection system to not cause more harm than being of help.

The main goal of this research project is to build a distributed denial of service detection system that can be used in today's networks and meets the demands formulated in the previous paragraph. In order to build such a system, many considerations have to be done. Some of these are presented in this work.

The remainder of this paper is structured as follows: In section 2 an introduction to distributed denial of service attacks and known countermeasures is given, section 3 points out known test datasets. In section 4 some already existing approaches are presented and finally section 5 concludes this work and gives insight in future work.

## 2. INTRODUCTION TO DDOS ATTACKS

Denial of service and distributed denial of service attacks are not a new threat in the internet. In [15] the first notable denial of service attack is dated to 1996 when the internet provider Panix was taken down for a week by a TCP SYN flood attack. The same article dates the first noteworthy distributed denial of service attack to the year 1997 when internet service providers in several countries as well as an IRC network were attacked by a teenager. Since then, many of the more elaborate attacks that worked well in the past, have been successfully defused.

Let us, as an example, examine the TCP SYN flood attack. A TCP connection is established by a three way hand-

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

shake. On getting a SYN request packet, in order to open a TCP connection, the addressed computer has to store some information on the incoming packet and then answers with a SYN ACK packet which is, on regularly opening a TCP connection, again replied by an ACK packet.

The idea of the SYN flood attack is to cause a memory overrun on the victim by sending many TCP SYN packets. As for every such packet the victim has to store information while the attacker just generates new packets and ignores the victim's answers. By this the whole available memory of the victim can be used up, thus disabling the victim to open legitimate connection to regular clients. As a countermeasure, in [7] SYN cookies were introduced. Here, instead of storing the information associated with the only half opened TCP connection in the local memory, that information is coded into the TCP sequence number. Since that number is returned by regular clients on sending the last packet of the already described three way handshake and initial sequence numbers can be arbitrarily chosen by each connection partner, no changes on the TCP implementation of the client side have to be made. Essentially, this reduces the SYN cookie attack to a mere bandwidth based attack.

The same applies to many other attack methods that have been successfully used in the past, such as the smurf attack [1] or the fraggle attack. Both of these attacks are so called reflector attacks that consist of sending an echo packet – ICMP echo in case of the smurf attack and UDP echo in case of the fraggle attack – to a network's broadcast address. The sender's address in this packet has to be forged so that the packet occurs to be sent by the victim of the attack, so that all replies caused by the echo packet are routed to the victim.

Thus, it seems like nowadays most denial of service attacks are distributed denial of service attack trying to exhaust the victims bandwidth. Examples for this are the attacks on Estonian government and business computers in 2007 [12].

As already mentioned, distributed denial of service attacks are denial of service attacks with several participating attackers. The number of participating computers can differ largely, ranging from just a few machines to several thousands. Also, in most cases, the owners of these computers are not aware that they are part of an attack. This lies in the nature of most DDoS attacks which consist of three steps:

1. Building or reusing a malware that is able to receive commands from the main attacker ("master") and to carry out the attack. A popular DDoS framework is Stacheldraht [9].

2. Distribute the software created in step one to create a botnet. This step can essentially be carried out in every known method of distributing malware, for example by forged mail attachments or by adding it to software like pirate copies.

3. Launch the attack by giving the infected computers the command.

This procedure – from the point of view of the main attacker – has the advantage of not having to maintain a direct connection to the victim. This makes it very hard to track that person. It is notable though, that during attacks originating to Anonymous in the years 2010 and 2012 Low Orbit Ion Cannon [6] was used. This is originally a tool for stress testing that allows users, among other functions, to voluntary join a botnet in order to carry out an attack. Since the tool is mainly for testing purposes, the queries are not masqueraded so that it is easy to identify the participating persons. Again, however, the initiator of the attack does not necessarily have to have direct contact to the victim and thus remains unknown.

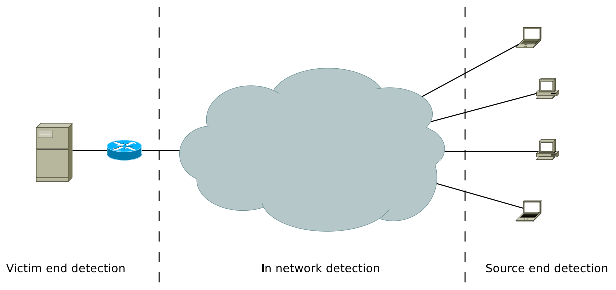


**Figure 1: Detection locations for DDoS attacks.**

A great diversity of approaches to solve the problem of detecting DDoS attacks exists. Note again, that this work focuses on anomaly detection methods only. This describes methods, that essentially make use of outlier detection methods to distinguish normal traffic and attack traffic. In a field with as many publications as intrusion detection and even, more specialised, DDoS detection, it is not surprising, that many different approaches are used, most of which are common in other knowledge discovery research fields as well.

As can be seen in Figure 1 this research part, again, can be divided in three major categories, namely distributed detection or in network detection, source end detection and end point or victim end detection.

By distributed detection approaches we denote all approaches that use more than one node in order to monitor the network traffic. This kind of solution is mostly aimed to be used by internet providers and sometimes cooperation between more than one or even all ISPs is expected. The main idea of almost all of these systems is to monitor the network traffic inside the backbone network. Monitors are mostly expected to be backbone routers, that communicate the results of their monitoring either to a central instance or among each other. These systems allow an early detection of suspicious network traffic so that an attack can be detected and disabled – by dropping the suspicious network packets – before it reaches the server the attack is aimed at. However, despite these methods being very mighty in theory, they suffer the main disadvantage of not being able to be employed without the help of one or more ISPs. Currently, this makes these approaches impractical for end users since, to the knowledge of the author, at this moment no ISP uses such an approach.

Source end detection describes approaches that monitor outgoing attack streams. Of course, such methods can only be successful if the owner of an attacking computer is not aware of his computer participating in that attack. A widely used deployment of such solutions is necessary for them to have an effect. If this happens, however, these methods have the chance to not only detect distributed denial of service attacks but also to prevent them by stopping the attacking traffic flows. However, in our opinion, the necessity of wide deployment makes a successful usage of this methods – at

| Packet type | No of packets | Percentage |
|---|---|---|
| IP | 65612516 | 100 |
| TCP | 65295894 | 99.5174 |
| UDP | 77 | 0.0001 |
| ICMP | 316545 | 0.4824 |

| Protocol | Incoming Traffic | Outgoing Traffic |
|---|---|---|
| IP | 24363685 | 41248831 |
| TCP | 24204679 | 41091215 |
| UDP | 77 | 0 |
| ICMP | 158929 | 157616 |

**Table 1: Distribution of web traffic on protocol types and incoming and outgoing traffic at the university's web server.**

least in the near future – difficult.

In contrast to the approaches described above, end point detection describes those methods that rely on one host only. In general, this host can be either the same server other applications are running on or a dedicated firewall in the case of small networks. Clearly, these approaches suffer one disadvantage: Attacks cannot be detected before the attack packets arrive at their destination, as only those packets can be inspected. On the other hand end point based methods allow individual deployment and can therefore be used nowadays. Due to this fact, our work focuses on end point approaches.

## 3. TEST TRACES OF DISTRIBUTED DENIAL OF SERVICE ATTACKS

Today, the testing of DDoS detection methods unfortunately is not easy, as not many recordings of real or simulated DDoS attacks exist or, at least, are not publicly available. The best known test trace is the KDD Cup 99 data set [3]. A detailed description of this data set is given in [18]. Other known datasets are the 1998 DARPA intrusion detection evaluation data set that has been described in [14] as well as the 1999 DARPA intrusion detection evaluation data set examined in [13].

In terms of the internet, with an age of 14 to 15 years, these data sets are rather old and therefore cannot reflect today's traffic volume and behaviour in a realistic fashion. Since testing with real distributed denial of service attacks is rather difficult both on technical as well as legal level, we suggest the usage of a DDoS simulator. In order to get a feeling for today's web traffic, we recorded a trace at the main web server of Heinrich-Heine-Universität. Tracing started on 17th September 2012 at eight o'clock local time and lasted until eight o'clock the next day.

This trace consists of 65612516 packets of IP traffic with 31841 unique communication partners contacting the web server. As can be seen in Table 1 almost all of these packets are TCP traffic. This is not surprising as the HTTP protocol uses the TCP protocol and web page requests are HTTP messages.

About one third of the TCP traffic is incoming traffic. This, too, is no surprise as most clients send small request messages and, in return, get web pages that often include images or other larger data and thus consist of more than one package. It can also be seen, clearly, that all of the

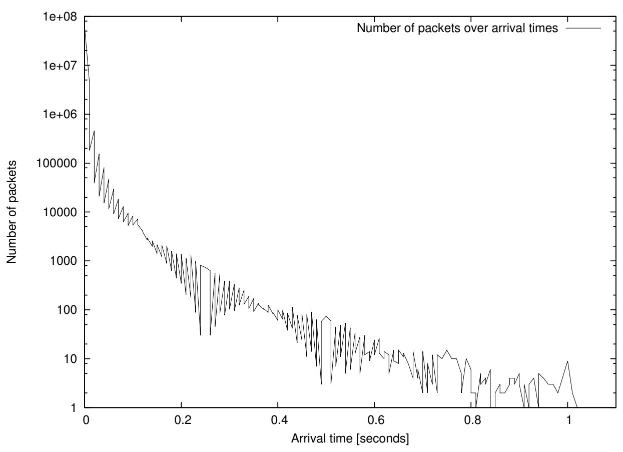


**Figure 2: Arrival times for the university's webserver trace.**

UDP packets seem to be unwanted packets as none of these packets is replied. The low overall number of these packets is an indicator for this fact, too. With ICMP traffic, incoming and outgoing packet numbers are nearly the same which lies in the nature of this message protocol.

In order to overcome the problems with old traces, based on the characteristics of the web trace, as a next step we implement a simulator for distributed denial of service attacks. As the results in [20] show, the network simulators OMNeT++ [19], ns-3 [10] and JiST [5] are, in terms of speed and memory usage, more or less equal. To not let the simulation become either too slow or too inaccurate, it is intended to simulate a nearer neighbourhood of the victim server very accurately. With greater distance to the victim, it is planned to simulate in less detail. In this context, the distance between two network nodes is given by the number of hops between the nodes.

Simulation results then will be compared with the aforementioned network trace to ensure its realistic behaviour. After the simulation of normal network traffic resembles the real traffic at the victim server close enough, we will proceed by implementing distributed denial of service attacks in the simulator environment. With this simulator it will then, hopefully, be possible to test existing and new distributed denial of service detection approaches in greater detail as has been possible in the past.

## 4. EXISTING APPROACHES

Many approaches to the detection of distributed denial of service attacks already exist. As has been previously pointed out in section 1, in contrast to many other outlier and novelty detection applications in the KDD field, the detection of DDoS attacks is extremely time critical, hence near real time detection is necessary.

Intuitively, the less parameters are observed by an approach, the faster it should work. Therefore, first, we take a look at a recently issued method that relies on one parameter only.

### 4.1 Arrival Time Based DDoS Detection

In [17] the authors propose an approach that bases on irregularities in the inter packet arrival times. By this term

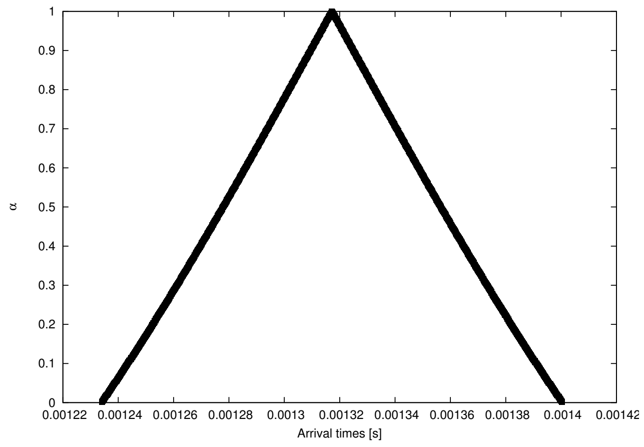


**Figure 3: The fuzzy mean estimator constructed according to [17].**

the authors describe the time that elapses between two subsequent packets.

The main idea of this work is based on [8] where non-asymptotic fuzzy estimators are used to estimate variable costs. Here, this idea is used to estimate the mean arrival time $\bar{x}$ of normal traffic packets. Then, the mean arrival time of the current traffic – denoted by $t_c$ – is estimated, too, and compared to the overall value. If $t_c > \bar{x}$, the traffic is considered as normal traffic and if $t_c < \bar{x}$ a DDoS attack is assumed to be happening. We suppose here, that for a value of $t_c = \bar{x}$ no attack is happening, although this case is not considered in the original paper.

To get a general feeling for the arrival times, we computed them for our trace. The result is shown in Figure 2. Note, that the $y$-axis is scaled logarithmic as values for arrival times larger than 0.1 seconds could not been distinguished from zero on a linear $y$-axis. It can be seen here, that most arrival times are very close to zero. It is also noteworthy that, due to the limited precision of libpcap [2], the most common arrival interval is zero.

Computing the fuzzy mean estimator for packet arrival times yields the graph presented in Figure 3 and $\bar{x} \approx 0.00132$. Note, that since the choice of parameter $\beta \in [0, 1)$ is not specified in [17], we here chose $\beta = \frac{1}{2}$. We will see, however, that, as far as our understanding of the proposed method goes, this parameter has no further influence.

To compute the $\alpha$-cuts of a fuzzy number, one has to compute

$$^{\alpha}M = \left[\bar{x} - z_{g(\alpha)}\frac{\sigma}{\sqrt{n}}, \bar{x} + z_{g(\alpha)}\frac{\sigma}{\sqrt{n}}\right]$$

where $\bar{x}$ is the mean value – i.e. exactly the value that is going to be estimated – and $\sigma$ is presumably the arrival times' deviation. Also

$$g(\alpha) = \left(\frac{1}{2} - \frac{\beta}{2}\right)\alpha + \frac{\beta}{2}$$

and

$$z_{g(\alpha)} = \Phi^{-1}(1 - g(\alpha)).$$

Note, that $^{\alpha}M$ is the $(1 - \alpha)(1 - \beta)$ confidence interval for $\mu$, the real mean value of packet arrival times.

Now, since we are solely interested in the estimation of $\bar{x}$, only $^{1}M$ is needed, which is computed to be $[\bar{x}, \bar{x}]$ since

$$g(1) = \left(\frac{1}{2} - \frac{\beta}{2}\right)1 + \frac{\beta}{2} = \frac{1}{2}(1 - \beta + \beta) = \frac{1}{2}$$

and

$$z_{g(1)} = \Phi^{-1}(1 - g(1)) = \Phi^{-1}(\frac{1}{2}) = 0.$$

During traffic monitoring, for a given time interval, the current traffic arrival times $t_c$ are computed by estimating

$$[t_c]_{\alpha} = \left[\ln\left(\frac{1}{1-p}\right)\frac{1}{r_{\alpha}}, \ln\left(\frac{1}{1-p}\right)\frac{1}{l_{\alpha}}\right]$$

where $p$ is some given but again not specified probability and $[l_{\alpha}, r_{\alpha}]$ are the $\alpha$-cuts for $E(T) = \bar{t}$. As described above, the only value that is of further use is $t_c$, the only value in the interval of $[t_c]_1$. Since $[E(T)]_1 = [\bar{t}]_1 = [\bar{t}, \bar{t}]$ it follows that

$$[t_c]_1 = \left[\ln\left(\frac{1}{1-p}\right)\frac{1}{\bar{t}}, \ln\left(\frac{1}{1-p}\right)\frac{1}{\bar{t}}\right]$$

and thus

$$t_c = \ln\left(\frac{1}{1-p}\right)\frac{1}{\bar{t}} = \frac{1}{\bar{t}}\left(\ln(1) - \ln(1-p)\right).$$

As $\ln(1) = 0$ this can be further simplified to

$$t_c = -\frac{\ln(1-p)}{\bar{t}} \in [0, \infty)$$

with $p \in [0, 1)$.

By this we are able to determine a value for $p$ by choosing the smallest $p$ where $t_c \geq \bar{x}$ for all intervals in our trace. An interval length of four seconds was chosen to ensure comparability with the results presented in [17].

During the interval with the highest traffic volume 53568 packets arrived resulting in an average arrival time of $\bar{t} \approx 7.4671 \cdot 10^{-5}$ seconds. Note here, that we did not maximise the number of packets for the interval but instead let the first interval begin at the first timestamp in our trace rounded down to full seconds and split the trace sequentially from there on.

Now, in order to compute $p$ one has to set

$$p = 1 - e^{-\bar{x}\bar{t}}$$

leading to $p \approx 9.8359 \cdot 10^{-8}$. As soon as this value of $p$ is learned, the approach is essentially a static comparison.

There are, however, other weaknesses to this approach as well: Since the only monitored value is the arrival time, a statement on values such as bandwidth usage cannot be made. Consider an attack where multiple corrupted computers try to download a large file from a server via a TCP connection. This behaviour will result in relatively large packets being sent from the server to the clients, resulting in larger arrival times as well. Still, the server's connection can be jammed by this traffic thus causing a denial of service.

By this, we draw the conclusion that a method relying on only one parameter – in this example arrival times – cannot detect all kinds of DDoS attacks. Thus, despite its low processing requirements, such an approach in our opinion is not suited for general DDoS detection even if it seems that it can detect packet flooding attacks with high precision as stated in the paper.

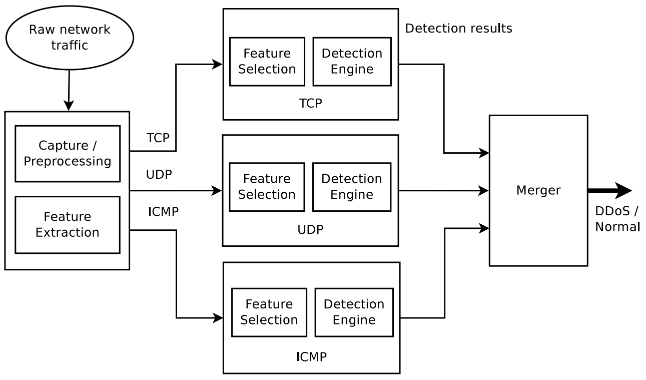


**Figure 4: Protocol specific DDoS detection architecture as proposed in [11].**

## 4.2 Protocol Type Specific DDoS Detection

In [11] another approach is presented: Instead of using the same methods on all types of packets, different procedures are used for different protocol types. This is due to the fact, that different protocols show different behaviour. Especially TCP traffic behaviour differs from UDP and ICMP traffic because of its flow control features. By this the authors try to minimise the feature set characterising distributed denial of service attacks for every protocol type, separately, such that computation time is minimised, too.

The proposed detection scheme is described as a four step approach, as shown in Figure 4. Here, the first step is the preprocessing where all features of the raw network traffic are extracted. Then packets are forwarded to the correspondent modules based on the packet's protocol type.

The next step is the protocol specific feature selection. Here, per protocol type, the most significant features are selected. This is done by using the linear correlation based feature selection (LCFS) algorithm that has been introduced in [4], which essentially ranks the given features by their correlation coefficients given by

$$corr(X,Y) := \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{x})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

for two random variables $X, Y$ with values $x_i, y_i, 1 \leq i \leq n$, respectively. A pseudo code version of LCFS is given in Algorithm 1. As can be seen there, the number of features in the reduced set must be given by the user. This number characterises the trade-off between precision of the detection and detection speed.

The third step is the classification of the instances in either normal traffic or DDoS traffic. The classification is trained on the reduced feature set generated in the previous step. The authors tested different well known classification techniques and established C4.5 [16] as the method working best in this case.

Finally, the outputs of the classifiers are given to the merger to be able to report warnings over one alarm generation interface instead of three. The authors mention that there is a check for false positives in the merger, too. However, there is no further information given on how this check works apart from the fact that it is relatively slow.

The presented experiments have been carried out on the aforementioned KDD Cup data set as well as on two self-made data sets for which the authors attacked a server within

**Algorithm 1** LCFS algorithm based on [11].

```
Require: the initial set of all features $I$,
    the class-outputs $y$,
    the desired number of features $n$
Ensure: the dimension reduced subset $F \subset I$

 1: for all $f_i \in I$ do
 2:     compute $corr(f_i, y)$
 3: end for
 4: $f := \max\{correlation(f_i, y) | f_i \in I\}$
 5: $F := \{f\}$
 6: $I := I \setminus \{f\}$
 7: while $|F| < n$ do
 8:     $f := \max\left\{ corr(f_i, y) - \frac{1}{|F|} \sum_{f_j \in F} corr(f_i, f_j) \middle| f_i \in I \right\}$
 9:     $F := F \cup \{f\}$
10:     $I := I \setminus \{f\}$
11: end while
12: return $F$
```

the university's campus. The presented results show that on all data sets the DDoS detection accuracy varies in the range of 99.683% to 99,986% if all of the traffic's attributes are used. When reduced to three or five attributes, accuracy stays high with DDoS detection of 99.481% to 99.972%. At the same time, the computation time shrinks by a factor of two leading to a per instance computation time of 0.0116ms (three attributes) on the KDD Cup data set and 0.0108ms (three attributes) and 0.0163ms (five attributes) on the self-recorded data sets of the authors.

Taking into account the 53568 packets in a four second interval we recorded, the computation time during this interval would be about ($53568 \cdot 0.0163$ms $\approx$) 0.87 seconds. However, there is no information about the machine that carried out the computations given in the paper such that this number appears to be rather meaningless. If we suppose a fast machine with no additional tasks, this computation time would be relatively high.

Nevertheless, the results presented in the paper at hand are promising enough to consider a future re-evaluation on a known machine with our recorded trace and simulated DDoS attacks.

## 5. CONCLUSION

We have seen that distributed denial of service attacks are, in comparison to the age of the internet itself, a relatively old threat. Against many of the more sophisticated attacks specialised counter measures exist, such as TCP SYN cookies in order to prevent the dangers of SYN flooding. Thus, most DDoS attacks nowadays are pure bandwidth or brute force attacks and attack detection should focus on this types of attacks, making outlier detection techniques the method of choice. Still, since many DDoS toolkits such as Stacheldraht allow for attacks like SYN flooding properties of this attacks can still indicate an ongoing attack.

Also, albeit much research on the field of DDoS detection has been done during the last two decades that lead to a nearly equally large number of possible solutions, in section 3 we have seen that one of the biggest problems is the unavailability of recent test traces or a simulator being able to produce such traces. With the best known test series having an age of fourteen years, today, the results presented in many of the research papers on this topic are difficult to compare and confirm.

Even if one can rate the suitability of certain approaches in respect to detect certain approaches, such as seen in section 4, a definite judgement of given methods is not easy. We therefore, before starting to implement an own approach to distributed denial of service detection, want to overcome this problem by implementing a DDoS simulator.

With the help of this tool, we will be subsequently able to compare existing approaches among each other and to our ideas in a fashion reproducible for others.

## 6. REFERENCES

[1] CERT CC. Smurf Attack. *http://www.cert.org/advisories/CA-1998-01.html.*

[2] The Homepage of Tcpdump and Libpcap. *http://www.tcpdump.org/.*

[3] KDD Cup Dataset. *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 1999.

[4] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani. Mutual Information-based Feature Selection for Intrusion Detection Systems. *Journal of Network and Computer Applications*, 34(4):1184–1199, 2011.

[5] R. Barr, Z. J. Haas, and R. van Renesse. JiST: An Efficient Approach to Simulation Using Virtual Machines. *Software: Practice and Experience*, 35(6):539–576, 2005.

[6] A. M. Batishchev. Low Orbit Ion Cannon. *http://sourceforge.net/projects/loic/.*

[7] D. Bernstein and E. Schenk. TCP SYN Cookies. *on-line journal, http://cr.yp.to/syncookies.html*, 1996.

[8] K. A. Chrysafis and B. K. Papadopoulos. Cost–volume–profit Analysis Under Uncertainty: A Model with Fuzzy Estimators Based on Confidence Intervals. *International Journal of Production Research*, 47(21):5977–5999, 2009.

[9] D. Dittrich. The 'Stacheldraht' Distributed Denial of Service Attack Tool. *http://staff.washington.edu/dittrich/misc/stacheldraht.analysis*, 1999.

[10] T. Henderson. ns-3 Overview. *http://www.nsnam.org/docs/ns-3-overview.pdf*, May 2011.

[11] H. J. Kashyap and D. Bhattacharyya. A DDoS Attack Detection Mechanism Based on Protocol Specific Traffic Features. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, pages 194–200. ACM, 2012.

[12] M. Lesk. The New Front Line: Estonia under Cyberassault. *Security & Privacy, IEEE*, 5(4):76–79, 2007.

[13] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer networks*, 34(4):579–595, 2000.

[14] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, et al. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, pages 12–26. IEEE, 2000.

[15] G. Loukas and G. Öke. Protection Against Denial of Service Attacks: A Survey. *The Computer Journal*, 53(7):1020–1037, 2010.

[16] J. R. Quinlan. *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

[17] S. N. Shiaeles, V. Katos, A. S. Karakos, and B. K. Papadopoulos. Real Time DDoS Detection Using Fuzzy Estimators. *Computers & Security*, 31(6):782–790, 2012.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani. A Detailed Analysis of the KDD CUP 99 Data Set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[19] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[20] E. Weingartner, H. vom Lehn, and K. Wehrle. A Performance Comparison of Recent Network Simulators. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–5, 2009.

# A Conceptual Model for the XML Schema Evolution

## Overview: Storing, Base-Model-Mapping and Visualization

Thomas Nösinger, Meike Klettke, Andreas Heuer
Database Research Group
University of Rostock, Germany
(tn, meike, ah)@informatik.uni-rostock.de

## ABSTRACT

In this article the conceptual model EMX (Entity Model for XML-Schema) for dealing with the evolution of XML Schema (XSD) is introduced. The model is a simplified representation of an XSD, which hides the complexity of XSD and offers a graphical presentation. For this purpose a unique mapping is necessary which is presented as well as further information about the visualization and the logical structure. A small example illustrates the relationships between an XSD and an EMX. Finally, the integration into a developed research prototype for dealing with the co-evolution of corresponding XML documents is presented.

## 1. INTRODUCTION

The eXtensible Markup Language (XML) [2] is one of the most popular formats for exchanging and storing structured and semi-structured information in heterogeneous environments. To assure that well-defined XML documents can be understood by every participant (e.g. user or application) it is necessary to introduce a document description, which contains information about allowed structures, constraints, data types and so on. XML Schema [4] is one commonly used standard for dealing with this problem. An XML document is called valid, if it fulfills all restrictions and conditions of an associated XML Schema.

XML Schema that have been used for years have to be modified from time to time. The main reason is that the requirements for exchanged information can change. To meet these requirements the schema has to be adapted, for example if additional elements are added into an existing content model, the data type of information changed or integrity constraints are introduced. All in all every possible structure of an XML Schema definition (XSD) can be changed. A question occurs: In which way can somebody make these adaptions without being coerced to understand and deal with the whole complexity of an XSD? One solution is the definition of a conceptual model for simplifying the base-model; in this paper we outline further details of our conceptual model called EMX (Entity Model for XML-Schema).

A further issue, not covered in this paper, but important in the overall context of exchanging information, is the validity of XML documents [5]. Modifications of XML Schema require adaptions of all XML documents that are valid against the former XML Schema (also known as co-evolution).

One unpractical way to handle this problem is to introduce different versions of an XML Schema, but in this case all versions have to be stored and every participant of the heterogeneous environment has to understand all different document descriptions. An alternative solution is the evolution of the XML Schema, so that just one document description exists at one time. The above mentioned validity problem of XML documents is not solved, but with the standardized description of the adaptions (e.g. a sequence of operations [8]) and by knowing a conceptual model inclusively the corresponding mapping to the base-model (e.g. XSD), it is possible to derive necessary XML document transformation steps automatically out of the adaptions [7]. The conceptual model is an essential prerequisite for the here not in detail but incidentally handled process of the evolution of XML Schema.

This paper is organized as follows. **Section 2** gives the necessary background of XML Schema and corresponding concepts. **Section 3** presents our conceptual model by first giving a formal definition (**3.1**), followed by the specification of the unique mapping between EMX and XSD (**3.2**) and the logical structure of the EMX (**3.3**). After introducing the conceptual model we present an example of an EMX in **section 4**. In **section 5** we describe the practical use of EMX in our prototype, which was developed for handle the co-evolution of XML Schema and XML documents. Finally in **section 6** we draw our conclusions.

## 2. TECHNICAL BACKGROUND

In this section we present a common notation used in the rest of the paper. At first we will shortly introduce the *abstract data model* (ADM) and *element information item* (EII) of XML Schema, before further details concerning different modeling styles are given.

The XML Schema *abstract data model* consists of different components or node types[1], basically these are: type definition components (simple and complex types), declaration components (elements and attributes), model group components, constraint components, group definition components

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

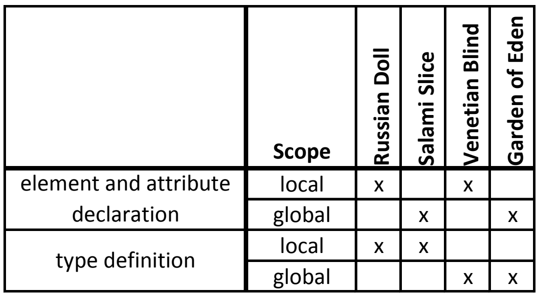
[1]An XML Schema can be visualized as a directed graph with different nodes (components); an edge realizes the hierarchy

and annotation components [3]. Additionally the *element information item* exists, an XML representation of these components. The *element information item* defines which content and attributes can be used in an XML Schema. Table 1 gives an overview about the most important components and their concrete representation. The <include>,

| ADM | Element Information Item |
|---|---|
| *declarations* | <element>, <attribute> |
| *group-definitions* | <attributeGroup> |
| *model-groups* | <all>, <choice>, <sequence>, <any>, <anyAttribute> |
| *type-definitions* | <simpleType>, <complexType> |
| N.N. | <include>, <import>, <redefine>, <overwrite> |
| *annotations* | <annotation> |
| *constraints* | <key>, <unique>, <keyref>, <assert>, <assertion> |
| N.N. | <schema> |

**Table 1: XML Schema Information Items**

<import>, <redefine> and <overwrite> are not explicitly given in the *abstract data model* (N.N. - Not Named), but they are important components for embedding externally defined XML Schema (esp. element declarations, attribute declarations and type definitions). In the rest of the paper, we will summarize them under the node type "module". The <schema> "is the document (root) element of any W3C XML Schema. It's both a container for all the declarations and definitions of the schema and a place holder for a number of default values expressed as attributes" [9]. Analyzing the possibilities of specifying declarations and definitions leads to four different modeling styles of XML Schema, these are: *Russian Doll*, *Salami Slice*, *Venetian Blind* and *Garden of Eden* [6]. These modeling styles influence mainly the reusability of element declarations or defined data types and also the flexibility of an XML Schema in general. Figure 1 summarizes the modeling styles with their scopes. The



|  | Scope | Russian Doll | Salami Slice | Venetian Blind | Garden of Eden |
|---|---|---|---|---|---|
| element and attribute declaration | local | x |  | x |  |
|  | global |  | x |  | x |
| type definition | local | x | x |  |  |
|  | global |  |  | x | x |

**Figure 1: XSD Modeling Styles according to [6]**

scope of element and attribute declarations as well as the scope of type definitions is global iff the corresponding node is specified as a child of the <schema> and can be referenced (by knowing e.g. the name and namespace). Locally specified nodes are in contrast not directly under <schema>, the re-usability is not given respectively not possible.

An XML Schema in the *Garden of Eden* style just contains global declarations and definitions. If the requirements against exchanged information change and the underlying schema has to be adapted then this modeling style is the most suitable. The advantage of the *Garden of Eden* style is that all components can be easily identified by knowing the *QNAME* (qualified name). Furthermore the position of components within an XML Schema is obvious. A qualified name is a colon separated string of the target namespace of the XML Schema followed by the name of the declaration or definition. The name of a declaration and definition is a string of the data type *NCNAME* (non-colonized name), a string without colons. The *Garden of Eden* style is the basic modeling style which is considered in this paper, a transformation between different styles is possible.[2]

## 3. CONCEPTUAL MODEL

In [7] the three layer architecture for dealing with XML Schema adaptions (i.e. the XML Schema evolution) was introduced and the correlations between them were mentioned. An overview is illustrated in figure 2. The first

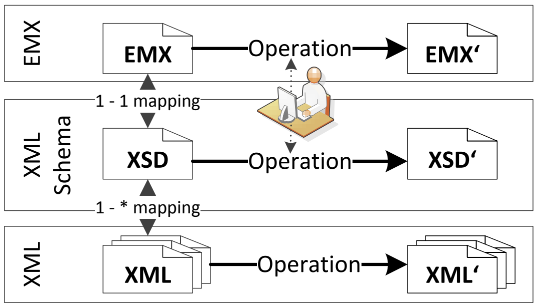


**Figure 2: Three Layer Architecture**

layer is our conceptual model EMX (Entity Model for XML-Schema), a simplified representation of the second layer. This layer is the XML Schema (XSD), where a unique mapping between these layers exists. The mapping is one of the main aspects of this paper (see section 3.2). The third layer are XML documents or instances, an ambiguous mapping between XSD and XML documents exists. It is ambiguous because of the optionality of structures (e.g. minOccurs = '0'; use = 'optional') or content types (e.g. <choice>). The third layer and the mapping between layer two and three, as well as the operations for transforming the different layers are not covered in this paper (parts were published in [7]).

### 3.1 Formal Definition

The conceptual model EMX is a triplet of nodes ($N_M$), directed edges between nodes ($E_M$) and features ($F_M$).

$$EMX = (N_M, E_M, F_M) \quad (1)$$

Nodes are separated in simple types (*st*), complex types (*ct*), *elements*, *attribute-groups*, *groups* (e.g. content model), *annotations*, *constraints* and *modules* (i.e. externally managed XML Schemas). Every node has under consideration of the *element information item* of a corresponding XSD different attributes, e.g. an *element* node has a name, occurrence values, type information, etc. One of the most important

[2]A student thesis to address the issue of converting different modeling styles into each other is in progress at our professorship; this is not covered in this paper.

attributes of every node is the *EID* (EMX ID), a unique identification value for referencing and localization of every node; an *EID* is one-to-one in every EMX. The directed edges are defined between nodes by using the *EID*s, i.e. every edge is a pair of *EID* values from a source to a target. The direction defines the include property, which was specified under consideration of the possibilities of an XML Schema. For example if a *model-group* of the *abstract data model* (i.e. an EMX *group* with "EID = 1") contains different *elements* (e.g. EID = {2,3}), then two edges exist: (1,2) and (1,3). In section 3.3 further details about allowed edges are specified (see also figure 5). The additional features allow the user-specific setting of the overall process of co-evolution. It is not only possible to specify default values but also to configure the general behaviour of operations (e.g. only capacity-preserving operations are allowed). Furthermore all XML Schema properties of the *element information item* <schema> are included in the additional features. The additional features are not covered in this paper.

## 3.2 Mapping between XSD and EMX

An overview about the components of an XSD has been given in table 1. In the following section the unique mapping between these XSD components and the EMX nodes introduced in section 3.1 is specified. Table 2 summarizes the mapping. For every *element information item* (EII) the

| EII | EMX Node | Visualization |
|---|---|---|
| <element> | *element* | Ⓔ |
| <attribute>, <attributeGroup> | *attribute-group* | @ |
| <all>, <choice>, <sequence><br><any><br><anyAttribute> | *group* | ▲<br>▲ Ⓦ<br>@ Ⓦ |
| <simpleType> | *st* | implicit and specifiable |
| <complexType> | *ct* | implicit and derived |
| <include>, <import>, <redefine>, <overwrite> | *module* | Ⓜ |
| <annotation> | *annotation* | # |
| <key>, <unique>, <keyref><br><assert><br><assertion> | *constraint* | Ⓚ<br>implicit in *ct*<br>restriction in *st* |
| <schema> | the EMX itself | |

**Table 2: Mapping and Visualization**

corresponding EMX node is given as well as the assigned visualization. For example an EMX node *group* represents the *abstract data model* (ADM) node *model-group* (see table 1). This ADM node is visualized through the EII content models <all>, <choice> and <sequence>, and the wildcards <any> and <anyAttribute>. In an EMX the visualization of a *group* is the blue "triangle with a G" in it. Furthermore if a *group* contains an element wildcard then this is visualized by adding a "blue W in a circle", a similar behaviour takes place if an attribute wildcard is given in an <attributeGroup>.

The *type-definitions* are not directly visualized in an EMX. Simple types for example can be specified and afterwards be referenced by *elements* or *attributes*[3] by using the *EID* of the corresponding EMX node. The complex type is also implicitly given, the type will be automatically derived from the structure of the EMX after finishing the modeling process. The XML Schema specification 1.1 has introduced different logical constraints, which are also integrated in the EMX. These are the EIIs <assert> (for constraints on complex types) and <assertion>. An <assertion> is under consideration of the specification a facet of a restricted simple type [4]. The last EII is <schema>, this "root" is an EMX itself. This is also the reason why further information or properties of an XML Schema are stored in the additional features as mentioned above.

## 3.3 Logical Structure

After introducing the conceptual model and specifying the mapping between an EMX and XSD, in the following section details about the logical structure (i.e. the storing model) are given. Also details about the valid edges of an EMX are illustrated. Figure 3 gives an overview about the different relations used as well as the relationships between them. The logical structure is the direct consequence of the used

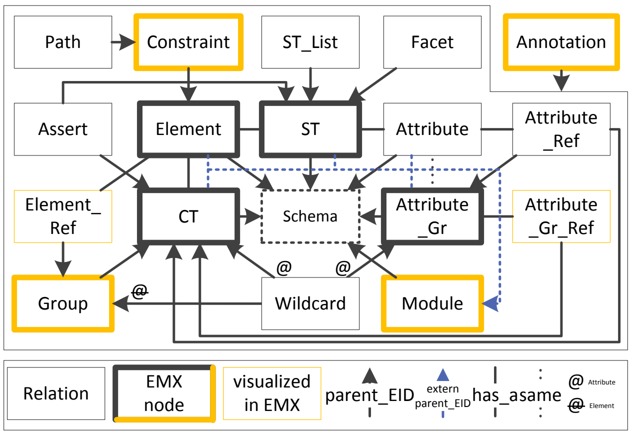


**Figure 3: Logical Structure of an EMX**

modeling style *Garden of Eden*, e.g. *elements* are either element declarations or element references. That's why this separation is also made in the EMX.

All in all there are 18 relations, which store the content of an XML Schema and form the base of an EMX. The different nodes reference each other by using the well known foreign key constraints of relational databases. This is expressed by using the directed "parent_EID" arrows, e.g. the EMX nodes ("rectangle with thick line") *element*, *st*, *ct*, *attribute-group* and *modules* reference the "Schema" itself. If declarations or definitions are externally defined then the "parent_EID" is the *EID* of the corresponding *module* ("blue arrow"). The "Schema" relation is an EMX respectively the root of an EMX as already mentioned above.

[3]The EII <attribute> and <attributeGroup> are the same in the EMX, an *attribute-group* is always a container

The "Annotation" relation can reference every other relation according to the XML Schema specification. Wildcards are realized as an element wildcard, which belongs to a "Group" (i.e. EII <any>), or they can be attribute wildcards which belongs to a "CT" or "Attribute_Gr" (i.e. EII <anyAttribute>). Every "Element" relation (i.e. element declaration) has either a simple type or a complex type, and every "Element_Ref" relation has an element declaration. Attributes and attribute-groups are the same in an EMX, as mentioned above.

Moreover figure 3 illustrates the distinction between visualized ("yellow border") and not visualized relations. Under consideration of table 2 six relations are direct visible in an EMX: constraints, annotations, modules, groups and because of the *Garden of Eden* style element references and attribute-group references. Table 3 summarizes which relation of figure 3 belongs to which EMX node of table 2.

| **EMX Node** | **Relation** |
|---|---|
| *element* | Element, Element_Ref |
| *attribute-group* | Attribute, Atttribute_Ref, Attribute_Gr, Attribute_Gr_Ref |
| *group* | Group, Wildcard |
| *st* | ST, ST_List, Facet |
| *ct* | CT |
| *annotation* | Annotation |
| *constraint* | Contraint, Path, Assert |
| *module* | Module |

**Table 3: EMX Nodes with Logical Structure**

The EMX node *st* (i.e. simple type) has three relations. These are the relation "ST" for the most simple types, the relation "ST_List" for set free storing of simple union types and the relation "Facet" for storing facets of a simple restriction type. *Constraints* are realized through the relation "Path" for storing all used XPath statements for the *element information items* (EII) <key>, <unique> and <keyref> and the relation "Constraint" for general properties e.g. name, XML Schema id, visualization information, etc. Furthermore the relation "Assert" is used for storing logical constraints against complex types (i.e. EII <assert>) and simple types (i.e. EII <assertion>). Figure 4 illustrates the

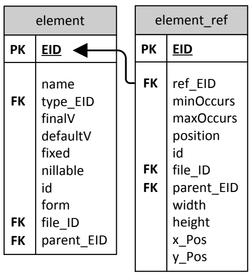
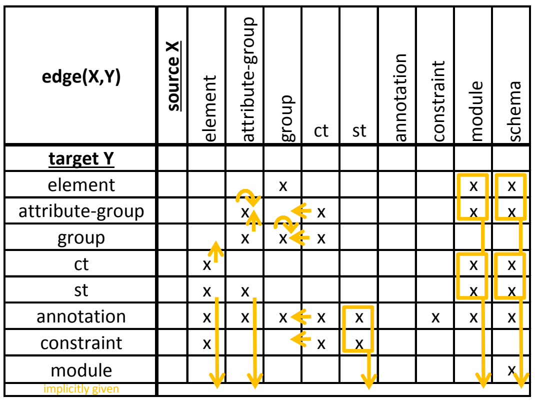


**Figure 4: Relations of EMX Node *element***

stored information concerning the EMX node *element* respectively the relations "Element" and "Element_Ref". Both relations have in common, that every tuple is identified by using the primary key *EID*. The *EID* is one-to-one in every EMX as mentioned above. The other attributes are specified under consideration of the XML Schema specification [4], e.g. an element declaration needs a "name" and a type ("type_EID" as a foreign key) as well as other optional values like the final ("finalV"), default ("defaultV"), "fixed", "nillable", XML Schema "id" or "form" value. Other EMX specific attributes are also given, e.g. the "file_ID" and the "parent_EID" (see figure 3). The element references have a "ref_EID", which is a foreign key to a given element declaration. Moreover attributes of the occurrence ("minOccurs", "maxOccurs"), the "position" in a content model and the XML Schema "id" are stored. Element references are visualized in an EMX. That's why some values about the position in an EMX are stored, i.e. the coordinates ("x_Pos", "y_Pos") and the "width" and "height" of an EMX node. The same position attributes are given in every other visualized EMX node.

The edges of the formal definition of an EMX can be derived by knowing the logical structure and the visualization of an EMX. Figure 5 illustrates the allowed edges of EMX nodes. An edge is always a pair of *EID*s, from a source



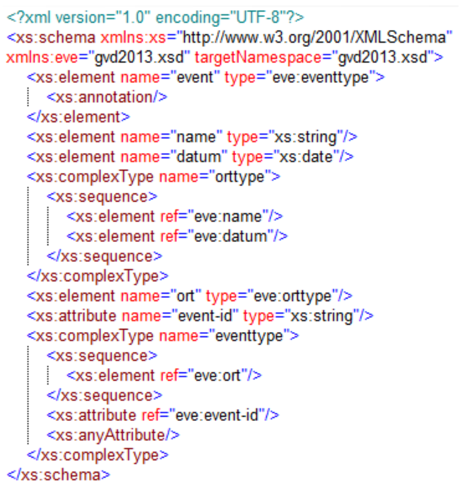
| **edge(X,Y)** | **source X** | element | attribute-group | group | ct | st | annotation | constraint | module | schema |
|---|---|---|---|---|---|---|---|---|---|---|
| **target Y** | | | | | | | | | | |
| element | | | | x | | | | | x | x |
| attribute-group | | | x | | x | | | | x | x |
| group | | | x | x | x | | | | | |
| ct | | x | | | | | | | x | x |
| st | | x | x | | | | | | x | x |
| annotation | | x | x | x | x | x | | x | x | x |
| constraint | | x | | | x | x | | | | |
| module | | | | | | | | | | x |
| implicitly given | | | | | | | | | | |

**Figure 5: Allowed Edges of EMX Nodes**

("X") to a target ("Y"). For example it is possible to add an edge outgoing from an *element* node to an *annotation*, *constraint*, *st* or *ct*. A "black cross" in the figure defines a possible edge. If an EMX is visualized then not all EMX nodes are explicitly given, e.g. the *type-definitions* of the *abstract data model* (i.e. EMX nodes *st*, *ct*; see table 2). In this case the corresponding "black cross" has to be moved along the given "yellow arrow", i.e. an edge in an EMX between a *ct* (source) and an *attribute-group* (target) is valid. If this EMX is visualized, then the *attribute-group* is shown as a child of the *group* which belongs to above mentioned *ct*. Some information are just "implicitly given" in a visualization of an EMX (e.g. simple types). A "yellow arrow" which starts and ends in the same field is a hint for an union of different nodes into one node, e.g. if a *group* contains a wildcard then in the visualization only the *group* node is visible (extended with the "blue W"; see table 2).

## 4. EXAMPLE

In section 3 the conceptual model EMX was introduced. In the following section an example is given. Figure 6 illustrates an XML Schema in the *Garden of Eden* modeling style. An event is specified, which contains a place ("ort") and an id ("event-id"). Furthermore the integration of other



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:eve="gvd2013.xsd" targetNamespace="gvd2013.xsd">
  <xs:element name="event" type="eve:eventtype">
    <xs:annotation/>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="datum" type="xs:date"/>
  <xs:complexType name="orttype">
    <xs:sequence>
      <xs:element ref="eve:name"/>
      <xs:element ref="eve:datum"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ort" type="eve:orttype"/>
  <xs:attribute name="event-id" type="xs:string"/>
  <xs:complexType name="eventtype">
    <xs:sequence>
      <xs:element ref="eve:ort"/>
    </xs:sequence>
    <xs:attribute ref="eve:event-id"/>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:schema>
```

**Figure 6: XML Schema in *Garden of Eden* Style**

attributes is possible, because of an attribute wildcard in the respective complex type. The place is a sequence of a name and a date (”datum”).

All type definitions (*NCNAME*s: ”orttype”, ”eventtype”) and declarations (*NCNAME*s: ”event”, ”name”, ”datum”, ”ort” and the attribute ”event-id”) are globally specified. The target namespace is ”eve”, so the *QNAME* of e.g. the complex type definition ”orttype” is ”eve:orttype”. By using the *QNAME* every above mentioned definition and declaration can be referenced, so the re-usability of all components is given. Furthermore an attribute wildcard is also specified, i.e. the complex type ”eventtype” contains apart from the content model sequence and the attribute reference ”eve:event-id” the *element information item* <anyAttribute>.

Figure 7 is the corresponding EMX of the above specified XML Schema. The representation is an obvious simplifica-

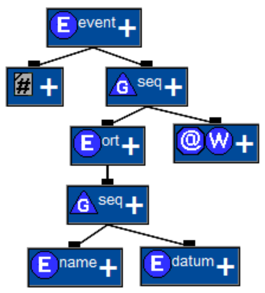


**Figure 7: EMX to XSD of Figure 6**

tion, it just contains eight well arranged EMX nodes. These are the *elements* ”event”, ”ort”, ”name” and ”datum”, an *annotation* as a child of ”event”, the *groups* as a child under ”event” and ”ort”, as well as an *attribute-group* with wildcard. The simple types of the element references ”name” and ”datum” are implicitly given and not visualized. The complex types can be derived by identifying the elements which have no specified simple type but groups as a child (i.e. ”event” and ”ort”).

The edges are under consideration of figure 5 pairs of not visualized, internally defined *EID*s. The source is the side of the connection without ”black rectangle”, the target is the other side. For example the given *annotation* is a child of the *element* ”event” and not the other way round; an element can never be a child of an *annotation*, neither in the XML Schema specification nor in the EMX.

The logical structure of the EMX of figure 7 is illustrated in figure 8. The relations of the EMX nodes are given as well

**Schema**

| EID | xmlns_xs | targetName | TNPrefix |
|---|---|---|---|
| 1 | http://www.w3.org/2001/XMlSchema | gvd2013.xsd | eve |

**Element**

| EID | name | type_EID | parent_EID |
|---|---|---|---|
| 2 | event | 14 | 1 |
| 3 | name | 11 | 1 |
| 4 | datum | 12 | 1 |
| 5 | ort | 13 | 1 |

**Annotation**

| EID | parent_EID | x_Pos | y_Pos |
|---|---|---|---|
| 10 | 2 | 50 | 100 |

**Wildcard**

| EID | parent_EID |
|---|---|
| 17 | 14 |

**Element_Ref**

| EID | ref_EID | minOccurs | maxOccurs | parent_EID | x_Pos | y_Pos | |
|---|---|---|---|---|---|---|---|
| 6 | 2 | 1 | 1 | 1 | 75 | 75 | event |
| 7 | 3 | 1 | 1 | 16 | 60 | 175 | name |
| 8 | 4 | 1 | 1 | 16 | 150 | 175 | datum |
| 9 | 5 | 1 | 1 | 15 | 100 | 125 | ort |

**ST**

| EID | name | mode | parent_EID |
|---|---|---|---|
| 11 | xs:string | built-in | 1 |
| 12 | xs:date | built-in | 1 |

**CT**

| EID | name | parent_EID |
|---|---|---|
| 13 | orttype | 1 |
| 14 | eventtype | 1 |

**Group**

| EID | mode | parent_EID | x_Pos | y_Pos | |
|---|---|---|---|---|---|
| 15 | sequence | 14 | 125 | 100 | eventsequence |
| 16 | sequence | 13 | 100 | 150 | ortsequence |

**Attribute**

| EID | name | parent_EID |
|---|---|---|
| 18 | event-id | 1 |

**Attribute_Ref**

| EID | ref_EID | parent_EID |
|---|---|---|
| 19 | 18 | 14 |

**Attribute_Gr**

| EID | parent_EID |
|---|---|
| 20 | 1 |

**Attribute_Gr_Ref**

| EID | ref_EID | parent_EID | x_Pos | y_Pos |
|---|---|---|---|---|
| 21 | 20 | 14 | 185 | 125 |

**Figure 8: Logical Structure of Figure 7**

as the attributes and corresponding values relevant for the example. Next to every tuple of the relations ”Element_Ref” and ”Group” small hints which tuples are defined are added (for increasing the readability). It is obvious that an *EID* has to be unique, this is a prerequisite for the logical structure. An *EID* is created automatically, a user of the EMX can neither influence nor manipulate it.

The element references contain information about the occurrence (”minOccurs”, ”maxOccurs”), which are not explicitly given in the XSD of figure 6. The XML Schema specification defines default values in such cases. If an element reference does not specify the occurrence values then the standard value ”1” is used; an element reference is obligatory. These default values are also added automatically.

The stored names of element declarations are *NCNAME*s, but by knowing the target namespace of the corresponding schema (i.e. ”eve”) the *QNAME* can be derived. The name of a type definition is also the *NCNAME*, but if e.g. a built-in type is specified then the name is the *QNAME* of the XML Schema specification (”xs:string”, ”xs:date”).

## 5. PRACTICAL USE OF EMX

The co-evolution of XML documents was already mentioned in section 1. At the University of Rostock a research prototype for dealing with this co-evolution was developed: CodeX (Conceptual design and evolution for XML Schema)

[5]. The idea behind it is simple and straightforward at the same time: Take an XML Schema, transform it to the specifically developed conceptual model (EMX - Entity Model for XML-Schema), change the simplified conceptual model instead of dealing with the whole complexity of XML Schema, collect these changing information (i.e. the user interaction with EMX) and use them to create automatically transformation steps for adapting the XML documents (by using XSLT - Extensible Stylesheet Language Transformations [1]). The mapping between EMX and XSD is unique, so it is possible to describe modifications not only on the EMX but also on the XSD. The transformation and logging language ELaX (Evolution Language for XML-Schema [8]) is used to unify the internally collected information as well as introduce an interface for dealing directly with XML Schema. Figure 9 illustrates the component model of CodeX, firstly published in [7] but now extended with the ELaX interface.

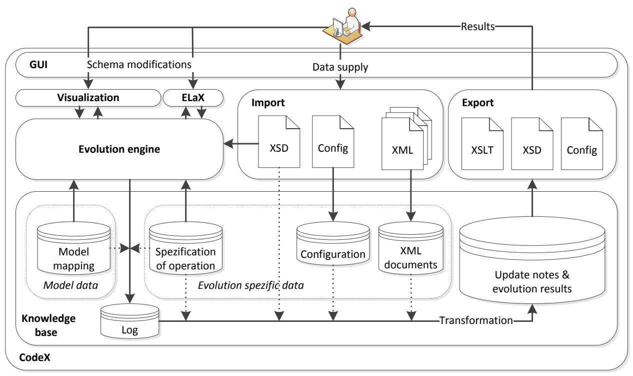


**Figure 9: Component Model of CodeX [5]**

The component model illustrates the different parts for dealing with the co-evolution. The main parts are an import and export component for collecting and providing data of e.g. a user (XML Schemas, configuration files, XML document collections, XSLT files), a knowledge base for storing information (model data, evolution specific data and co-evolution results) and especially the logged ELaX statements ("Log"). The mapping information between XSD and EMX of table 2 are specified in the "Model data" component.

Furthermore the CodeX prototype also provides a graphical user interface ("GUI"), a visualization component for the conceptual model and an evolution engine, in which the transformations are derived. The visualization component realizes the visualization of an EMX introduced in table 2. The ELaX interface for modifying imported XML Schemas communicates directly with the evolution engine.

## 6. CONCLUSION

Valid XML documents need e.g. an XML Schema, which restricts the possibilities and usage of declarations, definitions and structures in general. In a heterogeneous changing environment (e.g. an information exchange scenario), also "old" and longtime used XML Schema have to be modified to meet new requirements and to be up-to-date.

EMX (Entity Model for XML-Schema) as a conceptual model is a simplified representation of an XSD, which hides its complexity and offers a graphical presentation. A unique mapping exists between every in the *Garden of Eden* style modeled XSD and an EMX, so it is possible to representatively adapt or modify the conceptual model instead of the XML Schema.

This article presents the formal definition of an EMX, all in all there are different nodes, which are connected by directed edges. Thereby the *abstract data model* and *element information item* of the XML Schema specification were considered, also the allowed edges are specified according to the specification. In general the most important components of an XSD are represented in an EMX, e.g. elements, attributes, simple types, complex types, annotations, constrains, model groups and group definitions. Furthermore the logical structure is presented, which defines not only the underlying storing relations but also the relationships between them. The visualization of an EMX is also defined: outgoing from 18 relations in the logical structure, there are eight EMX nodes in the conceptual model, from which six are visualized.

Our conceptual model is an essential prerequisite for the prototype CodeX (Conceptual design and evolution for XML Schema) as well as for the above mentioned co-evolution. A remaining step is the finalization of the implementation in CodeX. After this work an evaluation of the usability of the conceptual model is planned. Nevertheless we are confident, that the usage is straightforward and the simplification of EMX in comparison to deal with the whole complexity of an XML Schema itself is huge.

## 7. REFERENCES

[1] XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/2007/REC-xslt20-20070123/`, January 2007. Online; accessed 26-March-2013.
[2] Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/2008/REC-xml-20081126/`, November 2008. Online; accessed 26-March-2013.
[3] XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition). `http://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/`, December 2010. Online; accessed 26-March-2013.
[4] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. `http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/`, April 2012. Online; accessed 26-March-2013.
[5] M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63, 2007.
[6] E. Maler. Schema design rules for ubl...and maybe for you. In *XML 2002 Proceedings by deepX*, 2002.
[7] T. Nösinger, M. Klettke, and A. Heuer. Evolution von XML-Schemata auf konzeptioneller Ebene - Übersicht: Der CodeX-Ansatz zur Lösung des Gültigkeitsproblems. In *Grundlagen von Datenbanken*, pages 29–34, 2012.
[8] T. Nösinger, M. Klettke, and A. Heuer. Automatisierte Modelladaptionen durch Evolution - (R)ELaX in the Garden of Eden. Technical Report CS-01-13, Institut für Informatik, Universität Rostock, Rostock, Germany, Jan. 2013. Published as technical report CS-01-13 under ISSN 0944-5900.
[9] E. van der Vlist. *XML Schema*. O'Reilly & Associates, Inc., 2002.

# Semantic Enrichment of Ontology Mappings: Detecting Relation Types and Complex Correspondences

Patrick Arnold*
Universität Leipzig
arnold@informatik.uni-leipzig.de

## ABSTRACT

While there are numerous tools for ontology matching, most approaches provide only little information about the true nature of the correspondences they discover, restricting themselves on the mere links between matching concepts. However, many disciplines such as ontology merging, ontology evolution or data transformation, require more-detailed information, such as the concrete relation type of matches or information about the cardinality of a correspondence (one-to-one or one-to-many). In this study we present a new approach where we denote additional semantic information to an initial ontology mapping carried out by a state-of-the-art matching tool. The enriched mapping contains the relation type (like equal, is-a, part-of) of the correspondences as well as complex correspondences. We present different linguistic, structural and background knowledge strategies that allow semi-automatic mapping enrichment, and according to our first internal tests we are already able to add valuable semantic information to an existing ontology mapping.

## Keywords

ontology matching, relation type detection, complex correspondences, semantic enrichment

## 1. INTRODUCTION

Ontology matching plays a key role in data integration and ontology management. With the ontologies getting increasingly larger and more complex, as in the medical or biological domain, efficient matching tools are an important prerequisite for ontology matching, merging and evolution. There are already various approaches and tools for ontology matching, which exploit most different techniques like lexicographic, linguistic or structural methods in order to identify the corresponding concepts between two ontologies [16], [2]. The determined correspondences build a so-called alignment or ontology mapping, with each correspondence being a tripe $(s, t, c)$, where $s$ is a concept in the source ontology, $t$ a concept in the target ontology and $c$ the confidence (similarity).

*

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

These tools are able to highly reduce the effort of manual ontology mapping, but most approaches solely focus on detecting the matching pairs between two ontologies, without giving any specific information about the true nature of these matches. Thus, a correspondence is commonly regarded an equivalence relation, which is correct for a correspondence like (zip code, postal code), but incorrect for correspondences like (car, vehicle) or (tree trunk, tree), where is-a resp. part-of would be the correct relation type. This restriction is an obvious shortcoming, because in many cases a mapping should also include further kinds of correspondences, such as is-a, part-of or related. Adding these information to a mapping is generally beneficial and has been shown to considerably improve ontology merging [13]. It provides more precise mappings and is also a crucial aspect in related areas, such as data transformation, entity resolution and linked data.

An example is given in Fig. 1, which depicts the basic idea of our approach. While we get a simple alignment as input, with the mere links between concepts (above picture), we return an enriched alignment with the relation type annotated to each correspondence (lower picture). As we will point out in the course of this study, we use different linguistic methods and background knowledge in order to find the relevant relation type. Besides this, we have to distinguish between simple concepts (as "Office Software") and complex concepts, which contain itemizations like "Monitors and Displays", and which need a special treatment for relation type detection.

Another issue of present ontology matchers is their restriction to (1:1)-correspondences, where exactly one source concept matches exactly one target concept. However, this can occasionally lead to inaccurate mappings, because there may occur complex correspondences where more than one source element corresponds to a target element or vice versa, as the two concepts *first name* and *last name* correspond to a concept *name*, leading to a (2:1)-correspondence. We will show in section 5 that distinguishing between one-to-one and one-to-many correspondences plays an important role in data transformation, and that we can exploit the results from the relation type detection to discover such complex matches in a set of (1:1)-matches to add further knowledge to a mapping.

In this study we present different strategies to assign the relation types to an existing mapping and demonstrate how

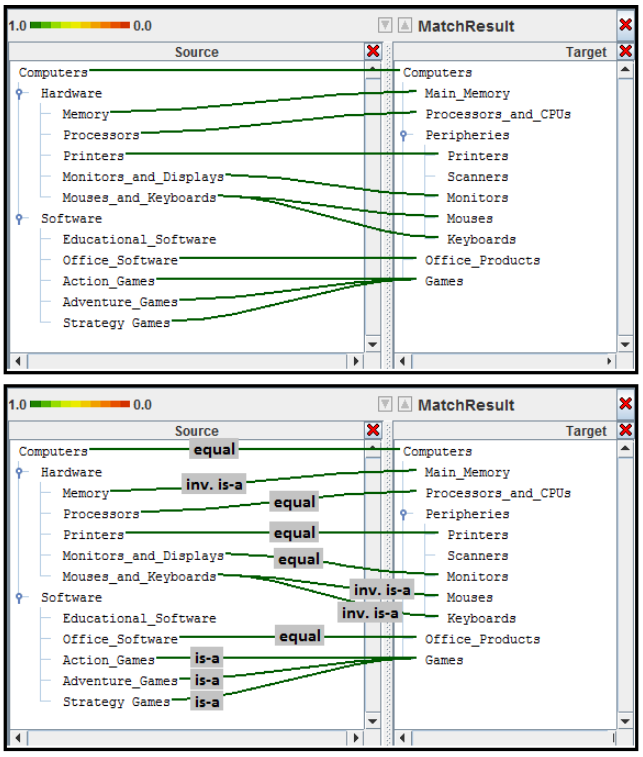


**Figure 1: Input (above) and output (below) of the Enrichment Engine**

complex correspondences can be discovered. Our approach, which we refer to as Enrichment Engine, takes an ontology mapping generated by a state-of-the-art matching tool as input and returns a more-expressive mapping with the relation type added to each correspondence and complex correspondences revealed. According to our first internal tests, we recognized that even simple strategies already add valuable information to an initial mapping and may be a notable gain for current ontology matching tools.

Our paper is structured as follows: We discuss related work in section 2 and present the architecture and basic procedure of our approach in section 3. In section 4 we present different strategies to determine the relation types in a mapping, while we discuss the problem of complex correspondence detection in section 5. We finally conclude in section 6.

## 2. RELATED WORK

Only a few tools and studies regard different kinds of correspondences or relationships for ontology matching. S-Match [6][7] is one of the first such tools for "semantic ontology matching". They distinguish between equivalence, subset (is-a), overlap and mismatch correspondences and try to provide a relationship for any pair of concepts of two ontologies by utilizing standard match techniques and background knowledge from WordNet. Unfortunately, the result mappings tend to become very voluminous with many correspondences per concept, while users are normally interested only in the most relevant ones.

Taxomap [11] is an alignment tool developed for the geographic domain. It regards the correspondence types equivalence, less/more-general (is-a / inverse is-a) and is-close ("related") and exploits linguistic techniques and background sources such as WordNet. The linguistic strategies seem rather simple; if a term appears as a part in another term, a more-general relation is assumed which is not always the case. For example, in Figure 1 the mentioned rule holds for the correspondence between *Games* and *Action_Games*, but not between *Monitors* and *Monitors_and_Displays*. In [14], the authors evaluated Taxomap for a mapping scenario with 162 correspondences and achieved a recall of 23 % and a precision of 89 %.

The LogMap tool [9] distinguishes between equivalence and so-called weak (subsumption / is-a) correspondences. It is based on Horn Logic, where first lexicographic and structural knowledge from the ontologies is accumulated to build an initial mapping and subsequently an iterative process is carried out to first enhance the mapping and then to verify the enhancement. This tool is the least precise one with regard to relation type detection, and in evaluations the relation types were not further regarded.

Several further studies deal with the identification of semantic correspondence types without providing a complete tool or framework. An approach utilizing current search engines is introduced in [10]. For two concepts $A$, $B$ they generate different search queries like "A, such as B" or "A, which is a B" and submit them to a search engine (e.g., Google). They then analyze the snippets of the search engine results, if any, to verify or reject the tested relationship. The approach in [15] uses the Swoogle search engine to detect correspondences and relationship types between concepts of many crawled ontologies. The approach supports equal, subset or mismatch relationships. [17] exploits reasoning and machine learning to determine the relation type of a correspondence, where several structural patterns between ontologies are used as training data.

Unlike relation type determination, the complex correspondence detection problem has hardly been discussed so far. It was once addressed in [5], coming to the conclusion that there is hardly any approach for complex correspondence detection because of the vast amount of required comparisons in contrast to (1:1)-matching, as well as the many possible operators needed for the mapping function. One key observation for efficient complex correspondence detection has been the need of large amounts of domain knowledge, but until today there is no available tool being able to semi-automatically detect complex matches.

One remarkable approach is iMAP [4], where complex matches between two schemas could be discovered and even several transformation functions calculated, as $RoomPrice = RoomPrice * (1 + TaxPrice)$. For this, iMAP first calculates (1:1)-matches and then runs an iterative process to gradually combine them to more-complex correspondences. To justify complex correspondences, instance data is analyzed and several heuristics are used. In [8] complex correspondences were also regarded for matching web query interfaces, mainly exploiting co-occurrences. However, in order to derive common co-occurrences, the approach requires a large amount of schemas as input, and thus does not appear appropriate for matching two or few schemas.

While the approaches presented in this section try to achieve both matching and semantic annotation in one step, thus often tending to neglect the latter part, we will demonstrate a two-step architecture in which we first perform a schema mapping and then concentrate straight on the enrichment of the mapping (semantic part). Additionally, we want to analyze several linguistic features to provide more qualitative mappings than obtained by the existing tools, and finally develop an independent system that is not restricted to schema and ontology matching, but will be differently exploitable in the wide field of date integration and data analysis.

## 3. ARCHITECTURE

As illustrated in Fig. 2 our approach uses a 2-step architecture in which we first calculate an ontology mapping (match result) using our state-of-the-art matching tool COMA 3.0 (step 1) [12] and then perform an enrichment on this mapping (step 2).

Our 2-step approach for semantic ontology matching offers different advantages. First of all, we reduce complexity compared to 1-step approaches that try to directly determine the correspondence type when comparing concepts in $O_1$ with concepts in $O_2$. For large ontologies, such a direct matching is already time-consuming and error-prone for standard matching. The proposed approaches for semantic matching are even more complex and could not yet demonstrate their general effectiveness.

Secondly, our approach is generic as it can be used for different domains and in combination with different matching tools for the first step. We can even re-use the tool in different fields, such as entity resolution or text mining. On the other hand, this can also be a disadvantage, since the enrichment step depends on the completeness and quality of the initially determined match result. Therefore, it is important to use powerful tools for the initial matching and possibly to fine-tune their configuration.

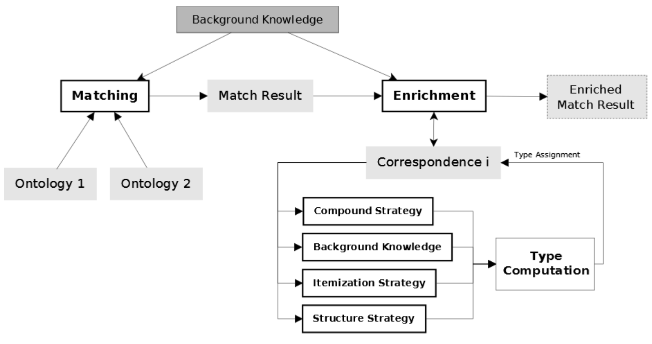


**Figure 2: Basic Workflow for Mapping Enrichment**

The basics of the relation type detection, on which we focus in this study, can be seen in the right part of Fig. 2. We provide 4 strategies so far (Compound, Background Knowledge, Itemization, Structure), where each strategy returns the relation type of a given correspondence, or "undecided" in case no specific type can be determined. In the Enrichment step we thus iterate through each correspondence in the mapping and pass it to each strategy. We eventually annotate the type that was most frequently returned by the strategies (type computation). In this study, we regard 4 distinct relation types: equal, is-a and inv. is-a (composition), part-of and has-a (aggregation), as well as related.

There are two problems we may encounter when computing the correspondence type. First, all strategies may return

| **Strategy** | equal | is-a | part-of | related |
|---|---|---|---|---|
| Compounding | | X | | |
| Background K. | X | X | X | X |
| Itemization | X | X | | |
| Structure | | X | X | |

**Table 1: Supported correspondence types by the strategies**

"undecided". In this case we assign the relation type "equal", because it is the default type in the initial match result and possibly the most likely one to hold. Secondly, there might be different outcomes from the strategies, e.g., one returns is-a, one equal and the others undecided. There are different ways to solve this problem, e.g., by prioritizing strategies or relation types. However, we hardly discovered such cases so far, so we currently return "undecided" and request the user to manually specify the correct type.

At the present, our approach is already able to fully assign relation types to an input mapping using the 4 strategies, which we will describe in detail in the next section. We have not implemented strategies to create complex matches from the match result, but will address a couple of conceivable techniques in section 5.

## 4. IMPLEMENTED STRATEGIES

We have implemented 4 strategies to determine the type of a given correspondence. Table 1 gives an overview of the strategies and the relation types they are able to detect. It can be seen that the Background Knowledge approach is especially valuable, as it can help to detect all relationship types. Besides, all strategies are able to identify is-a correspondences.

In the following let $O_1, O_2$ be two ontologies with $c_1, c_2$ being two concepts from $O_1$ resp. $O_2$. Further, let $C = (c_1, c_2)$ be a correspondence between two concepts (we do not regard the confidence value in this study).

### 4.1 Compound Strategy

In linguistics, a compound is a special word $W$ that consists of a head $W_H$ carrying the basic meaning of $W$, and a modifier $W_M$ that specifies $W_H$ [3]. In many cases, a compound thus expresses something more specific than its head, and is therefore a perfect candidate to discover an is-a relationship. For instance, a blackboard is a board or an apple tree is a tree. Such compounds are called *endocentric compounds*, while *exocentric compounds* are not related with their head, such as buttercup, which is not a cup, or saw tooth, which is not a tooth. These compounds are of literal meaning (metaphors) or changed their spelling as the language evolved, and thus do not hold the is-a relation, or only to a very limited extent (like airport, which is a port only in a broad sense). There is a third form of compounds, called *appositional* or *copulative* compounds, where the two words are at the same level, and the relation is rather more-general (inverse is-a) than more-specific, as in Bosnia-Herzegowina, which means both Bosnia and Herzegowina, or bitter-sweet, which means both bitter and sweet (not necessarily a "specific bitter" or a "specific sweet"). However, this type is quite rare.

In the following, let $A$, $B$ be the literals of two concepts of a correspondence. The Compound Strategy analyzes whether $B$ ends with $A$. If so, it seems likely that $B$

is a compound with head $A$, so that the relationship $B$ is-a $A$ (or $A$ inv. is-a $B$) is likely to hold. The Compound approach allows us to identify the three is-a correspondences shown in Figure 1 (below).

We added an additional rule to this simple approach: $B$ is only considered a compound to $A$ if $length(B) - length(A) \geq 3$, where $length(X)$ is the length of a string $X$. Thus, we expect the supposed compound to be at least 3 characters longer than the head it matches. This way, we are able to eliminate obviously wrong compound conclusions, like *stable* is a *table*, which we call *pseudo compounds*. The value of 3 is motivated by the observation that typical nouns or adjectives consist of at least 3 letters.

## 4.2 Background Knowledge

Background knowledge is commonly of great help in ontology matching to detect more difficult correspondences, especially in special domains. In our approach, we intend to use it for relation type detection. So far, we use WordNet 3.0 to determine the relation that holds between two words (resp. two concepts). WordNet is a powerful dictionary and thesaurus that contains synonym relations (equivalence), hypernym relations (is-a) and holonym relations (part-of) between words [22]. Using the Java API for WordNet Search (JAWS), we built an interface that allows to answer questions like "Is X a synonym to Y?", or "Is X a direct hypernym of Y?". The interface is also able to detect cohyponyms, which are two words $X, Y$ that have a common direct hypernym $Z$. We call a correspondence between two cohyponyms $X$ and $Y$ related, because both concepts are connected to the same father element. For example, the relation between *apple tree* and *pear tree* is related, because of the common father concept *tree*.

Although WordNet has a limited vocabulary, especially with regard to specific domains, it is a valuable source to detect the relation type that holds between concepts. It allows an excellent precision, because the links in WordNet are manually defined, and contains all relation types we intend to detect, which the other strategies are not able to achieve.

## 4.3 Itemization

In several taxonomies we recognized that itemizations appear very often, and which cannot be processed with the previously presented strategies. Consider the correspondence ("books and newspapers", "newspapers"). The compound strategy would be mislead and consider the source concept a compound, resulting in the type "is-a", although the opposite is the case (inv. is-a). WordNet would not know the word "books and newspapers" and return "undecided".

Itemizations thus deserve special treatment. We first split each itemization in its atomic items, where we define an item as a string that does not contain commas, slashes or the words "and" and "or".

We now show how our approach determines the correspondence types between two concepts $C_1, C_2$ where at least one of the two concepts is an itemization with more than one item. Let $I_1$ be the item set of $C_1$ and $I_2$ the item set of $C_2$. Let $w_1, w_2$ be two words, with $w_1 \neq w_2$. Our approach works as follows:

1. In each set $I$ remove each $w_1 \in I$ which is a hyponym of $w_2 \in I$.
2. In each set $I$, replace a synonym pair ($w_1 \in I, w_2 \in I$) by $w_1$.
3. Remove each $w_1 \in I_1$, $w_2 \in I_2$ if there is a synonym pair $(w_1, w_2)$.
4. Remove each $w_2 \in I_2$ which is a hyponym of $w_1 \in I_1$.
5. Determine the relation type:
   (a) If $I_1 = \emptyset, I_2 = \emptyset$: equal
   (b) If $I_1 = \emptyset, |I_2| \geq 1$: is-a
   If $I_2 = \emptyset, |I_1| \geq 1$: inverse is-a
   (c) If $|I_1| \geq 1, I_2 \geq 1$: undecided

The rationale behind this algorithm is that we remove items from the item sets as long as no information gets lost. Then we compare what is left in the two sets and come to the conclusions presented in step 5.

Let us consider the concept pair $C_1$ = "books, ebooks, movies, films, cds" and $C_2$ ="novels, cds". Our item sets are $I_1 = \{books, ebooks, movies, films, cds\}$, $I_2 = \{novels, cds\}$. First, we remove synonyms and hyponyms within each set, because this would cause no loss of information (steps 1+2). We remove $films$ in $I_1$ (because of the synonym *movies*) and *ebooks* in $I_1$, because it is a hyponym of *books*. We have $I_1 = \{books, movies, cds\}$ , $I_2 = \{novels, cds\}$. Now we remove synonym pairs between the two item sets, so we remove *cds* in either set (step 3). Lastly, we remove a hyponym in $I_1$ if there is a hypernym in $I_2$ (step 4). We remove *novel* in $I_2$, because it is a *book*. We have $I_1 = \{books, movies\}$ , $I_2 = \emptyset$. Since $I_1$ still contains items, while $I_2$ is empty, we conclude that $I_1$ specifies something more general, i.e., it holds $C_1$ inverse is-a $C_2$.

If neither item set is empty, we return "undecided" because we cannot derive an equal or is-a relationship in this case.

## 4.4 Structure Strategy

The structure strategy takes the structure of the ontologies into account. For a correspondence between concepts $Y$ and $Z$ we check whether we can derive a semantic relationship between a father concept $X$ of $Y$ and $Z$ (or vice versa). For an is-a relationship between $Y$ and $X$ we draw the following conclusions:

- $X$ equiv $Z \rightarrow Y$ is-a $Z$
- $X$ is-a $Z \rightarrow Y$ is-a $Z$

For a part-of relationship between $Y$ and $X$ we can analogously derive:

- $X$ equiv $Z \rightarrow Y$ part-of $Z$
- $X$ part-of $Z \rightarrow Y$ part-of $Z$

The approach obviously utilizes the semantics of the intra-ontology relationships to determine the correspondence types for pairs of concepts for which the semantic relationship cannot directly be determined.

## 4.5 Comparison

We tested our strategies and overall system on 3 user-generated mappings in which each correspondence was tagged with its supposed type. After running the scenarios, we checked how many of the non-trivial relations were detected by the program. The 3 scenario consisted of about 350

.. 750 correspondences. We had a German-language scenario (product catalogs from online shops), a health scenario (diseases) and a text annotation catalog scenario (everyday speech).

Compounding and Background Knowledge are two independent strategies that separately try to determine the relation type of a correspondence. In our tests we saw that Compounding offers a good precision (72 .. 97 %), even without the many exocentric and pseudo-compounds that exist. By contrast, we recognized only moderate recall, ranging from 12 to 43 %. Compounding is only able to determine is-a relations, however, it is the only strategy that invariably works.

Background Knowledge has a low or moderate recall (10 .. 50 %), depending on the scenario at hand. However, it offers an excellent precision being very close to 100 % and is the only strategy that is able to determine all relation types we regard. As matter of fact, it did not work on our German-language example and only poorly in our health scenario.

Structure and Itemization strategy depend much on the given schemas and are thus very specific strategies to handle individual cases. They exploit the Compound and Background Knowledge Strategy and are thus not independent. Still, they were able to boost the recall to some degree.

We realized that the best result is gained by exploiting all strategies. Currently, we do not weight the strategies, however, we may do so in order to optimize our system. We finally achieved an overall recall between 46 and 65 % and precision between 69 and 97 %.

## 5. COMPLEX CORRESPONDENCES

Schema and ontology matching tools generally calculate (1:1)-correspondences, where exactly one source element matches exactly one target element. Naturally, either element may take part in different correspondences, as in (name, first name) and (name, last name), however, having these two separate correspondences is very imprecise and the correct mapping would rather be the single correspondence ( (first name, last name), (name)). These kind of matches are called complex correspondences or one-to-many correspondences.

The disambiguation between a complex correspondence or 2 (or more) one-to-one correspondences is an inevitable premise for data transformation where data from a source database is to be transformed into a target database, which we could show in [1]. Moreover, we could prove that each complex correspondence needs a transformation function in order to correctly map data. If elements are of the type string, the transformation function is normally concatenation in (n:1)-matches and split in (1:n)-matches. If the elements are of a numerical type, as in the correspondence ( (costs), ((operational costs), (material costs), (personnel costs))), a set of numerical operations is normally required.

There are proprietary solutions that allow to manually create transformation mappings including complex correspondences, such as Microsoft Biztalk Server [19], Altova MapForce [18] or Stylus Studio [20], however, to the best of our knowledge there is no matching tool that is able to detect complex correspondences automatically. Next to relation type detection, we therefore intend to discover complex correspondences in the initial mapping, which is a second important step of mapping enrichment.

We already developed simple methods that exploit the

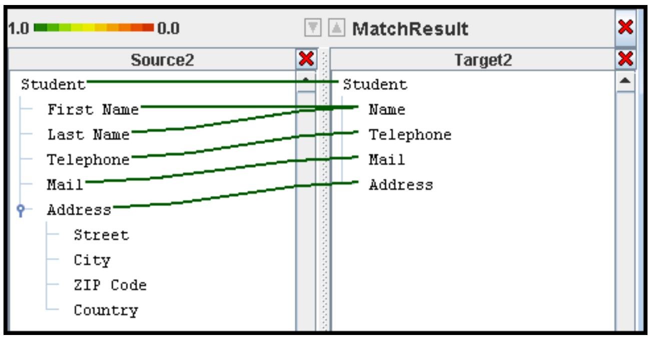


**Figure 3: Match result containing two complex correspondences (name and address)**

structure of the schemas to transform several (1:1)-correspondences into a complex correspondence, although these approaches will fail in more intricate scenarios. We used the structure of the schemas and the already existing (1:1)-matches to derive complex correspondences. Fig. 3 demonstrates this approach. There are two complex correspondences in the mapping, ( (First Name, Last Name), (Name)) and ( (Street, City, Zip Code, Country), Address), represented by simple (1:1)-correspondences. Our approach was able to detect both complex correspondences. The first one (name) was detected, because first name and last name cannot be mapped to one element at the same time, since the name element can only store either of the two values. The second example (address) is detected since schema data is located in the leaf nodes, not in inner nodes. In database schemas we always expect data to reside in the leaf nodes, so that the match (Address, Address) is considered unreasonable.

In the first case, our approach would apply the concatenation function, because two values have to be concatenated to match the target value, and in the second case the split function would be applied, because the Address values have to be split into the address components (street, city, zip code, country). The user needs to adjust these functions, e.g., in order to tell the program where in the address string the split operations have to be performed.

This approach was mostly based on heuristics and would only work in simple cases. Now that we are able to determine the relation types of (1:1)-matches, we can enhance this original approach. If a node takes part in more than one composition relation (part-of / has-a), we can conclude that it is a complex correspondence and can derive it from the (1:1)-correspondences. For instance, if we have the 3 correspondences (day part-of date), (month part-of date), (year part-of date) we could create the complex correspondence ( (day, month, year), date).

We have not implemented this approach so far, and we assume that detecting complex correspondences and the correct transformation function will still remain a very challenging issue, so that we intend to investigate additional methods like using instance data to allow more effectiveness. However, adding these techniques to our existing Enrichment Engine, we are able to present a first solution that semi-automatically determines complex correspondences, which is another step towards more precise ontology matching, and an important condition for data transformation.

## 6. OUTLOOK AND CONCLUSION

We presented a new approach to semantically enrich ontology mappings by determining the concrete relation type of a correspondence and detecting complex correspondences. For this, we developed a 2-step architecture in which the actual ontology matching and the semantic enrichment are strictly separated. This makes the Enrichment Engine highly generic so that it is not designed for any specific ontology matching tool, and moreover, can be used independently in various fields different from ontology matching, such as data transformation, entity resolution and text mining.

In our approach we developed new linguistic strategies to determine the relation type, and with regard to our first internal tests even the rather simple strategies already added much useful information to the input mapping. We also discovered that some strategies (Compounding, and to a less degree Itemization and Structure) are rather independent from the language of the ontologies, so that our approach provided remarkable results both in German and English-language ontologies.

One important obstacle is the strong dependency to the initial mapping. We recognized that matching tools tend to discover equivalence relations, so that different non-equivalence correspondences are not contained by the initial mapping, and can thus not be detected. It is future work to adjust our tool COMA 3.0 to provide a more convenient input, e.g., by using relaxed configurations. A particular issue we are going to investigate is the use of instance data connected with the concepts to derive the correct relation type if the other strategies (which operate on the meta level) fail. This will also result in a time-complexity problem, which we will have to consider in our ongoing research.

Our approach is still in a rather early state, and there is still much space for improvement, since the implemented strategies have different restrictions so far. For this reason, we will extend and fine-tune our tool in order to increase effectiveness and precision. Among other aspects, we intend to improve the structure strategy by considering the entire concept path rather than the mere father concept, to add further background knowledge to the system, especially in specific domains, and to investigate further linguistic strategies, for instance, in which way compounds also indicate the part-of relation. Next to relation type detection, we will also concentrate on complex correspondence detection in data transformation to provide further semantic information to ontology mappings.

## 7. ACKNOWLEDGMENT

This study was partly funded by the European Commission through Project "LinkedDesign" (No. 284613 FoF-ICT-2011.7.4).

## 8. REFERENCES

[1] Arnold P.: The Basics of Complex Correspondences and Functions and their Implementation and Semi-automatic Detection in COMA++ (Master's thesis), University of Leipzig, 2011.

[2] Bellahsene., Z., Bonifati, A., Rahm, E. (eds.): Schema Matching and Mapping, Springer (2011)

[3] Bisetto, A., Scalise, S.: Classification of Compounds. University of Bologna, 2009. In: The Oxford Handbook of Compounding, Oxford University Press, pp. 49-82.

[4] Dhamankar, R., Yoonkyong, L., Doan, A., Halevy, A., Domingos, P.: iMAP: Discovering Complex Semantic Matches between Database Schemas. In: SIGMOD '04, pp. 383–394

[5] Doan, A., Halevy, A. Y.: Semantic Integration Research in the Database Community: A Brief Survey. In AI Mag. (2005), pp. 83–94

[6] Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: An Algorithm and an Implementation of Semantic Matching. Proceedings of the European Semantic Web Symposium (2004), LNCS 3053, pp. 61–75

[7] Giunchiglia, F., Autayeu, A., Pane, J.: S-Match: an open source framework for matching lightweight ontologies. In: Semantic Web, vol. 3-3 (2012), pp. 307-317

[8] He, B., Chen-Chuan Chang, H., Han, J.: Discovering complex matchings across web query interfaces: A correlation mining approach. In: KDD '04, pp. 148–157

[9] Jiménez-Ruiz, E., Grau, B. C.: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference (2011), LNCS 7031, pp. 273–288

[10] van Hage, W. R., Katrenko, S., Schreiber, G. A Method to Combine Linguistic Ontology-Mapping Techniques. In: International Semantic Web Conference (2005), LNCS 3729, pp. 732–744

[11] Hamdi, F., Safar, B., Niraula, N. B., Reynaud, C.: TaxoMap alignment and refinement modules: Results for OAEI 2010. Proceedings of the ISWC Workshop (2010), pp. 212–219

[12] Massmann, S., Raunich, S., Aumueller, D., Arnold, P., Rahm, E. Evolution of the COMA Match System. Proc. Sixth Intern. Workshop on Ontology Matching (2011)

[13] Raunich, S.,Rahm, E.: ATOM: Automatic Target-driven Ontology Merging. Proc. Int. Conf. on Data Engineering (2011)

[14] Reynaud, C., Safar, B.: Exploiting WordNet as Background Knowledge. Proc. Intern. ISWCŠ07 Ontology Matching (OM-07) Workshop

[15] Sabou, M., d'Aquin, M., Motta, E.: Using the semantic web as background knowledge for ontology mapping. Proc. 1st Intern. Workshop on on Ontology Matching (2006).

[16] Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches. J. Data Semantics IV (2005), pp. 146–171

[17] Spiliopoulos, V., Vouros, G., Karkaletsis, V: On the discovery of subsumption relations for the alignment of ontologies. Web Semantics: Science, Services and Agents on the World Wide Web 8 (2010), pp. 69-88

[18] Altova MapForce - Graphical Data Mapping, Conversion, and Integration Tool. `http://www.altova.com/mapforce.html`

[19] Microsoft BizTalk Server. `http://www.microsoft.com/biztalk`

[20] XML Editor, XML Tools, and XQuery - Stylus Studio. `http://www.stylusstudio.com/`

[21] Java API for WordNet Searching (JAWS), `http://lyle.smu.edu/~tspell/jaws/index.html`

[22] WordNet - A lexical database for English, `http://wordnet.princeton.edu/wordnet/`

# Extraktion und Anreicherung von Merkmalshierarchien durch Analyse unstrukturierter Produktrezensionen

Robin Küppers
Institut für Informatik
Heinrich-Heine-Universität
Universitätsstr. 1
40225 Düsseldorf, Deutschland
kueppers@cs.uni-duesseldorf.de

## ABSTRACT

Wir präsentieren einen Algorithmus zur Extraktion bzw. Anreicherung von hierarchischen Produktmerkmalen mittels einer Analyse von unstrukturierten, kundengenerierten Produktrezensionen. Unser Algorithmus benötigt eine initiale Merkmalshierarchie, die in einem rekursiven Verfahren mit neuen Untermerkmalen angereichert wird, wobei die natürliche Ordnung der Merkmale beibehalten wird. Die Funktionsweise unseres Algorithmus basiert auf häufigen, grammatikalischen Strukturen, die in Produktrezensionen oft benutzt werden, um Eigenschaften eines Produkts zu beschreiben. Diese Strukturen beschreiben Obermerkmale im Kontext ihrer Untermerkmale und werden von unserem Algorithmus ausgenutzt, um Merkmale hierarchisch zu ordnen.

## Kategorien

H.2.8 [**Database Management**]: Database Applications—*data mining*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*text analysis*

## Schlüsselwörter

Text Mining, Review Analysis, Product Feature

## 1. EINLEITUNG

Der Einkauf von Waren (z. B. Kameras) und Dienstleistungen (z. B. Hotels) über Web-Shops wie Amazon unterliegt seit Jahren einem stetigen Wachstum. Web-Shops geben ihren Kunden (i. d. R.) die Möglichkeit die gekaufte Ware in Form einer Rezension zu kommentieren und zu bewerten. Diese kundengenerierten Rezensionen enthalten wertvolle Informationen über das Produkt, die von potentiellen Kunden für ihre Kaufentscheidung herangezogen werden. Je positiver ein Produkt bewertet wird, desto wahrscheinlicher wird es von anderen Kunden gekauft.
Der Kunde kann sich so ausführlicher über die Vor- und Nachteile eines Produkts informieren, als dies über redaktionelle Datenblätter oder Produktbeschreibungen möglich wäre, da diese dazu tendieren, die Vorteile eines Produkts zu beleuchten und die Nachteile zu verschweigen. Aus diesem Grund haben potentielle Kunden ein berechtigtes Interesse an der subjektiven Meinung anderer Käufer.
Zudem sind kundengenerierte Produktrezensionen auch für Produzenten interessant, da sie wertvolle Informationen über Qualität und Marktakzeptanz eines Produkts aus Kundensicht enthalten. Diese Informationen können Produzenten dabei helfen, die eigene Produktpalette zu optimieren und besser an Kundenbedürfnisse anzupassen.
Mit wachsendem Umsatz der Web-Shops nimmt auch die Anzahl der Produktrezensionen stetig zu, so dass es für Kunden (und Produzenten) immer schwieriger wird, einen umfassenden Überblick über ein Produkt / eine Produktgruppe zu behalten. Deshalb ist unser Ziel eine feingranulare Zusammenfassung von Produktrezensionen, die es erlaubt Produkte dynamisch anhand von Produktmerkmalen (*product features*) zu bewerten und mit ähnlichen Produkten zu vergleichen. Auf diese Weise wird ein Kunde in die Lage versetzt ein Produkt im Kontext seines eigenen Bedürfnisses zu betrachten und zu bewerten: beispielsweise spielt das Gewicht einer Kamera keine große Rolle für einen Kunden, aber es wird viel Wert auf eine hohe Bildqualität gelegt. Produzenten können ihre eigene Produktpalette im Kontext der Konkurrenz analysieren, um z. B. Mängel an den eigenen Produkten zu identifizieren.
Das Ziel unserer Forschung ist ein Gesamtsystem zur Analyse und Präsentation von Produktrezensionen in zusammengefasster Form (vgl. [3]). Dieses System besteht aus mehreren Komponenten, die verschiedene Aufgaben übernehmen, wie z.B. die Extraktion von Meinungen und die Bestimmung der Tonalität bezüglich eines Produktmerkmals (siehe dazu auch Abschnitt 2). Im Rahmen dieser Arbeit beschränken wir uns auf einen wichtigen Teilaspekt dieses Systems: die Extraktion und Anreicherung von hierarchisch organisierten Produktmerkmalen.
Der Rest dieser Arbeit ist wie folgt gegliedert: zunächst geben wir in Abschnitt 2 einen Überblick über verwandte Arbeiten, die auf unsere Forschung entscheidenen Einfluss hatten. Anschließend präsentieren wir in Abschnitt 3 einen Algorithmus zur Extraktion und zur Anreicherung von hierarchisch organisierten Produktmerkmalen. Eine Bewertung des Algorithmus wird in Abschnitt 4 vorgenommen, sowie einige Ergebnisse präsentiert, die die Effektivität unseres Algorithmus demonstrieren. Die gewonnenen Erkenntnisse werden in Abschnitt 5 diskutiert und zusammengefasst. Des

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

Weiteren geben wir einen Ausblick auf unsere zukünftige Forschung.

## 2. VERWANDTE ARBEITEN

Dieser Abschnitt gibt einen kurzen Überblick über verwandte Arbeiten, die einen Einfluss auf unsere Forschung hatten. Die Analyse von Produktrezensionen basiert auf Algorithmen und Methoden aus verschiedensten Disziplinen. Zu den Wichtigsten zählen: Feature Extraction, Opining Mining und Sentiment Analysis.
Ein typischer Algorithmus zur merkmalsbasierten Tonalitätsanalyse von Produktrezensionen ist in 3 unterschiedliche Phasen unterteilt (vgl. [3]):

1. Extraktion von Produktmerkmalen.
2. Extraktion von Meinungen über Produktmerkmale.
3. Tonalitätsanalyse der Meinungen.

Man unterscheidet zwischen impliziten und expliziten Merkmalen[3]: explizite Merkmale werden direkt im Text genannt, implizite Merkmale müssen aus dem Kontext erschlossen werden. Wir beschränken uns im Rahmen dieser Arbeit auf die Extraktion expliziter Merkmale.
Die Autoren von [3] extrahieren häufig auftretende, explizite Merkmale mit dem a-priori Algorithmus. Mit Hilfe dieser Produktmerkmale werden Meinungen aus dem Text extrahiert, die sich auf ein Produktmerkmal beziehen. Die Tonalität einer Meinung wird auf die Tonalität der enthaltenen Adjektive zurückgeführt. Die extrahierten Merkmale werden - im Gegensatz zu unserer Arbeit - nicht hierarchisch modelliert.
Es gibt auch Ansätze, die versuchen die natürliche Hierarchie von Produktmerkmalen abzubilden. Die Autoren von [1] nutzen die tabellarische Struktur von Produktbeschreibungen aus, um explizite Produktmerkmale zu extrahieren, wobei die hierarchische Struktur aus der Tabellenstruktur abgeleitet wird. Einen ähnlichen Ansatz verfolgen [5] et. al.: die Autoren nutzen ebenfalls die oftmals hochgradige Strukturierung von Produktbeschreibungen aus. Die Produktmerkmale werden mit Clusteringtechniken aus einem Korpus extrahiert, wobei die Hierarchie der Merkmale durch das Clustering vorgegeben wird. Die Extraktion von expliziten Merkmalen aus strukturierten Texten ist (i. d. R.) einfacher, als durch Analyse unstrukturierter Daten.
Die Methode von [2] et. al. benutzt eine Taxonomie zur Abbildung der Merkmalshierarchie, wobei diese von einem Experten erstellt wird. Diese Hierarchie bildet die Grundlage für die Meinungsextraktion. Die Tonalität der Meinungen wird über ein Tonalitätswörterbuch gelöst. Für diesen Ansatz wird - im Gegensatz zu unserer Methode - umfangreiches Expertenwissen benötigt.
Die Arbeit von [8] et. al. konzentriert sich auf die Extraktion von Meinungen und die anschließende Tonalitätsanalyse. Die Autoren unterscheiden zwischen subjektiven und komparativen Sätze. Sowohl subjektive, als auch komparative Sätze enthalten Meinungen, wobei im komparativen Fall eine Meinung nicht direkt gegeben wird, sondern über einen Vergleich mit einem anderen Produkt erfolgt. Die Autoren nutzen komparative Sätze, um Produktgraphen zu erzeugen mit deren Hilfe verschiedene Produkte hinsichtlich eines Merkmals geordnet werden können. Die notwendigen Tonalitätswerte werden einem Wörterbuch entnommen.

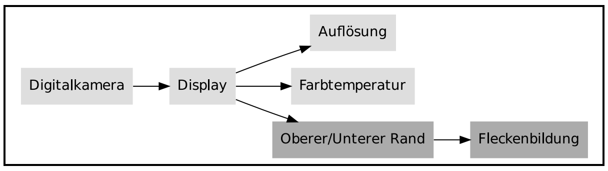


**Abbildung 1: Beispielhafte Merkmalshierarchie einer Digitalkamera.**

Wir haben hauptsächlich Arbeiten vorgestellt, die Merkmale und Meinungen aus Produktrezensionen extrahieren, aber Meinungsanalysen sind auch für andere Domänen interessant: z. B. verwenden die Autoren von [7] einen von Experten annotierten Korpus mit Nachrichten, um mit Techniken des maschinellen Lernens einen Klassifikator zu trainieren, der zwischen Aussagen (Meinungen) und Nicht-Aussagen unterscheidet. Solche Ansätze sind nicht auf die Extraktion von Produktmerkmalen angewiesen.

## 3. ANREICHERUNG VON MERKMALSHIERARCHIEN

Dieser Abschnitt dient der Beschreibung eines neuen Algorithmus zur Anreicherung einer gegebenen, unvollständigen Merkmalshierarchie mit zusätzlichen Merkmalen. Diese Merkmale werden aus unstrukturierten kundengenerierten Produktrezensionen gewonnen, wobei versucht wird die natürliche Ordnung der Merkmale (Unter- bzw. Obermerkmalsbeziehung) zu beachten.
Die Merkmalshierarchie bildet die Basis für weitergehende Analysen, wie z.B. die gezielte Extraktion von Meinungen und Tonalitäten, die sich auf Produktmerkmale beziehen. Diese nachfolgenden Analyseschritte sind nicht mehr Gegenstand dieser Arbeit. Produkte (aber auch Dienstleistungen) können durch eine Menge von Merkmalen (*product features*) beschrieben werden. Produktmerkmale folgen dabei einer natürlichen, domänenabhängigen Ordnung. Eine derartige natürliche Hierarchie ist exemplarisch in Abbildung 1 für das Produkt **Digitalkamera** dargestellt. Offensichtlich ist **Display** ein Untermerkmal von **Digitalkamera** und besitzt eigene Untermerkmale **Auflösung** und **Farbtemperatur**.
Hierarchien von Produktmerkmalen können auf Basis von strukturierten Texten erzeugt werden, wie z. B. technische Datenblättern und Produktbeschreibungen (vgl. [5]). Diese Datenquellen enthalten i. d. R. die wichtigsten Produktmerkmale. Der hohe Strukturierungsgrad dieser Datenquellen erlaubt eine Extraktion der Merkmale mit hoher Genauigkeit ($\approx 71\%$ [5]). Allerdings tendieren Datenblätter und Produktbeschreibungen dazu, ein Produkt relativ oberflächlich darzustellen oder zu Gunsten des Produkts zu verzerren. Zum Beispiel enthält die Hierarchie in Abbildung 1 eine Reihe von Merkmalen, wie sie häufig in strukturierten Datenquellen zu finden sind (helle Knoten). Allerdings sind weitere, detailliertere Merkmale denkbar, die für eine Kaufentscheidung von Interesse sein könnten. Beispielsweise könnte das **Display** einer Digitalkamera zur Fleckenbildung am unteren/oberen Rand neigen. **Unterer/Oberer Rand** wird in diesem Fall zu einem Untermerkmal von **Display** und Obermerkmal von **Fleckenbildung** (dunkle Knoten).
Eine derartige Anreicherung einer gegebenen, unvollständigen Merkmalshierarchie kann durch die Verarbeitung von

kundengenerierten, unstrukturierten Rezensionen erfolgen. Wir halten einen hybriden Ansatz für durchaus sinnvoll: zunächst wird eine initiale Merkmalshierarchie mit hoher Genauigkeit aus strukturierten Daten gewonnen. Anschließend wird diese Hierarchie in einer zweiten Verarbeitungshase mit zusätzlichen Produktmerkmalen angereichert.
Für den weiteren Verlauf dieses Abschnitts beschränken wir uns auf die zweite Analysephase, d.h. wir nehmen eine initiale Merkmalshierarchie als gegeben an. Für die Evaluation unseres Algorithmus (siehe Abschnitt 4) wurden die initialen Merkmalshierarchien manuell erzeugt.
Unser Algorithmus wurde auf der Basis einer Reihe von einfachen Beobachtungen entworfen, die wir bei der Analyse von unserem Rezensionskorpus gemacht haben.

1. Ein Produktmerkmal wird häufig durch ein Hauptwort repräsentiert.
2. Viele Hauptwörter können dasselbe Produktmerkmal beschreiben. (Synonyme)
3. Untermerkmale werden häufig im Kontext ihrer Obermerkmale genannt, wie z. B. "das Ladegerät der Kamera".
4. Textfragmente, die von Produktmerkmalen handeln, besitzen häufig eine sehr ähnliche grammatikalische Struktur, wie z.B. "die Auflösung der Anzeige" oder "die Laufzeit des Akkus", wobei Unter- und Obermerkmale gemeinsam genannt werden. Die Struktur der Fragmente lautet [DET, NOUN, DET, NOUN], wobei DET einen Artikel und NOUN ein Hauptwort beschreibt.

Der Rest dieses Abschnitts gliedert sich wie folgt: zunächst werden Definitionen in Unterabschnitt 3.1 eingeführt, die für das weitere Verständnis notwendig sind. Anschließend beschreiben wir unsere Analysepipeline, die für die Vorverarbeitung der Produktrezensionen verwendet wurde, in Unterabschnitt 3.2. Darauf aufbauend wird in Unterabschnitt 3.3 unser Algorithmus im Detail besprochen.

## 3.1 Definitionen

Für das Verständnis der nächsten Abschnitte werden einige Begriffe benötigt, die in diesem Unterabschnitt definiert werden sollen:

*Token.* Ein Token $t$ ist ein Paar $t = (v_{word}, v_{POS})$, wobei $v_{word}$ das Wort und $v_{pos}$ die Wortart angibt. Im Rahmen dieser Arbeit wurde das Universal Tagset [6] benutzt.

*Merkmal.* Wir definieren ein Produktmerkmal $f$ als ein Tripel $f = (S, C, p)$, wobei $S$ eine Menge von Synonymen beschreibt, die als textuelle Realisierung eines Merkmals Verwendung finden können. Die Elemente von $S$ können Worte, Produktbezeichnungen und auch Abkürzungen enthalten. Die Hierarchie wird über $C$ und $p$ kontrolliert, wobei $C$ eine Menge von Untermerkmalen und $p$ das Obermerkmal von $f$ angibt. Das Wurzelelement einer Hierarchie beschreibt das Produkt/die Produktgruppe selbst und besitzt kein Obermerkmal.

*POS-Muster.* Ein POS-Muster $q$ ist eine geordnete Sequenz von POS-Tags $p = [tag_1, tag_2, \ldots, tag_n]$, wobei $n$ die Musterlänge beschreibt. Ein POS-Tag beschreibt eine Wortart, z.B. steht DET für einen Artikel, NOUN für ein Hauptwort und ADJ für ein Adjektiv. Weitere Informationen über das Universal Tagset finden sich in [6].

## 3.2 Analysepipeline

Für die Verarbeitung und Untersuchung der Produktrezensionen haben wir eine für den NLP-Bereich (Natural Language Processing) typische Standardpipeline benutzt: die Volltexte der Rezensionen sind für unsere Zwecke zu grobgranular, so dass in einer ersten Phase der Volltext in Sätze zerteilt wird. Anschließend werden die Sätze tokenisiert und die Wortarten der einzelnen Worte bestimmt. Des Weiteren werden Stoppworte markiert - dafür werden Standard-Stoppwortlisten benutzt. Wir beenden die Analysepipeline mit einer Stammformreduktion für jedes Wort, um die verschiedenen Flexionsformen eines Wortes auf eine kanonische Basis zu bringen.
Für die Bestimmung zusätzlicher Produktmerkmale aus Produktrezensionen, sind vor allem Hauptworte interessant, die i. d. R. keine Stoppworte sind. Allerdings ist uns aufgefallen, dass überdurchschnittlich viele Worte fälschlicherweise als ein Hauptwort erkannt werden - viele dieser Worte sind Stoppworte. Wir nehmen an, dass die variierende, grammatikalische Qualität der Produktrezensionen für die hohe Anzahl falsch bestimmer Worte verantwortlich ist. Die Stoppwortmarkierung hilft dabei, diesen Fehler etwas auszugleichen.

## 3.3 Der Algorithmus

In diesem Abschnitt beschreiben wir einen neuen Algorithmus, um eine initiale Hierarchie von Produktmerkmalen mit zusätzlichen Merkmalen anzureichern, wobei die natürliche Ordnung der Merkmale erhalten bleibt (siehe Algorithmus 1).
Der Algorithmus erwartet 3 Parameter: eine 2-dimensionale Liste von Token $T$, die sämtliche Token für jeden Satz enthält (dabei beschreibt die erste Dimension die Sätze, die zweite Dimensionen die einzelnen Wörter), eine initiale Hierarchie von Merkmalen $f$ und eine Menge von POS-Mustern $P$. Da der Algorithmus rekursiv arbeitet, wird zusätzlich ein Parameter $d$ übergeben, der die maximale Rekursionstiefe angibt. Der Algorithmus bricht ab, sobald die vorgegebene Tiefe erreicht wird (Zeile 1-3).

*Kandidatensuche (Zeile 4-11).* Um geeignete Kandidaten für neue Produktmerkmale zu finden, werden alle Sätze betrachtet und jeweils entschieden, ob der Satz eine Realisierung des aktuell betrachteten Merkmals enthält oder nicht. Wenn ein Satz eine Realisierung hat, dann wird die Funktion $applyPatterns$ aufgerufen. Diese Funktion sucht im übergebenen Satz nach gegebenen POS-Mustern und gibt – sofern mindestens ein Muster anwendbar ist – die entsprechenden Token als Kandidat zurück, wobei die Mustersuche auf das unmittelbare Umfeld der gefundenen Realisierung eingeschränkt wird, damit das korrekte POS-Muster zurückgeliefert wird, da POS-Muster mehrfach innerhalb eines Satzes vorkommen können.
Im Rahmen dieser Arbeit haben wie die folgenden POS-Muster verwendet:

- [DET, NOUN, DET, NOUN]
- [DET, NOUN, VERB, DET, ADJ, NOUN]

**Algorithm 1:** refineHierarchy

```
Eingabe : $T$ : Eine 2-dimensionale Liste von Token.
Eingabe : $P$ : Ein Array von POS-Mustern.
Eingabe : $f$ : Eine initiale Merkmalshierarchie.
Eingabe : $d$ : Die maximale Rekursionstiefe.
Ausgabe: Das Wurzelmerkmal der angereicherten
         Hierarchie.
1  if $d = 0$ then
2      return $f$
3  end
4  $C \leftarrow \{\}$ ;
5  for Token[] $T' \in T$ do
6      for Token $t \in T'$ do
7          if $t.word \in f.S$ then
8              $C \leftarrow C \bigcup applyPattern(T', P)$ ;
9          end
10     end
11 end
12 for Token[] $C' \in C$ do
13     for Token $t \in C'$ do
14         if $t.pos \neq NOUN$ then
15             next ;
16         end
17         if $t.length \leq 3$ then
18             next ;
19         end
20         if $hasParent(t.word, f)$ then
21             next ;
22         end
23         if $isSynonym(t.word, f.S)$ then
24             $f.S \leftarrow t.word$ ;
25             next ;
26         end
27         $f.C \leftarrow f.C \bigcup (\{t.word\}, \{\}, f)$ ;
28     end
29 end
30 for Feature[] $f' \in f.C$ do
31     $refineHierarchy(T, f', P, d-1)$;
32 end
```

***Validierungsphase (Zeile 12-29).*** Die Validierungsphase dient dazu die gefundenen Kandidaten zu validieren, also zu entscheiden, ob ein Kandidat ein neues Merkmal enthält. Man beachte, dass es sich bei diesem neuen Merkmal um ein Untermerkmal des aktuellen Produktmerkmals handelt, sofern es existiert. Für die Entscheidungsfindung nutzen wir eine Reihe von einfachen Heuristiken. Ein Token $t$ ist **kein** Produktmerkmal und wird übergangen, falls $t.v_{word}$:

1. kein Hauptwort ist (Zeile 14-16).
2. keine ausreichende Länge besitzt (Zeile 17-19).
3. ein Synonym von $f$ (oder eines Obermerkmals von $f$) ist (Zeile 20-22).
4. ein neues Synonym von $f$ darstellt (Zeile 23-26).

Die 3. Heuristik stellt sicher, dass sich keine Kreise in der Hierarchie bilden können. Man beachte, dass Obermerkmale, die nicht direkt voneinander abhängen, gleiche Untermerkmale tragen können.
Die 4. Heuristik dient zum Lernen von vorher unbekannten Synonymen. Dazu wird das Wort mit den Synonymen von $f$ verglichen (z.B. mit der Levenshtein-Distanz) und als Synonym aufgenommen, falls eine ausreichende Ähnlichkeit besteht. Damit soll verhindert werden, dass die falsche Schreibweise eines eigentlich bekannten Merkmals dazu führt, dass ein neuer Knoten in die Hierarchie eingefügt wird.
Wenn der Token $t$ die Heuristiken erfolgreich passiert hat, dann wird $t$ zu einem neuen Untermerkmal von $f$ (Zeile 27).

***Rekursiver Aufruf (Zeile 30-32).*** Nachdem das Merkmal $f$ nun mit zusätzlichen Merkmalen angereichert wurde, wird der Algorithmus rekursiv für alle Untermerkmale von $f$ aufgerufen, um diese mit weiteren Merkmalen zu versehen. Dieser Vorgang wiederholt sich solange, bis die maximale Rekursionstiefe erreicht wird.

***Nachbearbeitungsphase.*** Die Hierarchie, die von Algorithmus 1 erweitert wurde, muss in einer Nachbearbeitungsphase bereinigt werden, da viele Merkmale enthalten sind, die keine realen Produktmerkmale beschreiben (Rauschen). Für diese Arbeit verwenden wir die relative Häufigkeit eines Untermerkmals im Kontext seines Obermerkmals, um niederfrequente Merkmale (samt Untermerkmalen) aus der Hierarchie zu entfernen. Es sind aber auch andere Methoden denkbar, wie z.B. eine Gewichtung nach *tf-idf*[4]. Dabei wird nicht nur die Termhäufigkeit (*tf*) betrachtet, sondern auch die inverse Dokumenthäufigkeit (*idf*) mit einbezogen. Der *idf* eines Terms beschreibt die Bedeutsamkeit des Terms im Bezug auf die gesamte Dokumentenmenge.

## 4. EVALUATION

In diesem Abschnitt diskutieren wir die Vor- und Nachteile unseres Algorithmus. Um unseren Algorithmus evaluieren zu können, haben wir einen geeigneten Korpus aus Kundenrezensionen zusammengestellt. Unser Korpus besteht aus 4000 Kundenrezensionen von *amazon.de* aus der Produktgruppe **Digitalkamera**.
Wir haben unseren Algorithmus für die genannte Produktgruppe eine Hierarchie anreichern lassen. Die initiale Produkthierarchie enthält ein Obermerkmal, welches die Produktgruppe beschreibt. Zudem wurden häufig gebrauchte Synonyme hinzugefügt, wie z.B. **Gerät**. Im Weiteren präsentieren wir exemplarisch die angereicherte Hierarchie. Für dieses Experiment wurde die Rekursionstiefe auf 3 gesetzt, niederfrequente Merkmale (relative Häufigkeit $< 0,002$) wurden eliminiert. Wir haben für diese Arbeit Rezensionen in Deutscher Sprache verwendet, aber der Algorithmus kann leicht auf andere Sprachen angepasst werden. Die erzeugte Hierarchie ist in Abbildung 2 dargestellt. Es zeigt sich, dass unser Algorithmus – unter Beachtung der hierarchischen Struktur – eine Reihe wertvoller Merkmale extrahieren konnte: z. B. **Batterie** mit seinen Untermerkmalen **Haltezeit** und **Verbrauch** oder **Akkus** mit den Untermerkmalen **Auflad** und **Zukauf**. Es wurden aber auch viele Merkmale aus den Rezensionen extrahiert, die entweder keine echten Produktmerkmale sind (z.B. **Kompakt** oder eine falsche Ober-Untermerkmalsbeziehung abbilden (z. B. **Haptik** und **Kamera**). Des Weiteren sind einige Merkmale, wie z. B. **Qualität** zu generisch und sollten nicht als Produktmerkmal benutzt werden.

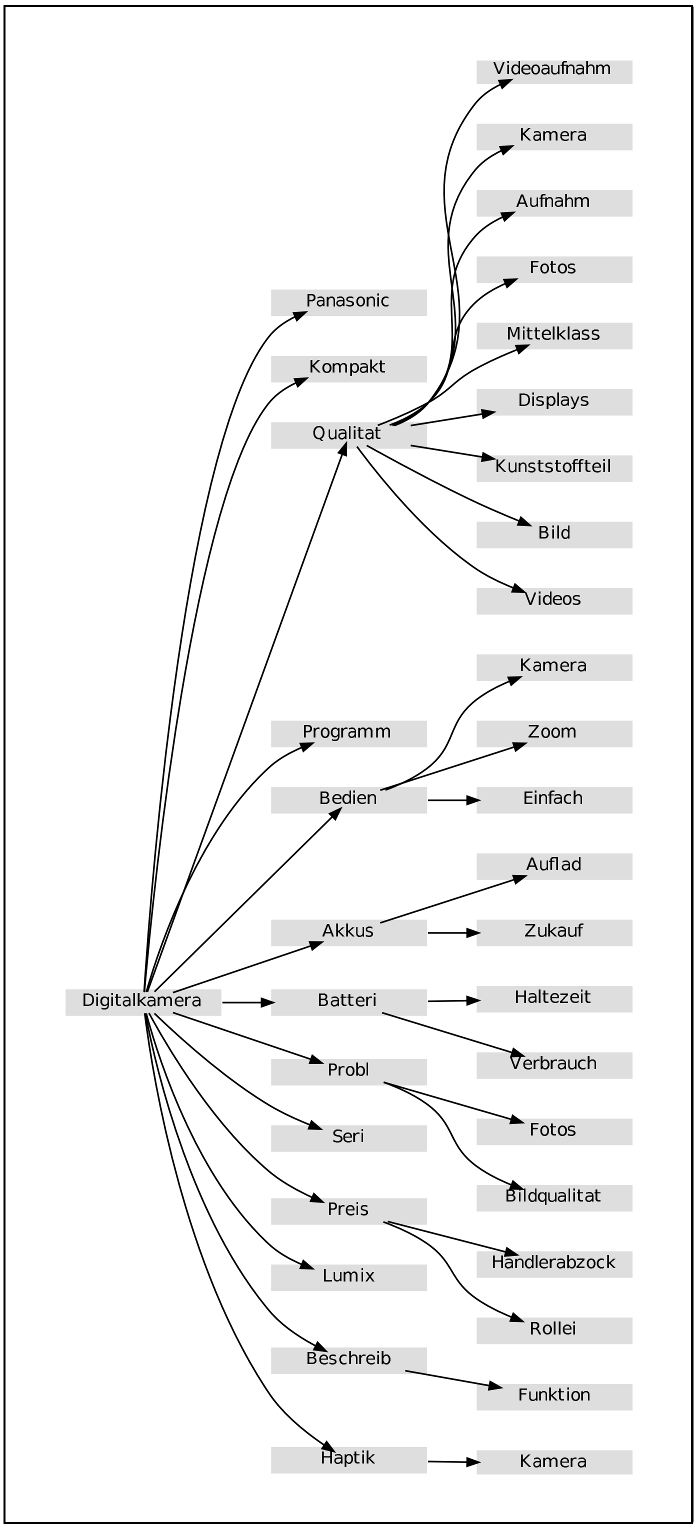


**Abbildung 2: Angereicherte Hierarchie für die Produktgruppe Digitalkamera.**

## 5. RESÜMEE UND AUSBLICK

In dieser Arbeit wurde ein neuer Algorithmus vorgestellt, der auf Basis einer gegebenen – möglicherweise flachen – Merkmalshierarchie diese Hierarchie mit zusätzlichen Merkmalen anreichert. Die neuen Merkmale werden automatisch aus unstrukturierten Produktrezensionen gewonnen, wobei der Algorithmus versucht die natürliche Ordnung der Produktmerkmale zu beachten.
Wir konnten zeigen, dass unser Algorithmus eine initiale Merkmalshierarchie mit sinnvollen Untermerkmalen anreichern kann, allerdings werden auch viele falsche Merkmale extrahiert und in fehlerhafte Merkmalsbeziehungen gebracht. Wir halten unseren Algorithmus dennoch für vielversprechend. Unsere weitere Forschung wird sich auf Teilaspekte dieser Arbeit konzentrieren:

- Die Merkmalsextraktion muss verbessert werden: wir haben beobachtet, dass eine Reihe extrahierter Merkmale keine echten Produktmerkmale beschreiben. Dabei handelt es sich häufig um sehr allgemeine Wörter wie z.B. **Möglichkeiten**. Wir bereiten deshalb den Aufbau einer Stoppwortliste für Produktrezensionen vor. Auf diese Weise könnte diese Problematik abgeschwächt werden.
- Des Weiteren enthalten die angereicherten Hierarchien teilweise Merkmale, die in einer falschen Beziehung zueinander stehen, z.B. induzieren die Merkmale **Akku** und **Akku-Ladegerät** eine Ober-Untermerkmalsbeziehung: **Akku** kann als Obermerkmal von **Ladegerät** betrachtet werden. Außerdem konnte beobachtet werden, dass einige Merkmalsbeziehungen alternieren: z.B. existieren 2 Merkmale **Taste** und **Druckpunkt** in wechselnder Ober-Untermerkmalbeziehung.
- Der Algorithmus benötigt POS-Muster, um Untermerkmale in Sätzen zu finden. Für diese Arbeit wurden die verwendeten POS-Muster manuell konstruiert, aber wir planen die Konstruktion der POS-Muster weitestgehend zu automatisieren. Dazu ist eine umfangreiche Analyse eines großen Korpus notwendig.
- Die Bereinigung der erzeugten Hierarchien ist unzureichend - die relative Häufigkeit eines Merkmals reicht als Gewichtung für unsere Zwecke nicht aus. Aus diesem Grund möchten wir mit anderen Gewichtungsmaßen experimentieren.
- Die Experimente in dieser Arbeit sind sehr einfach gestaltet. Eine sinnvolle Evaluation ist (z. Zt.) nicht möglich, da (unseres Wissens nach) kein geeigneter Testkorpus mit annotierten Merkmalshierarchien existiert. Die Konstruktion eines derartigen Korpus ist geplant.
- Des Weiteren sind weitere Experimente geplant, um den Effekt der initialen Merkmalshierarchie auf den Algorithmus zu evaluieren. Diese Versuchsreihe umfasst Experimente mit mehrstufigen, initialen Merkmalshierarchien, die sowohl manuell, als auch automatisch erzeugt wurden.
- Abschließend planen wir die Verwendung unseres Algorithmus zur Extraktion von Produktmerkmalen in einem Gesamtsystem zur automatischen Zusammenfassung und Analyse von Produktrezensionen einzusetzen.

## 6. REFERENZEN

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 45–54, New York, NY, USA, 2012. ACM.

[2] F. L. Cruz, J. A. Troyano, F. Enríquez, F. J. Ortega, and C. G. Vallejo. A knowledge-rich approach to feature-based opinion extraction from product reviews. In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, SMUC '10, pages 13–20, New York, NY, USA, 2010. ACM.

[3] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.

[4] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[5] X. Meng and H. Wang. Mining user reviews: From specification to summarization. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, pages 177–180, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[6] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. In N. C. C. Chair), K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, J. Odijk, and S. Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[7] T. Scholz and S. Conrad. Extraction of statements in news for a media response analysis. In *Proc. of the 18th Intl. conf. on Applications of Natural Language Processing to Information Systems 2013 (NLDB 2013)*, 2013. (to appear).

[8] K. Zhang, R. Narayanan, and A. Choudhary. Voice of the customers: Mining online customer reviews for product feature-based ranking. In *Proceedings of the 3rd conference on Online social networks*, WOSN'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

# Ein Verfahren zur Beschleunigung eines neuronalen Netzes für die Verwendung im Image Retrieval

Daniel Braun
Heinrich-Heine-Universität
Institut für Informatik
Universitätsstr. 1
D-40225 Düsseldorf, Germany
braun@cs.uni-duesseldorf.de

## ABSTRACT

Künstliche neuronale Netze haben sich für die Mustererkennung als geeignetes Mittel erwiesen. Deshalb sollen verschiedene neuronale Netze verwendet werden, um die für ein bestimmtes Objekt wichtigen Merkmale zu identifizieren. Dafür werden die vorhandenen Merkmale als erstes durch ein Art2-a System kategorisiert. Damit die Kategorien verschiedener Objekte sich möglichst wenig überschneiden, muss bei deren Berechnung eine hohe Genauigkeit erzielt werden. Dabei zeigte sich, dass das Art2 System, wie auch die Art2-a Variante, bei steigender Anzahl an Kategorien schnell zu langsam wird, um es im Live-Betrieb verwenden zu können. Deshalb wird in dieser Arbeit eine Optimierung des Systems vorgestellt, welche durch Abschätzung des von dem Art2-a System benutzen Winkels die Anzahl der möglichen Kategorien für einen Eingabevektor stark einschränkt. Des Weiteren wird eine darauf aufbauende Indexierung der Knoten angegeben, die potentiell den Speicherbedarf für die zu überprüfenden Vektoren reduzieren kann. Wie sich in den durchgeführten Tests zeigte, kann die vorgestellte Abschätzung die Bearbeitungszeit für kleine Clusterradien stark reduzieren.

## Kategorien

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Neural Network*

## Schlüsselwörter

Neuronale Netze, Clustering, Image Retrieval

## 1. EINLEITUNG

Trainiert man ein Retrieval System mit einem festen Korpus und wendet die berechneten Daten danach unverändert an, so spielt die Berechnungsdauer für einen Klassifikator eine untergeordnete Rolle, da man die Berechnung vor der eigentlichen Anwendung ausführt. Will man allerdings auch während der Nutzung des Systems weiter lernen, so sollten die benötigten Rechnungen möglichst wenig Zeit verbrauchen, da der Nutzer ansonsten entweder auf die Berechnung warten muss oder das Ergebnis, dass ihm ausgegeben wird, berücksichtigt nicht die durch ihn hinzugefügten Daten.

Für ein fortlaufendes Lernen bieten sich künstliche neuronale Netze an, da sie so ausgelegt sind, dass jeder neue Input eine Veränderung des Gedächtnisses des Netzes nach sich ziehen kann. Solche Netze erfreuen sich, bedingt durch die sich in den letzten Jahren häufenden erfolgreichen Anwendungen - zum Beispiel in der Mustererkennung - einer steigenden Beliebtheit in verschiedensten Einsatzgebieten, wie zum Beispiel auch im Image Retrieval.

Der geplante Systemaufbau sieht dabei wie folgt aus: die Merkmalsvektoren eines Bildes werden nacheinander einer Clustereinheit übergeben, welche die Merkmalsvektoren clustert und die Kategorien der in dem Bild vorkommenden Merkmale berechnet. Das Clustering der Clustereinheit passiert dabei fortlaufend. Das bedeutet, dass die einmal berechneten Cluster für alle weiteren Bilder verwendet werden. Danach werden die für das Bild gefundenen Kategorien von Merkmalen an die Analyseeinheit übergeben, in der versucht wird, die für ein bestimmtes Objekt wichtigen Kategorien zu identifizieren. Die dort gefundenen Kategorien werden dann für die Suche dieser Objekte in anderen Bildern verwendet. Das Ziel ist es dabei, die Analyseeinheit so zu gestalten, dass sie nach einem initialen Training weiter lernt und so neue Merkmale eines Objektes identifizieren soll.

Für die Analyseeinheit ist die Verwendung verschiedener neuronaler Netze geplant. Da sie aber für die vorgenommenen Optimierungen irrelevant ist, wird im Folgenden nicht weiter auf sie eingegangen.

Von dem Clusteringverfahren für die Clustereinheit werden dabei die folgenden Punkte gefordert:

- Das Clustering soll nicht überwacht funktionieren. Das bedeutet, dass es keine Zielvorgabe für die Anzahl der Cluster geben soll. Das System soll also auch bei einem bestehenden Clustering für einen neuen Eingabevektor erkennen, ob er einem Cluster zugewiesen werden kann oder ob ein neuer Cluster erstellt werden muss.

- Die Ausdehnung der Cluster soll begrenzt sein. Das soll dazu führen, dass gefundene Merkmalskategorien mit höherer Wahrscheinlichkeit zu bestimmten Objek-

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

ten gehören und nicht die Vektoren anderer Objekte die Kategorie verschmutzen.

- Das Clustering Verfahren sollte auch bei einer hohen Anzahl an Clustern, die aus der gewünschten hohen Genauigkeit der einzelnen Cluster resultiert, schnell berechnet werden können.

In dieser Arbeit wird ein Adaptive Resonance Theory Netz [5] verwendet, genauer ein Art2 Netz [1], da es die beiden ersten Bedingungen erfüllt. Denn dieses neuronale Netz führt ein nicht überwachtes Clustering aus, wobei es mit jedem Eingabevektor weiter lernt und gegebenenfalls neue Cluster erschafft. Der genaue Aufbau dieses Systems wird in Kapitel 3 genauer dargestellt.

Zur Beschreibung des Bildes dienen SIFT Deskriptoren [9, 10], welche mit 128 Dimensionen einen sehr großen Raum für mögliche Kategorien aufspannen. Dadurch wächst die Knotenanzahl innerhalb des Art2 Netzes rapide an, was zu einer Verlangsamung des Netzes führt. Deshalb wird die Art2-a Variante [2] verwendet, welche das Verhalten des Art2 Systems approximiert. Dieses System hat die Vorteile, dass es zum Einen im Vergleich zu Art2 um mehrere Größenordnungen schneller ist und sich zum Anderen gleichzeitig auch noch größtenteils parallelisieren lässt, wodurch ein weiterer Geschwindigkeitsgewinn erzielt werden kann.

Dennoch zeigt sich, dass durch die hohe Dimension des Vektors, die für die Berechnung der Kategorie benötigten Skalarprodukte, unter Berücksichtigung der hohen Anzahl an Knoten, weiterhin sehr rechenintensiv sind. Dadurch steigt auch bei starker Parallelisierung, sofern die maximale Anzahl paralleler Prozesse begrenzt ist, die Zeit für die Bearbeitung eines neuen Vektors mit fortlaufendem Training kontinuierlich an. Aus diesem Grund wird in Kapitel 4 eine Erweiterung des Systems vorgestellt, die die Menge der Kandidaten möglicher Gewinnerknoten schon vor der teuren Berechnung des Skalarproduktes verkleinert.

Der weitere Aufbau dieser Arbeit sieht dabei wie folgt aus: in dem folgenden Kapitel 2 werden einige ausgewählte Ansätze aus der Literatur genannt, in denen neuronale Netze für das Image Retrieval verwendet werden. Um die Plausibilität der Erweiterung zu verstehen, werden dann in Kapitel 3 die dafür benötigten Mechanismen und Formeln eines Art2-a Systems vorgestellt. Kapitel 4 fokussiert sich danach auf die vorgeschlagene Erweiterung des bekannten Systems. In Kapitel 5 wird die Erweiterung evaluiert, um danach in dem folgenden Kapitel eine Zusammenfassung des Gezeigten sowie einen Ausblick zu geben.

## 2. VERWANDTE ARBEITEN

In diesem Kapitel werden einige Ansätze aus der Literatur vorgestellt, in denen neuronale Netze für verschiedene Aufgabenstellungen im Image Retrieval verwendet werden. Darunter fallen Themengebiete wie Clustering und Klassifikation von Bildern und ihren Merkmalsvektoren.

Ein bekanntes Beispiel für die Verwendung von neuronalen Netzen im Image Retrieval ist das PicSOM Framework, welches in [8] vorgestellt wird. Dort werden TS-SOMs (Tree Structured Self Orienting Maps) für die Bildsuche verwendet. Ein Bild wird dabei durch einen Merkmalsvektor dargestellt. Diese Vektoren werden dann dem neuronalen Netz präsentiert, welches sie dann der Baumstruktur hinzufügt, so dass im Idealfall am Ende jedes Bild in der Baumstruktur repräsentiert wird. Bei der Suche wird der Baum dann

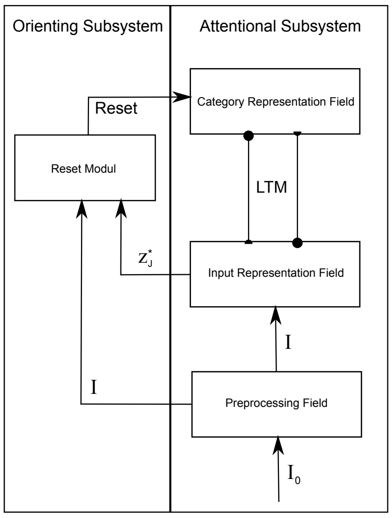


**Abbildung 1: Skizze eines Art2-a Systems**

durchlaufen und der ähnlichste Knoten als Antwort gewählt. Das Framework verwendet dabei das Feedback des Nutzers, wodurch nach jeder Iteration das Ergebnis verfeinert wird. Das neuronale Netz dient hier somit als Klassifikator.

[12] benutzt das Fuzzy Art neuronale Netz, um die Merkmalsvektoren zu klassifizieren. Sie schlagen dabei eine zweite Bewertungsphase vor, die dazu dient, das Netz an ein erwartetes Ergebnis anzupassen, das System damit zu überwachen und die Resultate des Netzes zu präzisieren.

In [6] wird ein Radial Basis Funktion Netzwerk (RBF) als Klassifikator verwendet. Eins ihrer Verfahren lässt dabei den Nutzer einige Bilder nach der Nähe zu ihrem Suchziel bewerten, um diese Bewertung dann für das Training ihrer Netzwerke zu verwenden. Danach nutzen sie die so trainierten neuronalen Netze zur Bewertung aller Bilder der Datenbank.

Auch [11] nutzt ein Radial Basis Funktion Netz zur Suche nach Bildern und trainiert dieses mit der vom Nutzer angegebenen Relevanz des Bildes, wobei das neuronale Netz nach jeder Iteration aus Bewertung und Suche weiter trainiert wird.

In [3] wird ein Multiple Instance Netzwerk verwendet. Das bedeutet, dass für jede mögliche Klasse von Bildern ein eigenes neuronales Netz erstellt wird. Danach wird ein Eingabebild jedem dieser Subnetze präsentiert und gegebenenfalls der dazugehörigen Klasse zugeordnet.

## 3. ART2-A BESCHREIBUNG

In diesem Kapitel werden die benötigten Mechanismen eines Art2-a Systems vorgestellt. Für das Verständnis sind dabei nicht alle Funktionen des Systems nötig, weshalb zum Beispiel auf die nähere Beschreibung der für das Lernen benötigten Formeln und des Preprocessing Fields verzichtet wird. Für weiterführende Informationen über diese beiden Punkte sowie generell über das Art2-a System sei deshalb auf [1, 2] verwiesen.

Wie in Bild 1 zu sehen ist, besteht das System aus zwei Subsystemen: einem Attentional Subsystem, in dem die Bearbeitung und Zuordnung eines an den Eingang angelegten Vektors ausgeführt wird, sowie einem Orienting Subsystem, welches die Ähnlichkeit des Eingabevektors mit der vorher gewählten Gewinnerkategorie berechnet und diese bei zu geringer Nähe zurücksetzt.

Innerhalb des Category Representation Field $F_2$ liegen die Knoten die für die einzelnen Vektorkategorien stehen. Dabei wird die Beschreibung der Kategorie in der Long Term Memory (LTM) gespeichert, die das Feld $F_2$ mit dem Input Representation Field $F_1$ in beide Richtungen verbindet.

Nach [2] gilt für den Aktivitätswert $T$ von Knoten $J$ in dem Feld $F_2$:

$$T_J = \begin{cases} \alpha \cdot \sum_{i=1}^{n} I_i, & \text{wenn } J \text{ nicht gebunden ist,} \\ I \cdot z_J^*, & \text{wenn } J \text{ gebunden ist.} \end{cases}$$

$I_i$ steht dabei für den durch das Preprocessing Field $F_0$ berechneten Input in das Feld $F_1$ und $\alpha$ ist ein wählbarer Parameter, der klein genug ist, so dass die Aktivität eines ungebundenen Knotens für bestimmte Eingangsvektoren nicht immer größer ist als alle Aktivitätswerte der gebundenen Knoten. Hierbei gilt ein Knoten als gebunden, wenn ihm mindestens ein Vektor zugeordnet wurde.

Da der Aktivitätswert für alle nicht gebundenen Knoten konstant ist und deshalb nur einmal berechnet werden muss, ist dieser Fall für eine Effizienzsteigerung von untergeordnetem Interesse und wird deshalb im Folgenden nicht weiter betrachtet.

Wie in [2] gezeigt wird, sind sowohl $I$ als auch $z_J^*$, durch die Anwendung der euklidischen Normalisierung, Einheitsvektoren, weshalb folglich

$$\|I\| = \|z_J^*\| = 1 \tag{1}$$

gilt. Deshalb folgt für die Aktivitätswerte der gebunden Kategorieknoten:

$$\begin{aligned} T_J &= I \cdot z_J^* \\ &= \|I\| \cdot \|z_J^*\| \cdot \cos\theta \\ &= \cos\theta \end{aligned} \tag{2}$$

Die Aktivität eines Knotens entspricht damit dem Winkel zwischen dem Eingangsvektor $I$ und dem LTM-Vektor $z_J^*$. Damit der Knoten mit dem Index $J$ gewählt wird, muss

$$T_J = \max_j \{T_j\}$$

gelten, sprich der Knoten mit der maximalen Aktivität wird als mögliche Kategorie gewählt. Dabei wird bei Gleichheit mehrerer Werte der zu erst gefundene Knoten genommen. Die maximale Distanz, die das Resetmodul akzeptiert, wird durch den Schwellwert $\rho$, im Folgenden Vigilance Parameter genannt, bestimmt, mit dem die, für die Art2-a Variante benötigte, Schwelle $\rho^*$ wie folgt berechnet wird:

$$\rho^* = \frac{\rho^2(1+\sigma)^2 - (1+\sigma^2)}{2\sigma}$$

mit

$$\sigma \equiv \frac{cd}{1-d} \tag{3}$$

und c und d als frei wählbare Parameter des Systems, die der Beschränkung

$$\frac{cd}{1-d} \leq 1$$

unterliegen. Damit wird der Knoten $J$ genau dann abgelehnt, wenn

$$T_J < \rho^* \tag{4}$$

gilt. Ist das der Fall, wird ein neuer Knoten aktiviert und somit eine neue Kategorie erstellt. Mit 2 und 4 folgt damit, dass ein Knoten nur dann ausgewählt werden kann, wenn für den Winkel $\theta$ zwischen dem Eingabevektor $I$ und dem gespeicherten LTM-Vektor $z_J^*$

$$\cos\theta \geq \rho^* \tag{5}$$

gilt. Da die einzelnen Rechnungen, die von dem System ausgeführt werden müssen, unabhängig sind, ist dieses System hochgradig parallelisierbar, weshalb alleine durch Ausnutzung dieser Tatsache die Berechnungszeit stark gesenkt werden kann. Mit steigender Knotenanzahl lässt sich das System dennoch weiter optimieren, wie in dem folgenden Kapitel gezeigt werden soll.

Das Art2-a System hat dabei allerdings einen Nachteil, denn bedingt durch die Nutzung des Kosinus des Winkels zwischen zwei Vektoren werden Vektoren, die linear abhängig sind, in dieselbe Kategorie gelegt. Dieses Verhalten ist für die geforderte Genauigkeit bei den Kategorien unerwünscht. Dennoch lässt sich dieses Problem leicht durch die Erhebung weiterer Daten, wie zum Beispiel den Clustermittelpunkt, lösen, weshalb im Folgenden nicht weiter darauf eingegangen wird.

## 4. VORGENOMMENE OPTIMIERUNG

Dieses Kapitel dient der Beschreibung der verwendeten Abschätzung und ihrer Implementierung in das Art2-a System. Abschließend wird dann noch auf eine weitere Verbesserung, die sich durch diese Implementierung ergibt, eingegangen. Der Aufbau des Abschnitts ist dabei wie folgt: in Unterabschnitt 1 wird das Verfahren zur Abschätzung des Winkels vorgestellt. In dem folgenden Unterabschnitt 2 wird dann gezeigt, wie man diese Abschätzung in das Art2-a System integrieren kann. In dem letzten Unterabschnitt folgt dann eine Vorstellung der Abschätzung als Index für die Knoten.

### 4.1 Abschätzung des Winkels

In [7] wird eine Methode zur Schätzung der Distanz zwischen einem Anfragevektor und einem Datenvektor beschrieben. Im Folgenden wird beschrieben, wie man Teile dieses Verfahrens nutzen kann, um damit die Menge möglicher Knoten schon vor der Berechnung des Aktivitätswertes $T_J$ zu verringern. Das Ziel ist es, die teure Berechnung des Skalarproduktes zwischen $I$ und $z_J^*$ möglichst selten auszuführen und gleichzeitig möglichst wenig Daten im Speicher vorrätig halten zu müssen. Dafür wird der unbekannte Winkel $\theta$ zwischen den beiden Vektoren $P$ und $Q$ durch die bekannten Winkel $\alpha$ und $\beta$ zwischen beiden Vektoren und einer festen Achse $T$ wie folgt approximiert:

$$\begin{aligned} \cos\theta &\leq \cos(|\alpha - \beta|) \\ &= \cos(\alpha - \beta) \\ &= \cos\alpha\cos\beta + \sin\alpha\sin\beta \\ &= \cos\alpha\cos\beta + \sqrt{1-\cos\alpha^2}\sqrt{1-\cos\beta^2} \end{aligned} \tag{6}$$

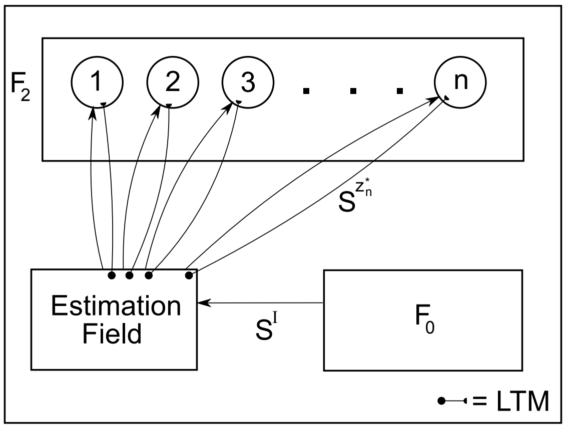


**Abbildung 2: Erweiterung des Art2 Systems mit einem neuen Feld für die Abschätzung des Winkels**

Als Achse $T$ wird hierbei ein n-dimensionaler mit Einsen gefüllter Vektor verwendet, wodurch für die L2-Norm des Achsenvektors $\|T\| = \sqrt{n}$ folgt. Eingesetzt in die Formel

$$\cos\theta = \frac{\langle P, Q\rangle}{\|P\|\|Q\|}$$

ergibt sich damit, unter Ausnutzung von (1), für das System mit den Vektoren $I$ und $z^*_J$:

$$cos\alpha = \frac{\sum_{i=1}^{n} I_i}{\sqrt{n}} \text{ und } cos\beta = \frac{\sum_{i=1}^{n} z^*_{Ji}}{\sqrt{n}}$$

Mit $S^I$ und $S^{z^*_J}$ als jeweilige Summe der Vektorwerte reduziert sich, unter Verwendung der Formel (6), die Abschätzung des Kosinus vom Winkel $\theta$ auf

$$\begin{aligned}\cos\theta &\leq \frac{S^I * S^{z^*_J}}{n} + \sqrt{(1 - \frac{S^{I^2}}{n})(1 - \frac{S^{z^*_J{}^2}}{n})} \\ &= \frac{S^I * S^{z^*_J} + \sqrt{(n - S^{I^2})(n - S^{z^*_J{}^2})}}{n}\end{aligned}$$

Diese Abschätzung ermöglicht es nun, die Menge der Kandidaten möglicher Knoten für einen Eingabevektor $I$ vorzeitig zu reduzieren, indem man ausnutzt, dass der wirkliche Winkel zwischen Eingabevektor und in der LTM gespeichertem Vektor maximal genauso groß ist, wie der mit der gezeigten Formel geschätzte Winkel zwischen beiden Vektoren. Damit ist diese Abschätzung des wirklichen Winkels $\theta$ verlustfrei, denn es können keine Knoten mit einem tatsächlich größeren Kosinuswert des Winkels aus der Menge an Kandidaten entfernt werden. Daraus resultiert, dass ein Knoten nur dann weiter betrachtet werden muss, wenn die Bedingung

$$\frac{S^I * S^{z^*_J} + \sqrt{(n - S^{I^2})(n - S^{z^*_J{}^2})}}{n} \geq \rho^* \qquad (7)$$

erfüllt wird.

## 4.2 Erweiterung des Systems

Damit die Bedingung (7) ausgenutzt werden kann, wird das Art2 System um ein weiteres Feld, im Folgenden Estimation Field genannt, erweitert. Dieses Feld soll als Schnittstelle zwischen $F_0$ und $F_2$ dienen und die Abschätzung des Winkels zwischen dem Eingabevektor und dem gespeicherten LTM Vektor vornehmen. Dazu wird dem Feld, wie in Abbildung 2 gezeigt wird, von dem Feld $F_0$ die Summe $S^I$ übergeben. Innerhalb des Feldes gibt es dann für jeden Knoten $J$ im Feld $F_2$ eine zugehörige Estimation Unit $J'$. In der Verbindung von jedem Knoten $J$ zu der ihm zugehörigen Estimation Unit $J'$ wird die Summe der Werte des jeweiligen LTM Vektors $S^{z^*_J}$ als LTM gespeichert. Die Estimation Unit berechnet dann die Funktion

$$f(J) = \frac{S^I * S^{z^*_J} + \sqrt{(n - S^{I^2})(n - S^{z^*_J{}^2})}}{n}$$

für den ihr zugehörigen Knoten $J$. Abschließend wird als Aktivierungsfunktion, für die Berechnung der Ausgabe $o_{J'}$ der Estimation Unit $J'$, die folgende Schwellenfunktion verwendet:

$$o_{J'} = \begin{cases} 1, & \text{wenn } f(J) \geq \rho^* \\ 0, & \text{sonst} \end{cases} \qquad (8)$$

Damit ergibt sich für die Aktivitätsberechnung jedes Knotens des $F_2$ Feldes die angepasste Formel

$$T_J = \begin{cases} \alpha * \sum_i I_i, & \text{wenn } J \text{ nicht gebunden ist,} \\ I * z^*_J, & \text{wenn } J \text{ gebunden ist und } o_{J'} = 1 \text{ gilt,} \\ 0 & \text{wenn } o_{J'} = 0 \text{ gilt.} \end{cases} \qquad (9)$$

mit $o_{J'}$ als Ausgabe des Estimation Field zu Knoten $J$.

## 4.3 Verwendung als Index

Durch die gezeigte Kosinusschätzung werden unnötige Skalarprodukte vermieden und somit das System beschleunigt. Allerdings kann es bei weiterhin wachsender Anzahl der Knoten, zum Beispiel weil der Arbeitsspeicher nicht ausreicht, nötig werden, nicht mehr alle LTM Vektoren im Speicher zu halten, sondern nur ein Set möglicher Kandidaten zu laden und diese dann gezielt zu analysieren. In dem folgenden Abschnitt wird gezeigt, wie die Abschätzung sinnvoll als Index für die Knoten verwendet werden kann.

Für die Indexierung wird als Indexstruktur ein $B^+$-Baum mit der Summe der Werte jedes LTM Vektors $S^{z^*_J}$ und der ID $J$ des Knotens als zusammengesetzten Schlüssel verwendet. Für die Sortierreihenfolge gilt: zuerst wird nach $S^{z^*_J}$ sortiert und dann nach $J$. Dadurch bleibt der $B^+$-Baum für partielle Bereichsanfragen nach dem Wert der Summe optimiert. Damit das funktioniert muss allerdings die Suche so angepasst werden, dass sie bei einer partiellen Bereichsanfrage für die ID den kleinstmöglichen Wert einsetzt und dann bei der Ankunft in einem Blatt der Sortierung bis zum ersten Vorkommen, auch über Blattgrenzen hinweg, der gesuchten Summe folgt.

Dieser Index wird nun verwendet, um die Menge der Kandidaten einzuschränken, ohne, wie in der vorher vorgestellten Optimierung durch die Estimation Unit, alle Knoten durchlaufen zu müssen. Anschaulich bedeutet das, dass das Art2-a System nur noch die der Menge an Kandidaten für den Eingabevektor $I$ angehörenden Knoten sehen soll und somit nur in diesen den Gewinnerknoten suchen muss. Für diesen Zweck müssen mögliche Wertebereiche der gespeicherten $S^{z^*_J}$ für einen beliebigen Eingabevektor festgelegt werden. Dies geschieht wieder mit Hilfe der Bedingung (7):

$$\frac{S^I \cdot S^{z^*_J} + \sqrt{(n - {S^I}^2)(n - {S^{z^*_J}}^2)}}{n} \geq \rho$$
$$\sqrt{(n - {S^I}^2)(n - {S^{z^*_J}}^2)} \geq \rho \cdot n - S^I \cdot S^{z^*_J}$$

Für $\rho \cdot n - S^I \cdot S^{z^*_J} < 0$ ist diese Ungleichung offensichtlich immer erfüllt, da die Quadratwurzel auf der linken Seite immer positiv ist. Damit ergibt sich die erste Bedingung:

$$S^{z^*_J} > \frac{\rho \cdot n}{S^I} \qquad (10)$$

Nun wird im Folgenden noch der Fall $\rho \cdot n \geq S^I \cdot S^{z^*_J}$ weiter betrachtet:

$$\sqrt{(n - {S^I}^2)(n - {S^{z^*_J}}^2)} \geq \rho \cdot n - S^I \cdot S^{z^*_J}$$
$$n \cdot (1 - \rho^2) - {S^I}^2 \geq {S^{z^*_J}}^2 - 2\rho S^I S^{z^*_J}$$
$$(n - {S^I}^2)(1 - \rho^2) \geq (S^{z^*_J} - \rho \cdot S^I)^2$$

Damit ergibt sich:

$$\sqrt{(n - {S^I}^2)(1 - \rho^2)} \geq S^{z^*_J} - \rho \cdot S^I \geq -\sqrt{(n - {S^I}^2)(1 - \rho^2)} \qquad (11)$$

Mit den Bedingungen (10) und (11) können nun die partiellen Bereichsanfragen an den Index für einen beliebigen Eingabevektor $I$ wie folgt formuliert werden:

$$r_1 = [\rho S^I - \sqrt{(n - {S^I}^2)(1 - \rho^2)}, \rho S^I + \sqrt{(n - {S^I}^2)(1 - \rho^2)}]$$
$$r_2 = [\frac{\rho \cdot n}{S^I}, \infty]$$

Da für diese Bereichsanfragen die Bedingung (7) genutzt wird und somit alle geschätzten Winkel größer als $\rho^*$ sind, hat bei der Verwendung des Indexes das Estimation Field keinen Effekt mehr.

## 5. EVALUATION

In diesem Kapitel wird die gezeigte Abschätzung evaluiert. Der vorgeschlagene Index wird dabei aber noch nicht berücksichtigt.

### 5.1 Versuchsaufbau

Für die Evaluierung des gezeigten Ansatzes wurde ein Computer mit einem Intel Core 2 Duo E8400 3 GHz als Prozesser und 4 GB RAM benutzt.

Als Datensatz wurden Bilder von Flugzeugen aus dem Caltech 101 Datensatz [4] verwendet. Diese Bilder zeigen verschiedene Flugzeuge auf dem Boden beziehungsweise in der Luft. Für den Geschwindigkeitstest wurden 20 Bilder aus dem Pool ausgewählt und nacheinander dem neuronalen Netz präsentiert. Im Schnitt produzieren die benutzten Bilder dabei 4871 SIFT Feature Vektoren pro Bild.

Bedingt dadurch, dass die Ansätze verlustfrei sind, wird nur die Rechenzeit der verschiedenen Verfahren gegenüber

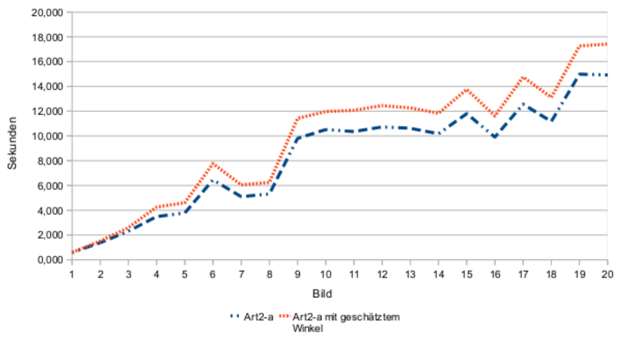


**Abbildung 3: Zeitmessung für $\rho = 0.95$**

gestellt, denn es sind keine Einbußen in der Güte des Ergebnisses zu erwarten. Außerdem wird die mögliche Parallelisierung nicht weiter betrachtet, da bei einer begrenzten Anzahl von parallelen Prozessen die Anzahl der Knoten pro Prozess mit jedem weiteren Bild steigt und den Prozess so verlangsamt. Als mögliche Werte für den Schwellwert $\rho$ wurden die zwei, in der Literatur öfter genannten, Werte 0.95 und 0.98 sowie der Wert 0.999 verwendet. Für die restlichen benötigten Parameter aus Formel (3) und (9) gilt: $c = 0.1$, $d = 0.9$ und $\alpha = 0$

### 5.2 Ergebnisse

Für die kleineren Vigilance Werte von 0.95 und 0.98 zeigt sich, wie in den Abbildungen 3 und 4 zu sehen ist, dass die Abschätzung hier kaum einen Vorteil bringt. Sie ist sogar langsamer als das originale System. Dies liegt daran, dass die Abschätzung durch Verwendung nur eines Wertes, nämlich der Summe, viel zu ungenau ist, um bei diesem Vigilance Wert genug Knoten herauszufiltern, da fast alle Knoten über der Grenze liegen. Da deshalb kaum Zeit gewonnen wird, wird das System durch den betriebenen Mehraufwand langsamer. Mit steigendem Vigilance Parameter nimmt auch der Nutzen des Verfahrens zu, da die Anzahl der entfernten Knoten signifikant zunimmt. Dies sieht man deutlich in Abbildung 5, in der die benötigte Rechenzeit für einen Wert von 0.999 dargestellt ist. In diesem Fall filtert die gezeigte Abschätzung sehr viele Knoten heraus, weshalb der Zeitgewinn den Zeitverlust durch den größeren Aufwand weit übersteigt. Da aber möglichst genaue Kategorien erwünscht sind, ist ein hoher Vigilance Parameter die richtige Wahl. Deshalb kann das gezeigte Verfahren für das angestrebte System adaptiert werden.

## 6. RESÜMEE UND AUSBLICK

In dieser Arbeit wurde eine Optimierung des Art2-a Systems vorgestellt, die durch Abschätzung des Winkels zwischen Eingabevektor und gespeichertem Vektor die Menge an zu überprüfenden Kandidaten für hohe Vigilance Werte stark reduzieren kann. Des Weiteren wurde ein Ansatz zur Indexierung der Knoten basierend auf der für die Abschätzung nötigen Summe vorgestellt. Auch wenn eine abschließende Analyse des gezeigten noch offen ist, so scheint dieser Ansatz dennoch erfolgversprechend für die erwünschten hohen Vigilance Werte.

Aufbauend auf dem gezeigten wird unsere weitere Forschung die folgenden Punkte beinhalten:

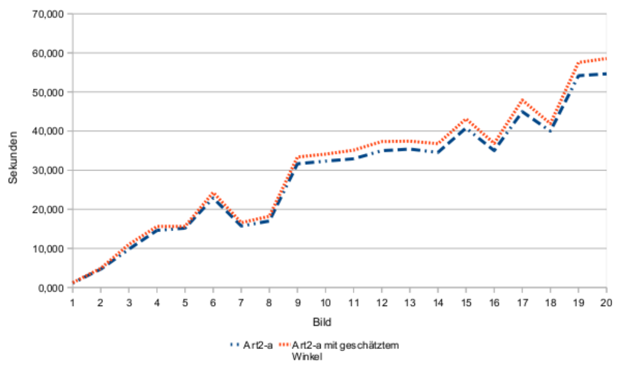


**Abbildung 4: Zeitmessung für** $\rho = 0.98$

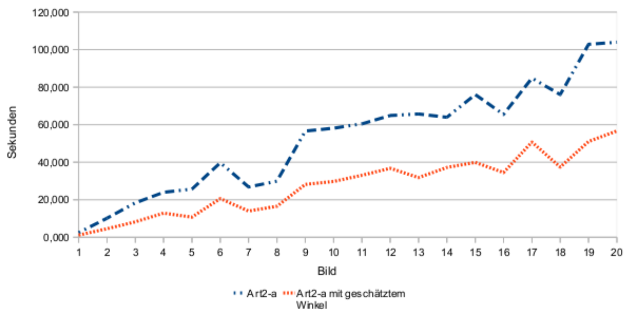


**Abbildung 5: Zeitmessung für** $\rho = 0.999$

- Es wird geprüft, ob die Abschätzung durch die Hinzunahme weiterer Daten verbessert werden kann und somit eine weitere Beschleunigung erzielt wird. Dafür kann man, um das Problem der zu geringen Präzision der Abschätzung bei kleinerem Vigilance Parameter zu umgehen, die Vektoren teilen und die Abschätzung wie in [7] aus den Teilsegmenten der Vektoren zusammensetzen. Dafür bräuchte man aber auch die Summe der Quadrate, da die Teilsegmente der Vektoren keine Einheitsvektoren mehr sind. Deshalb wird es sich noch zeigen, ob der Gewinn an Präzision durch eine Aufteilung den größeren Aufwand durch Berechnung und Speicherung weiterer Werte rechtfertigt. Des Weiteren soll damit überprüft werden, ob die Abschätzung auch für kleinere Vigilance Werte verwendet werden kann.
- Es wird überprüft, wie groß die Auswirkungen der vorgestellten Verfahren bei einer parallelen Berechnung des Gewinnerknotens sind. Des Weiteren wird das Verfahren auf größeren Datenmengen getestet, um zu überprüfen, ob eine weitere Beschleunigung nötig ist, damit man das Verfahren im Live Betrieb verwenden kann.
- Die Verwendung der Abschätzung zum Indexieren soll getestet und mit anderen Indexierungsverfahren verglichen werden, um ihren Nutzen besser bewerten zu können. Aber vor allem ihre Auswirkungen auf das Art2-a System im parallelisierten Betrieb sind noch offen und werden überprüft.
- Danach werden wir die Analyseeinheit konzipieren. Dafür wird als erstes überprüft, welche Daten man für ein fortlaufendes Lernen braucht, um einem Objekt keine falschen neuen Kategorien zuzuweisen oder richtige Kategorien zu entfernen. Danach soll ein geeignetes neuronales Netz aufgebaut werden, um damit die Zuordnung der Kategorien zu den Objekten durchführen zu können. Das Netz muss dann an die vorher erhobenen Daten angepasst werden, um die Präzision des Netzes zu erhöhen. Abschließend wird das Verfahren dann gegen andere populäre Verfahren getestet.

## 7. REFERENZEN

[1] G. A. Carpenter and S. Grossberg. Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4930, 1987.

[2] G. A. Carpenter, S. Grossberg, and D. B. Rosen. Art 2-a: an adaptive resonance algorithm for rapid category learning and recognition. In *Neural Networks*, volume 4, pages 493–504, 1991.

[3] S.-C. Chuang, Y.-Y. Xu, H. C. Fu, and H.-C. Huang. A multiple-instance neural networks based image content retrieval system. In *Proceedings of the First International Conference on Innovative Computing, Information and Control*, volume 2, pages 412–415, 2006.

[4] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories, 2004. CVPR 2004, Workshop on Generative-Model Based Vision.

[5] S. Grossberg. Adaptive pattern classification and universal recording: II. Feedback, expectation, olfaction, illusions. *Biological Cybernetics*, 23:187–202, 1976.

[6] B. Jyothi and D. Shanker. Neural network approach for image retrieval based on preference elicitation. *International Journal on Computer Science and Engineering*, 2(4):934–941, 2010.

[7] Y. Kim, C.-W. Chung, S.-L. Lee, and D.-H. Kim. Distance approximation techniques to reduce the dimensionality for multimedia databases. *Knowledge and Information Systems*, 2010.

[8] L. Koskela, , J. T. Laaksonen, J. M. Koskela, and E. Oja. Picsom a framework for content-based image database retrieval using self-organizing maps. In *In 11th Scandinavian Conference on Image Analysis*, pages 151–156, 1999.

[9] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, 1999.

[10] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[11] K. N. S., Čabarkapa Slobodan K., Z. G. J., and R. B. D. Implementation of neural network in cbir systems with relevance feedback. *Journal of Automatic Control*, 16:41–45, 2006.

[12] H.-J. Wang and C.-Y. Chang. Semantic real-world image classification for image retrieval with fuzzy-art neural network. *Neural Computing and Applications*, 21(8):2137–2151, 2012.

# Auffinden von Spaltenkorrelationen mithilfe proaktiver und reaktiver Verfahren

Katharina Büchse
Friedrich-Schiller-Universität
Institut für Informatik
Ernst-Abbe-Platz 2
07743 Jena
katharina.buechse@uni-jena.de

## KURZFASSUNG

Zur Verbesserung von Statistikdaten in relativen Datenbanksystemen werden seit einigen Jahren Verfahren für das Finden von Korrelationen zwischen zwei oder mehr Spalten entwickelt. Dieses Wissen über Korrelationen ist notwendig, weil der Optimizer des Datenbankmanagementsystems (DBMS) bei der Anfrageplanerstellung sonst von Unabhängigkeit der Daten ausgeht, was wiederum zu groben Fehlern bei der Kostenschätzung und somit zu schlechten Ausführungsplänen führen kann.

Die entsprechenden Verfahren gliedern sich grob in proaktive und reaktive Verfahren: Erstere liefern ein gutes Gesamtbild über sämtliche vorhandenen Daten, müssen dazu allerdings selbst regelmäßig auf die Daten zugreifen und benötigen somit Kapazität des DBMS. Letztere überwachen und analysieren hingegen die Anfrageergebnisse und liefern daher nur Korrelationsannahmen für bereits abgefragte Daten, was einerseits das bisherige Nutzerinteresse sehr gut widerspiegelt, andererseits aber bei Änderungen des Workloads versagen kann. Dafür wird einzig bei der Überwachung der Anfragen DBMS-Kapazität benötigt, es erfolgt kein eigenständiger Zugriff auf die Daten.

Im Zuge dieser Arbeit werden beide Ansätze miteinander verbunden, um ihre jeweiligen Vorteile auszunutzen. Dazu werden die sich ergebenden Herausforderungen, wie sich widersprechende Korrelationsannahmen, aufgezeigt und als Lösungsansatz u. a. der zusätzliche Einsatz von reaktiv erstellten Statistiken vorgeschlagen.

## Categories and Subject Descriptors

H.2 [**Information Systems**]: Database Management; H.2.4 [**Database Management**]: Systems—*Query processing*

## General Terms

Theory, Performance

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

## Keywords

Anfrageoptimierung, Spaltenkorrelation, Feedback

## 1. EINFÜHRUNG

Die Verwaltung großer Datenmengen benötigt zunehmend leistungsfähigere Algorithmen, da die Verbesserung der Technik (Hardware) nicht mit dem immer höheren Datenaufkommen heutiger Zeit mithalten kann. Bspw. werden wissenschaftliche Messergebnisse aufgrund besserer Messtechnik immer genauer und umfangreicher, sodass Wissenschaftler sie detaillierter, aber auch umfassender analysieren wollen und müssen, oder Online-Shops speichern sämtliche ihrer Verkaufsdaten und werten sie aus, um dem Benutzer passend zu seinen Interessen zeitnah und individuell neue Angebote machen zu können.

Zur Verwaltung dieser wie auch anderer Daten sind (im Datenbankbereich) insbesondere schlaue Optimizer gefragt, weil sie für die Erstellung der Anfragepläne (und somit für die Ausführungszeit einer jeden Anfrage) verantwortlich sind. Damit sie in ihrer Wahl nicht völlig daneben greifen, gibt es Statistiken, anhand derer sie eine ungefähre Vorstellung bekommen, wie die vorhandene Datenlandschaft aussieht. Hierbei ist insbesondere die zu erwartende Tupelanzahl von Interesse, da sie in hohem Maße die Ausführungszeit einer Anfrage beeinflusst. Je besser die Statistiken die Verteilung der Daten wiedergeben (und je aktueller sie sind), desto besser ist der resultierende Ausführungsplan. Sind die Daten unkorreliert (was leider sehr unwahrscheinlich ist), genügt es, pro zu betrachtender Spalte die Verteilung der Werte innerhalb dieser Spalte zu speichern. Treten in diesem Fall später in den Anfragen Kombinationen der Spalten auf, ergibt sich die zu erwartende Tupelanzahl mithilfe einfacher statistischer Weisheiten (durch Multiplikation der Einzelwahrscheinlichkeiten).

Leider versagen diese ab einem bestimmten Korrelationsgrad (also bei korrelierten Daten), und zwar in dem Sinne, dass die vom Optimizer berechneten Schätzwerte zu stark von der Wirklichkeit abweichen, was wiederum zu schlechten Ausführungszeiten führt. Diese ließen sich u.U. durch die Wahl eines anderen Plans, welcher unter Berücksichtigung der Korrelation vom Optimizer erstellt wurde, verringern oder sogar vermeiden.

Zur Veranschaulichung betrachten wir eine Tabelle, welche u. a. die Spalten $A$ und $B$ besitzt, und eine Anfrage, welche Teile eben dieser Spalten ausgeben soll. Desweiteren liege auf Spalte $B$ ein Index, den wir mit $I_B$ bezeichnen wollen, und es existiere ein zusammengesetzter Index $I_{A,B}$ für beide Spalten. Beide Indizes seien im DBMS mithilfe von Bäumen (bspw. B*-Bäume) implementiert, sodass wir auch (etwas informell) von „flachen“ oder „hohen“ Indizes sprechen können.

Sind beide Spalten unkorreliert, so lohnt sich in der Regel die Abfrage über $I_{A,B}$. Bei einer starken Korrelation beider Spalten dagegen könnte die alleinige Verwendung von $I_B$ vorteilhaft sein, und zwar wenn die Werte aus Spalte $A$ i.d.R. durch die Werte aus Spalte $B$ bestimmt werden (ein typisches Beispiel, welches auch in CORDS [7] anzutreffen ist, wäre eine Tabelle „Auto“ mit den Spalten $A =$ „Firma“ und $B =$ „Marke“, sodass sich für $A$ Werte wie „Opel“ oder „Mercedes“ und für $B$ Werte wie „Zafira“ oder „S-Klasse“ ergeben). Statt nun im vergleichsweise hohen Index $I_{A,B}$ erst passende $A$- und dann passende $B$-Werte zu suchen, werden sämtliche Tupel, welche die gewünschten $B$-Werte enthalten, über den flacheren Index $I_B$ geladen und überprüft, ob die jeweiligen $A$-Werte der Anfrage entsprechen (was aufgrund der Abhängigkeit der Regelfall sein sollte).

Das Wissen über Korrelationen fällt aber natürlich nicht vom Himmel, es hat seinen Preis. Jeder Datenbänkler hofft, dass seine Daten unkorreliert sind, weil sein DBMS dann weniger Metadaten (also Daten über die Daten) speichern und verwalten muss, sondern auf die bereits erwähnten statistischen Weisheiten zurückgreifen kann. Sind die Daten dagegen (stark) korreliert, lässt sich die Erkenntnis darüber nicht so einfach wie die Unabhängigkeit mit (anderen) statistischen Weisheiten verbinden und somit „abarbeiten“.

Nicht jede (eher kaum eine) Korrelation stellt eine (schwache) funktionale Abhängigkeit dar, wie es im Beispiel der Fall war, wo wir einfach sagen konnten „Aus der Marke folgt die Firma (bis auf wenige Ausnahmen)“. Oft liebäugeln bestimmte Werte der einen Spalte mit bestimmten Werten anderer Spalten, ohne sich jedoch in irgendeiner Weise auf diese Kombinationen zu beschränken. (In Stuttgart gibt es sicherlich eine Menge Porsches, aber die gibt es woanders auch.) Außerdem ändern sie möglicherweise mit der Zeit ihre Vorlieben (das Stuttgarter Porschewerk könnte bspw. nach China umziehen) oder schaffen letztere völlig ab (wer braucht schon einen Porsche? Oder überhaupt ein Auto?).

Deswegen werden für korrelierte Daten zusätzliche Statistiken benötigt, welche nicht nur die Werteverteilung einer, sondern die Werteverteilung mehrerer Spalten wiedergeben. Diese zusätzlichen Statistiken müssen natürlich irgendwo abgespeichert und, was noch viel schlimmer ist, gewartet werden. Somit ergeben sich zusätzlicher Speicherbedarf und zusätzlicher Aufwand, also viel zu viel von dem, was keiner so richtig will.

Da sich ein bisschen statistische Korrelation im Grunde überall findet, gilt es, die Korrelationen ausfindig zu machen, welche unsere statistischen Weisheiten alt aussehen lassen und dazu führen, dass das Anfrageergebnis erst nach einer gefühlten halben Ewigkeit ausgeben wird. Ob letzteres überhaupt passiert, hängt natürlich auch vom Anfrageverhalten auf die Datenbank ab. Wenn die Benutzer sich in ihren (meist mithilfe von Anwendungsprogrammen abgesetzten) SQL-Anfragen in der WHERE-Klausel jeweils auf eine Spalte beschränken und auf jedwede Verbünde (Joins) verzichten, dann ist die Welt in Ordnung. Leider lassen sich Benutzer nur ungern so stark einschränken.

Daher gibt es zwei grundsätzliche Möglichkeiten: Entweder schauen wir dem Benutzer auf die Finger und suchen in den von ihm abgefragten Daten nach Korrelationen (das entspricht einer reaktiven Vorgehensweise), oder wir „suchen uns selbst“ ein paar Daten der Datenbank „aus“, die wir untersuchen wollen (und gehen somit proaktiv vor). Beide Vorgehensweisen haben ihre Vor- und Nachteile. Während im reaktiven Fall keine Daten speziell zur Korrelationsfindung „angefasst“ werden müssen, hier aber alle Daten, die bis zu einer bestimmten Anfrage nie abgefragt wurden, als unkorreliert gelten, müssen wir für die proaktive Methode (also nur zum Feststellen, ob Korrelation vorherrscht) extra Daten lesen, sind aber für (fast) alle Eventualitäten gewappnet.

Interessanterweise kann es vorkommen, dass beide Methoden für ein und dieselbe Spaltenkombination unterschiedliche Ergebnisse liefern (der Einfachheit halber beschränken wir uns hierbei auf die möglichen Ergebnisse „korreliert“ oder „unabhängig“). Für den Fall, dass die reaktive Methode eine Spaltenkombination gar nicht betrachtet hat, sollte das klar sein. Aber nehmen wir an, dass die Kombination von beiden Methoden analysiert wurde. Da für die Analyse höchstwahrscheinlich jeweils unterschiedliche Tupel (Wertekombinationen) verwendet wurden, können sich natürlich auch die Schlüsse unterscheiden. Hier stellt sich nun die Frage, welches Ergebnis „besser“ ist. Dafür gibt es keine allgemeine Antwort, gehen wir aber von einer moderaten Änderung des Anfrageverhaltens aus, ist sicherlich das „reaktive Ergebnis“ kurzfristig entscheidender, während das „proaktive Ergebnis“ in die längerfristige Planung der Statistikerstellung mit aufgenommen werden sollte.

## 2. GRUNDLAGEN

Wie in der Einleitung bereits angedeutet, können Korrelationen einem Datenbanknutzer den Tag vermiesen. Um dies zu verhindern, wurden einige Methoden vorgeschlagen, welche sich auf verschiedene Art und Weise dieser Problematik annehmen (z. B. [7, 6]) oder sie sogar ausnutzen (z. B. [4, 8]), um noch an Performance zuzulegen. Letztere sind allerdings mit hohem Aufwand oder der Möglichkeit, fehlerhafte Anfrageergebnisse zu liefern[1], verbunden. Daher konzentrieren wir uns hier auf das Erkennen von Korrelationen allein zur Verbesserung der Statistiken und wollen hierbei zwischen proaktiven und reaktiven Verfahren unterscheiden.

### 2.1 Proaktive (datengetriebene) Verfahren

Proaktiv zu handeln bedeutet, etwas „auf Verdacht“ zu tun. Impfungen sind dafür ein gutes Beispiel – mithilfe einer Impfung ist der Körper in der Lage, Krankheitserreger zu bekämpfen, aber in vielen Fällen ist unklar, ob er diese Fähigkeit jemals benötigen wird. Da Impfungen auch mit Nebenwirkungen verbunden sein können, muss jeder für sich entscheiden, ob und wogegen er sich impfen lässt.

Auch Datenbanken können „geimpft“ werden, allerdings handelt es sich bei langen Anfrageausführungszeiten (die wir ja bekämpfen wollen) eher um Symptome (wie Bauchschmerzen oder eine laufende Nase), die natürlich unterschiedliche Ursachen haben können. Eine davon bilden ganz

[1]Da die Verfahren direkt in die Anfrageplanerstellung eingreifen und dabei auf ihr Wissen über Korrelationen aufbauen, muss, für ein korrektes Anfrageergebnis, dieses Wissen aktuell und vollständig sein.

klar Korrelationen zwischen den Daten, wobei natürlich erst ein gewisses Maß an Korrelation überhaupt als „krankhaft“ anzusehen ist. (Es benötigt ja auch eine gewisse Menge an Bakterien, damit eine Krankheit mit ihren Symptomen ausbricht.) Der grobe „Impfvorgang“ „gegen“ Korrelationen umfasst zwei Schritte:

1. Es werden Vermutungen aufgestellt, welche Spaltenkombinationen für spätere Anfragen eine Rolle spielen könnten.

2. Es wird kontrolliert, ob diese Kombinationen von Korrelation betroffen sind oder nicht.

Entscheidend dabei ist, dass die Daten bzw. ein Teil der Daten gelesen (und analysiert) werden, und zwar ohne damit konkrete Anfragen zu bedienen, sondern rein zur Ausführung des Verfahrens bzw. der „Impfung“ (in diesem Fall „gegen“ Korrelation, wobei die Korrelation natürlich nicht beseitigt wird, schließlich können wir schlecht den Datenbestand ändern, sondern die Datenbank lernt, damit umzugehen). Das Lesen und Analysieren kostet natürlich Zeit, womit klar wird, dass auch diese „Impfung“ „Nebenwirkungen“ mit sich bringt.

Eine konkrete Umsetzung haben Ilyas et al., aufbauend auf BHUNT [4], mit CORDS [7] vorgestellt. Dieses Verfahren findet Korrelationen zwischen Spaltenpaaren, die Spaltenanzahl pro Spaltenkombination wurde also auf zwei begrenzt.[2]

Es geht folgendermaßen vor: Im ersten „Impfschritt“ sucht es mithilfe des Katalogs oder mittels Stichproben nach Schlüssel-Fremdschlüssel-Beziehungen und führt somit eine Art Rückabbildung von Datenbank zu Datenmodell durch (engl. „reverse engineering“) [4]. Darauf aufbauend werden dann nur solche Spaltenkombinationen als für die Korrelationssuche infrage kommend angesehen, deren Spalten

a) aus derselben Tabelle stammen oder

b) aus einer Verbundtabelle stammen, wobei der Verbund („Join“) mittels (Un-) Gleichung zwischen Schlüssel- und Fremdschlüsselspalten entstanden ist.

Zudem gibt es zusätzliche Reduktionsregeln (engl. „pruning rules“) für das Finden der Beziehungen und für die Auswahl der zu betrachtenden Spaltenkombinationen. Schließlich kann die Spaltenanzahl sehr hoch sein, was die Anzahl an möglichen Kombinationen gegebenenfalls ins Unermessliche steigert.

Im zweiten „Impfschritt“ wird für jede Spaltenkombination eine Stichprobe entnommen und darauf aufbauend eine Kontingenztabelle erstellt. Letztere dient dann wiederum als Grundlage für einen Chi-Quadrat-Test, der als Ergebnis eine Zahl $\chi^2 \geq 0$ liefert. Gilt $\chi^2 = 0$, so sind die Spalten vollständig unabhängig. Da dieser Fall aber in der Praxis kaum auftritt, muss $\chi^2$ einen gewissen Schwellwert überschreiten, damit die entsprechende Spaltenkombination als korreliert angesehen wird. Zum Schluss wird eine Art Rangliste der Spaltenkombinationen mit den höchsten $\chi^2$-Werten erstellt und für die obersten $n$ Kombinationen werden zusätzliche Statistikdaten angelegt. Die Zahl $n$ ist dabei u. a. durch die Größe des Speicherplatzes (für Statistikdaten) begrenzt.

[2]Die Begrenzung wird damit begründet, dass auf diese Weise das beste Aufwand-Nutzen-Verhältnis entsteht. Das Verfahren selbst ist nicht auf Spaltenpaare beschränkt.

## 2.2 Reaktive (anfragegetriebene) Verfahren

Während wir im vorherigen Abschnitt Vermutungen aufgestellt und auf Verdacht gehandelt haben, um den Datenbankbenutzer glücklich zu machen, gehen wir jetzt davon aus, dass den Benutzer auch weiterhin das interessieren wird, wofür er sich bis jetzt interessiert hat.

Wir ziehen also aus der Vergangenheit Rückschlüsse für die Zukunft, und zwar indem wir den Benutzer bei seinem Tun beobachten und darauf reagieren (daher auch die Bezeichnung „reaktiv“). Dabei achten wir nicht allein auf die gestellten SQL-Anfragen, sondern überwachen viel mehr die von der Datenbank zurückgegebenen Anfrageergebnisse. Diese verraten uns nämlich alles (jeweils 100-prozentig aktuell!) über den Teil der vorhandenen Datenlandschaft, den der Benutzer bis jetzt interessant fand.

Auf diese Weise können bspw. Statistiken erzeugt werden [5, 11, 3] (wobei STHoles [5] und ISOMER [11] sogar in der Lage sind, mehrdimensionale Statistiken zu erstellen) oder es lassen sich mithilfe alter Anfragen neue, ähnliche Anfragen in ihrer Performance verbessern [12]. Sinnvoll kann auch eine Unterbrechung der Anfrageausführung mit damit verbundener Reoptimierung sein [9, 2, 10]. Zu guter letzt lässt sich mithilfe dieses Ansatzes zumindest herausfinden, welche Statistikdaten entscheidend sein könnten [1].

In [1] haben Aboulnaga et al. auch schon erste Ansätze für eine Analyse auf Spaltenkorrelation vorgestellt, welche später in [6] durch Haas et al. ausgebaut und verbessert wurden. In Analogie zu CORDS werden in [1] und [6] nur Spaltenpaare für die Korrelationssuche in Betracht gezogen. Allerdings fällt die Auswahl der infrage kommenden Spaltenpaare wesentlich leichter aus, weil einfach alle Spaltenpaare, die in den Anfragen (mit hinreichend vielen Daten[3]) vorkommen, potentielle Kandidaten bilden.

Während in [1] pro auftretendes Wertepaar einer Spaltenkombination ein Quotient aus „Häufigkeit bei Unabhängigkeit“ und „tatsächliche Häufigkeit“ gebildet und das Spaltenpaar als „korreliert“ angesehen wird, sobald zu viele dieser Quotienten von einem gewissen Wert abweichen, setzen Haas et al. in [6] einen angepassten Chi-Quadrat-Test ein, um Korrelationen zu finden. Dieser ist etwas aufwendiger als die Vorgehensweise von [1], dafür jedoch nicht so fehleranfällig [6]. Zudem stellen Haas et al. in [6] Möglichkeiten vor, wie sich die einzelnen „Korrelationswerte“ pro Spaltenpaar miteinander vergleichen lassen, sodass, ähnlich wie in CORDS, eine Rangliste der am stärksten korrelierten Spaltenpaare erstellt werden kann. Diese kann als Entscheidungshilfe für das Anlegen zusätzlicher Statistikdaten genutzt werden.

## 3. HERAUSFORDERUNGEN

In [6] wurde bereits vorgeschlagen, dieses Verfahren mit CORDS zu verbinden. Das reaktive Verfahren spricht aufgrund seiner Effizienz für sich, während das proaktive Verfahren eine gewisse Robustheit bietet und somit bei Lernphasen von [6] (wenn es neu eingeführt wird oder wenn sich die Anfragen ändern) robuste Schätzwerte zur Erstellung eines Anfrageplans berechnet werden können [6]. Dazu sollte CORDS entweder in einem gedrosselten Modus während des normalen Datenbankbetriebs laufen oder während Wartungszeiten ausgeführt werden. Allerdings werden in [6] keine Aussagen darüber getroffen, wie die jeweiligen Ergebnis-

[3]Um aussagefähige Ergebnisse zu bekommen, wird ein gewisses Mindestmaß an Beobachtungen benötigt, insb. in [6].

se beider Verfahren miteinander kombiniert werden sollten. Folgende Punkte sind dabei zu bedenken:

- Beide Verfahren liefern eine Rangliste mit den als am stärksten von Korrelation betroffenen Spalten. Allerdings sind die den Listen zugrunde liegenden „Korrelationswerte“ (s. bspw. $\chi^2$ im Abschnitt über proaktive Verfahren) auf unterschiedliche Weise entstanden und lassen sich nicht einfach vergleichen. Liefern beide Listen unterschiedliche Spaltenkombinationen, so kann es passieren, dass eine Kombination, die in der eine Liste sehr weit unten erscheint, stärker korreliert ist, als Kombinationen, die auf der anderen Liste sehr weit oben aufgeführt sind.
- Die Daten, welche zu einer gewissen Entscheidung bei den beiden Verfahren führen, ändern sich, werden aber in der Regel nicht gleichzeitig von beiden Verfahren gelesen. Das hängt damit zusammen, dass CORDS zu einem bestimmten Zeitpunkt eine Stichprobe entnimmt und darauf seine Analyse aufbaut, während das Verfahren aus [6] die im Laufe der Zeit angesammelten Anfragedaten auswertet.
- Da zusätzliche Statistikdaten Speicherplatz benötigen und vor allem gewartet werden müssen, ist es nicht sinnvoll, einfach für alle Spaltenkombinationen, die in der einen und/oder der anderen Rangliste vorkommen, gleich zu verfahren und zusätzliche Statistiken zu erstellen.

Zur Verdeutlichung wollen wir die Tabelle aller Firmenwagen eines großen, internationalen IT-Unternehmens betrachten, in welcher zu jedem Wagen u. a. seine Farbe und die Personal- sowie die Abteilungsnummer desjenigen Mitarbeiters verzeichnet ist, der den Wagen hauptsächlich nutzt. Diverse dieser Mitarbeiter wiederum gehen in einem Dresdener mittelständischen Unternehmen ein und aus, welches nur rote KFZ auf seinem Parkplatz zulässt (aus Kapazitätsgründen wurde eine solche, vielleicht etwas seltsam anmutende Regelung eingeführt). Da die Mitarbeiter sich dieser Regelung bei der Wahl ihres Wagens bewusst waren, fahren sie alle ein rotes Auto. Zudem sitzen sie alle in derselben Abteilung.

Allerdings ist das internationale Unternehmen wirklich sehr groß und besitzt viele Firmenwagen sowie unzählige Abteilungen, sodass diese roten Autos in der Gesamtheit der Tabelle nicht auffallen. In diesem Sinne würde das proaktive Verfahren CORDS also (eher) keinen Zusammenhang zwischen der Abteilungsnummer des den Wagen benutzenden Mitarbeiters und der Farbe des Autos erkennen.

Werden aber häufig genau diese Mitarbeiter mit der Farbe ihres Wagens abgefragt, z. B. weil sich diese kuriose Regelung des mittelständischen Unternehmens herumspricht, es keiner so recht glauben will und deswegen die Datenbank konsultiert, so könnte ein reaktives Verfahren feststellen, dass beide Spalten korreliert sind. Diese Feststellung tritt insbesondere dann auf, wenn sonst wenig Anfragen an beide betroffenen Spalten gestellt werden, was durchaus möglich ist, weil sonst die Farbe des Wagens eine eher untergeordnete Rolle spielt.

Insbesondere der letztgenannte Umstand macht deutlich, dass es nicht sinnvoll ist, Statistikdaten für die Gesamtheit beider Spalten zu erstellen und zu warten. Aber die Tatsache, dass bestimmte Spezialfälle für den Benutzer besonders interessant sein könnten, die möglicherweise eben gerade mit Korrelation einhergehen, spricht wiederum für eine Art „Hinweis“ an den Optimizer.

## 4. LÖSUNGSANSATZ

Da CORDS wie auch das Verfahren aus [6] nur Spaltenpaare betrachten und dies mit einem sich experimentell ergebenem Aufwand-Nutzen-Optimum begründen, werden auch wir uns auf Spaltenpaare begrenzen. Allerdings wollen wir uns für die Kombination von proaktiver und reaktiver Korrelationssuche zunächst nicht auf diese beiden Verfahren beschränken, müssen aber doch gewisse Voraussetzungen an die verwendeten Verfahren (und das Datenmodell der Datenbank) stellen. Diese seien hier aufgezählt:

1. Entscheidung über die zu untersuchenden Spaltenkombinationen:
   - Das proaktive Verfahren betreibt „reverse engineering“, um zu entscheiden, welche Spaltenkombinationen untersucht werden sollen.
   - Das Datenmodell der Datenbank ändert sich nicht, bzw. sind nur geringfügige Änderungen zu erwarten, welche vom proaktiven Verfahren in das von ihm erstellte Datenmodell sukzessive eingearbeitet werden können. Auf diese Weise können wir bei unseren Betrachtungen den ersten „Impfschritt“ vernachlässigen.
2. Datengrundlage für die Untersuchung:
   - Das proaktive Verfahren entnimmt für jegliche zu untersuchende Spaltenkombination eine Stichprobe, welche mit einem Zeitstempel versehen wird. Diese Stichprobe wird solange aufbewahrt, bis das Verfahren auf „Unkorreliertheit“ plädiert oder für die entsprechende Spaltenkombination eine neue Stichprobe erstellt wird.
   - Das reaktive Verfahren bedient sich eines Query-Feedback-Warehouses, in welchem die Beobachtungen („Query-Feedback-Records“) der Anfragen notiert sind.
3. Vergleich der Ergebnisse:
   - Beide Verfahren geben für jede Spaltenkombination, die sie untersucht haben, einen „Korrelationswert“ aus, der sich innerhalb des Verfahrens vergleichen lässt. Wie dieser genau berechnet wird, ist für uns unerheblich.
   - Aus den höchsten Korrelationswerten ergeben sich zwei Ranglisten der am stärksten korrelierten Spaltenpaare, die wir unterschiedlich auswerten wollen.

Zudem wollen wir davon ausgehen, dass das proaktive Verfahren in einem gedrosselten Modus ausgeführt wird und somit sukzessive seine Rangliste befüllt. (Zusätzliche Wartungszeiträume, bei denen das Verfahren ungedrosselt laufen kann, beschleunigen die Arbeit und bilden somit einen schönen Zusatz, aber da heutzutage viele Datenbanken quasi dauerhaft laufen müssen, wollen wir sie nicht voraussetzen.)

Das reaktive Verfahren dagegen wird zu bestimmten Zeitpunkten gestartet, um die sich bis dahin angesammelten Beobachtungen zu analysieren, und gibt nach beendeter Analyse seine Rangliste bekannt. Da es als Grundlage nur die Daten aus dem Query-Feedback-Warehouse benötigt, kann es völlig entkoppelt von der eigentlichen Datenbank laufen.

Ist die reaktive Rangliste bekannt, kann diese mit der (bis dahin angefertigten) proaktiven Rangliste verglichen werden. Tritt eine Spaltenkombination in beiden Ranglisten auf, so bedeutet das, dass diese Korrelation für die bisherigen Anfragen eine Rolle gespielt hat und nicht nur auf Einzelfälle beschränkt ist, sondern auch mittels Analyse einer repräsentativen Stichprobe an Wertepaaren gefunden wurde.

Unter diesen Umständen lassen wir mittels einer Stichprobe Statistikdaten für die betreffende Spaltenkorrelation erstellen. Dabei wählen wir die Stichprobe des proaktiven Verfahrens, solange diese ein gewisses Alter nicht überschritten hat. Ist sie zu alt, wird eine neue Stichprobe entnommen.[4]

Interessanter wird es, wenn nur eines der Verfahren auf Korrelation tippt, während das andere Verfahren die entsprechende Spaltenkombination nicht in seiner Rangliste enthält. Die Ursache dafür liegt entweder darin, dass letzteres Verfahren die Kombination noch nicht analysiert hat (beim reaktiven Verfahren heißt das, dass sie nicht oder zu selten in den Anfragen vorkam), oder bei seiner Analyse zu dem Ergebnis „nicht korreliert“ gekommen ist.

Diese Unterscheidung wollen wir insbesondere in dem Fall vornehmen, wenn einzig das reaktive Verfahren die Korrelation „entdeckt“ hat. Unter der Annahme, dass weitere, ähnliche Anfragen folgen werden, benötigt der Optimizer schnell Statistiken für den abgefragten Bereich. Diese sollen zunächst reaktiv mithilfe der Query-Feedback-Records aus der Query-Feedback-Warehouse erstellt werden (unter Verwendung von bspw. [11], wobei wir nur zweidimensionale Statistiken benötigen). Das kann wieder völlig getrennt von der eigentlichen Datenbank geschehen, da nur das Query-Feedback-Warehouse als Grundlage dient.

Wir überprüfen nun, ob das proaktive Verfahren das Spaltenpaar schon bearbeitet hat. Dies sollte anhand der Abarbeitungsreihenfolge der infrage kommenden Spaltenpaare erkennbar sein.

Ist dem so, hat das proaktive Verfahren das entsprechende Paar als „unkorreliert“ eingestuft und wir bleiben bei den reaktiv erstellten Statistiken, die auch nur reaktiv aktualisiert werden. Veralten sie später zu stark aufgrund fehlender Anfragen (und somit fehlendem Nutzerinteresse), können sie gelöscht werden.

Ist dem nicht so, geben wir die entsprechende Kombination an das proaktive Verfahren weiter mit dem Auftrag, diese zu untersuchen.[5] Beim nächsten Vergleich der Ranglisten muss es für das betrachtete Spaltenpaar eine konkrete Antwort geben. Entscheidet sich das proaktive Verfahren für „korreliert“ und befindet sich das Spaltenpaar auch wieder in der Rangliste des reaktiven Verfahrens, dann löschen wir die reaktiv erstellten Statistiken und erstellen neue Statistiken mittels einer Stichprobe, analog zum ersten Fall. (Die Kombination beider Statistiktypen wäre viel zu aufwendig, u. a. wegen unterschiedlicher Entstehungszeitpunkte.) Wenn das proaktive Verfahren dagegen explizit „unkorreliert“ ausgibt, bleibt es bei den reaktiv erstellten Statistiken, s. oben.

Wenn jedoch nur das proaktive Verfahren eine bestimmte Korrelation erkennt, dann ist diese Erkenntnis zunächst für die Benutzer unerheblich. Sei es, weil der Nutzer diese Spaltenkombination noch gar nicht abgefragt hat, oder weil er bis jetzt nur den Teil der Daten benötigt hat, der scheinbar unkorreliert ist. In diesem Fall markieren wir nur im Datenbankkatalog (wo die Statistiken abgespeichert werden) die beiden Spalten als korreliert und geben dem Optimizer somit ein Zeichen, dass hier hohe Schätzfehler möglich sind und er deswegen robuste Pläne zu wählen hat. Dabei bedeutet „robust“, dass der gewählte Plan für die errechneten Schätzwerte möglicherweise nicht ganz optimal ist, dafür aber bei stärker abweichenden „wahren Werten“ immer noch akzeptable Ergebnisse liefert. Zudem können wir ohne wirklichen Einsatz des reaktiven Verfahrens die Anzahl der Anfragen zählen, die auf diese Spalten zugreifen und bei denen sich der Optimizer stark verschätzt hat. Übersteigt der Zähler einen Schwellwert, werden mithilfe einer neuen Stichprobe (vollständige, also insb. mit Werteverteilung) Statistikdaten erstellt und im Katalog abgelegt.

Der Vollständigkeit halber wollen wir hier noch den Fall erwähnen, dass eine Spaltenkombination weder in der einen, noch in der anderen Rangliste vorkommt. Es sollte klar sein, dass diese Kombination als „unkorreliert“ angesehen und somit für die Statistikerstellung nicht weiter betrachtet wird.

## 5. AUSBLICK

Die hier vorgestellte Vorgehensweise zur Verbesserung der Korrelationsfindung mittels Einsatz zweier unterschiedlicher Verfahren muss weiter vertieft und insbesondere praktisch umgesetzt und getestet werden. Vor allem muss ein passendes Datenmodell für die reaktive Erstellung von Spaltenpaarstatistiken gefunden werden. Das vorgeschlagene Verfahren ISOMER [11] setzt hier auf STHoles [5], einem Datenmodell, welches bei sich stark überschneidenden Anfragen schnell inperformant werden kann. Für den eindimensionalen Fall wurde bereits von Informix-Entwicklern eine performante Lösung vorgestellt [3], welche sich aber nicht einfach auf den zweidimensionalen Fall übertragen lässt.

Eine weitere, noch nicht völlig ausgearbeitete Herausforderung bildet die Tatsache, dass das proaktive Verfahren im gedrosselten Modus läuft und erst sukzessive seine Rangliste erstellt. Das bedeutet, dass wir eigentlich nur Zwischenergebnisse dieser Rangliste mit der reaktiv erstellten Rangliste vergleichen. Dies kann zu unerwünschten Effekten führen, z. B. könnten beide Ranglisten völlig unterschiedliche Spaltenkombinationen enthalten, was einfach der Tatsache geschuldet ist, dass beide Verfahren unterschiedliche Spaltenkombinationen untersucht haben. Um solche Missstände zu vermeiden, muss die proaktive Abarbeitungsreihenfolge der Spaltenpaare überdacht werden. In CORDS wird bspw. als Reduktionsregel vorgeschlagen, nur Spaltenpaare zu be-

[4]Falls die betroffenen Spalten einen Zähler besitzen, der bei Änderungsoperationen hochgezählt wird (vgl. z. B. [1]), können natürlich auch solche Daten mit in die Wahl der Stichprobe einfließen, allerdings sind hier unterschiedliche „Ausgangszeiten“ zu beachten.

[5]Dadurch stören wir zwar etwas die vorgegebene Abarbeitungsreihenfolge der infrage kommenden Spaltenpaare, aber der Fall ist ja auch dringend.

trachten, die im Anfrageworkload vorkommen (dazu müssen von CORDS nur die Anfragen, aber nicht deren Ergebnisse betrachtet werden). Würde sich dann aber der Workload dahingehend ändern, dass völlig neue Spalten oder Tabellen abgefragt werden, hätten wir dasselbe Problem wie bei einem rein reaktiven Verfahren. Deswegen muss hier eine Zwischenlösung gefunden werden, die Spaltenkombinationen aus Anfragen bevorzugt „behandelt“, sich aber nicht darauf beschränkt.

Außerdem muss überlegt werden, wann wir Statistikdaten, die auf Stichproben beruhen, wieder löschen können. Im reaktiven Fall fiel die Entscheidung leicht aus, weil fehlender Zugriff auf die Daten auch ein fehlendes Nutzerinteresse widerspiegelt und auf diese Weise auch keine Aktualisierung mehr stattfindet, sodass die Metadaten irgendwann unbrauchbar werden.

Basieren die Statistiken dagegen auf Stichproben, müssen sie von Zeit zu Zeit aktualisiert werden. Passiert diese Aktualisierung ohne zusätzliche Überprüfung auf Korrelation (welche ja aufgrund geänderten Datenbestands nachlassen könnte), müssen mit der Zeit immer mehr zusätzliche Statistikdaten über Spaltenpaare gespeichert und gewartet werden. Der für Statistikdaten zur Verfügung stehende Speicherplatz im Katalog kann so an seine Grenzen treten, außerdem kostet die Wartung wiederum Kapazität des DBMS. Hier müssen sinnvolle Entscheidungen über die Wartung und das „Aufräumen“ nicht mehr benötigter Daten getroffen werden.

## 6. REFERENCES

[1] A. Aboulnaga, P. J. Haas, S. Lightstone, G. M. Lohman, V. Markl, I. Popivanov, and V. Raman. Automated statistics collection in DB2 UDB. In *VLDB*, pages 1146–1157, 2004.

[2] S. Babu, P. Bizarro, and D. J. DeWitt. Proactive re-optimization. In *SIGMOD Conference*, pages 107–118. ACM, 2005.

[3] E. Behm, V. Markl, P. Haas, and K. Murthy. Integrating query-feedback based statistics into informix dynamic server, Apr. 03 2008.

[4] P. Brown and P. J. Haas. BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data. In *VLDB 2003: Proceedings of 29th International Conference on Very Large Data Bases, September 9–12, 2003, Berlin, Germany*, pages 668–679, 2003.

[5] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. *SIGMOD Rec.*, 30(2):211–222, May 2001.

[6] P. J. Haas, F. Hueske, and V. Markl. Detecting attribute dependencies from query feedback. In *VLDB*, pages 830–841. ACM, 2007.

[7] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In ACM, editor, *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data 2004, Paris, France, June 13–18, 2004*, pages 647–658, pub-ACM:adr, 2004. ACM Press.

[8] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Correlation maps: A compressed access method for exploiting soft functional dependencies. *PVLDB*, 2(1):1222–1233, 2009.

[9] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust query processing through progressive optimization. In ACM, editor, *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data 2004, Paris, France, June 13–18, 2004*, pages 659–670. ACM Press, 2004.

[10] T. Neumann and C. Galindo-Legaria. Taking the edge off cardinality estimation errors using incremental execution. In *BTW*, pages 73–92, 2013.

[11] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. ISOMER: Consistent histogram construction using query feedback. In *ICDE*, page 39. IEEE Computer Society, 2006.

[12] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO - DB2's learning optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01)*, pages 19–28, Orlando, Sept. 2001.

# MVAL: Addressing the Insider Threat by Valuation-based Query Processing

Stefan Barthel
Institute of Technical and Business Information Systems
Otto-von-Guericke-University Magdeburg
Magdeburg, Germany
stefan.barthel@ovgu.de

Eike Schallehn
Institute of Technical and Business Information Systems
Otto-von-Guericke-University Magdeburg
Magdeburg, Germany
eike.schallehn@ovgu.de

## ABSTRACT

The research presented in this paper is inspired by problems of conventional database security mechanisms to address the insider threat, i.e. authorized users abusing granted privileges for illegal or disadvantageous accesses. The basic idea is to restrict the data one user can access by a valuation of data, e.g. a monetary value of data items, and, based on that, introducing limits for accesses. The specific topic of the present paper is the conceptual background, how the process of querying valuated data leads to valuated query results. For this, by analyzing operations of the relational algebra and SQL, derivation functions are added.

## 1. INTRODUCTION

An acknowledged main threat to data security are fraudulent accesses by authorized users, often referred to as the insider threat [2]. To address this problem, in [1] we proposed a novel approach of detecting authorization misuse based on a valuation of data, i.e. of an assigned description of the worth of data management in a system, which could for instance be interpreted as monetary values. Accordingly, possible security leaks exist if users access more valuable data than they are allowed to within a query or cumulated over a given time period. E.g., a bank account manager accessing a single customer record does not represent a problem, while dumping all data in an unrestricted query should be prohibited. Here, common approaches like role-based security mechanisms typically fail.

According to our proposal, the data valuation is first of all based on the relation definitions, i.e. as part of the data dictionary information about the value of data items such as attribute values and, derived from that, entire tuples and relations. Then, a key question is how the valuation of a query result can be derived from the input valuations, because performing operations on the base data causes transformations that have an impact on the data's significance.

This problem is addressed in the research presented here by considering relational and SQL operations and describing possible valuation derivations for them.

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

## 2. PRINCIPLES OF DATA VALUATION

In [1] we outlined our approach of a leakage-resistant data valuation which computes a monetary value (*mval*) for each query. This is based on the following basic principles: Every attribute $A_i \in R$ of a base relation schema $R$ is valuated by a certain monetary value ($mval(A_i) \in \mathbb{R}$). The attribute valuation for base tables are part of the data dictionary and can for instance be specified as an extension of the SQL DDL:

```
CREATE TABLE table_1
(
  attribute_1 INT PRIMARY KEY MVAL 0.1,
  attribute_2 UNIQUE COMMENT 'important' MVAL 10,
  attribute_3 DATE
);
```

With these attribute valuations, we derive a monetary value for one tuple $t \in r(R)$ given by Equation (1), as well as the total monetary value of the relation $r(R)$ given by Equation (2), if data is extracted by a query.

$$mval(t \in r(R)) = \sum_{A_i \in R} mval(A_i) \quad (1)$$

$$mval(r(R)) = \sum_{t \in r(R)} mval(t) = |r(R)| * mval(t \in r(R)) \quad (2)$$

To be able to consider the *mval* for a query as well as several queries of one user over a certain period of time, we log all *mvals* in an alert log and compare the current cumulated *mval* per user to two thresholds. If a user exceeds the first threshold – suspicious threshold – she will be categorized as suspect. After additionally exceeding the truncation threshold her query output will be limited by hiding tuples and presenting a user notification. We embedded our approach in an additional layer in the security defense-in-depth model for raw data, which we have enhanced by a backup entity (see Fig. 1). Furthermore, encryption has to be established to prevent data theft via unauthorized, physical reads as well as backup theft. In this paper we are going into detail about how to handle operations like joins, aggregate functions, stored procedures as well as common functions.

Most of the data stored in a database can be easily identified as directly interpretable. One example would be an

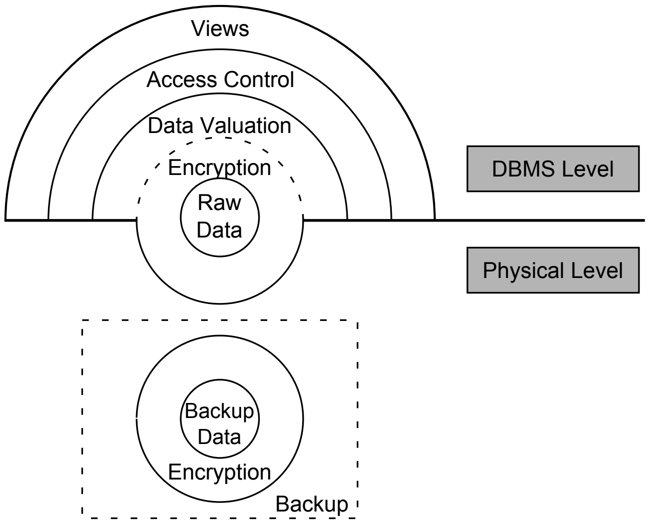


**Figure 1: Security defense model on DBMS and physical level**

employee-table, where each tuple has a value for attributes "first name", "surname" and "gender". In this case, it is also quite easy to calculate the monetary value for a query ($r(R_{emp})$) by simply summarizing all *mval* per attribute and multiply those with the number of involved rows (see Eq. (3)).

$$mval(r(R_{emp})) = \sum_{A_i \in R_{emp}} mval(A_i) * |r(R_{emp})| \qquad (3)$$

However, it becomes more challenging if an additional attribute "license plate number" is added, which does have some unset or unknown attribute values - in most cases *NULL* values. By knowing there is a *NULL* value for a certain record, this could be interpreted as either simply unknown whether there is any car or unset because this person has no car. So there is an uncertainty that could lead to an information gain which would be uncovered if no adequate valuation exists. Some other potentially implicit information gains are originated from joins and aggregate functions which we do mention in the regarding section.

Because the terms *information gain* and *information loss* are widely used and do not have a uniform definition, we do define them for further use. We call a situation where an attacker received new data (resp. information) *information gain* and the same situation in the view of the data owner an *information loss.*

### *Uncertainty Factor*

Some operators used for query processing obviously reduce the information content of the result set (e.g. selection, aggregations, semi joins, joins with resulting NULL values), but there is still an uncertain, implicit information gain. Since, the information gain by uncertainty is blurry, meaning in some cases more indicative than in others, we have to distinguish uncertainty of one attribute value generated out of one source attribute value (e.g., generated NULL values) and attribute values which are derived from information of several source attribute values (e.g., aggregations). In case of one source attribute value, an information gain by uncertainty has to be less valuable than properly set attribute values. Therefore, the monetary value should be only a percentage of the respective monetary value of an attribute value. If several source attribute values are involved, we recommend to value the computed attribute value as a percentage of the monetary value of all participating source attribute values. In general, we suggest a maximum of 50% for both valuations. Furthermore, we need to consider the overall purpose of our leakage-resistant data valuation which shall prevent extractions of large amounts of data. Therefore, the percentage needs to be increased with the amount of data, but not in a way that an unset or unknown attribute value becomes equivalent valuable than a properly set one. For that reason, exponential growth is not a suitable option. Additionally, we have to focus a certain area of application, because a trillion attributes ($10^{12}$) are conceivable whereas a septillion attributes ($10^{24}$) are currently not realistic. From the overall view on our data valuation, we assume depending on the application, that the extraction of sensitive data becomes critical when $10^3$ up to $10^9$ attribute values will be extracted. Therefore, the growth of our uncertainty factor *UF* increases much more until $10^9$ attribute values than afterwards, which predominantly points to a logarithmic growth. We also do not need to have a huge difference of the factor if theoretically much more attribute values shall be extracted (e.g., $10^{14}$ and more), because with respect to an extraction limiting approach, it is way too much data to return. This assumption does also refer to a logarithmic increase. We conclude that the most promising formula that was adapted to fit our needs is shown in Eq. (4).

$$UF = \frac{1}{30} log_{10}(|val_{A_i,\dots,A_k}| + 1) \qquad (4)$$

## 3. DERIVING VALUATIONS FOR DATABASE OPERATIONS

In this chapter we will describe valuation derivation for main database operations by first discussing core relational operations. Furthermore, we address specifics of join operations and finally functions (aggregate, user-defined, stored procedures) which are defined in SQL.

### 3.1 Core Operations of Relational Algebra

The relational algebra [4] consists of six basic operators, where *selection*, *projection*, and *rename* are unary operations and *union*, *set difference*, and *Cartesian product* are operators that take two relations as input (binary operation). Due to the fact that applying *rename* to a relation or attribute will not change the monetary value, we will only consider the rest.

### *Projection*

The projection $\pi_{attr\_list}(r(R))$ is a unary operation and eliminates all attributes (columns) of an input relation $r(R)$ except those mentioned in the attribute list. For computation of the monetary value of such a projection, only *mval* for chosen attributes of the input relation are considered while taking into account that a projection may eliminate duplicates (shown in Eq. (5)).

$$mval(\pi_{A_j,..,A_k}(r(R))) = \sum_{i=j}^{k} mval(A_i) * |\pi_{A_j,..,A_k}(r(R))| \tag{5}$$

### *Selection*

According to the relational algebra, a *selection* of a certain relation $\sigma_{pred}r(R)$ reduces tuples to a subset which satisfy specified predicates. Because the *selection* reduces the number of tuples, the calculation of the monetary value does not have to consider those filtered tuples and only the number of present tuples are relevant (shown in Eq. (6)).

$$mval(\sigma_{pred}(r(R))) = mval(t \in r(R)) * |\sigma_{pred}(r(R))| \tag{6}$$

### *Set Union*

A relation of all distinct elements (resp. tuples) of any two relations is called the *union* (denoted by $\cup$) of those relations. For performing set union, the two involved relations must be union-compatible – they must have the same set of attributes. In symbols, the union is represented as $R_1 \cup R_2 = \{x : x \in R_1 \vee x \in R_2\}$. However, if two relations contain identical tuples, within a resulting relation these tuples do only exist once, meaning duplicates are eliminated. Accordingly, the *mval* of a union of two relations is computed by adding *mval* of both relations, subtracted with *mval* of duplicates (shown in Eq. (7)).

$$\begin{aligned} mval(R_1 \cup R_2) = mval(r(R_1)) + \\ mval(r(R_2)) - \sum_i mval(t_i \in r(R_1 \cap R_2) \end{aligned} \tag{7}$$

### *Set Difference*

The difference of relations $R_1$ and $R_2$ is the relation that contains all the tuples that are in $R_1$, but do not belong to $R_2$. The set difference is denoted by $R_1 - R_2$ or $R_1 \backslash R_2$ and defined by $R_1 \backslash R_2 = \{x : x \in R_1 \wedge x \notin R_2\}$. Also, the set difference is union-compatible, meaning the relations must have the same number of attributes and the domain of each attribute is the same in both $R_1$ and $R_2$. The *mval* of a set difference of two relations is computed by subtracting the *mval* of tuples that have both relations in common from the monetary value of $R_1$ given by Equation (8).

$$mval(R_1 \backslash R_2) = mval(r(R_1) - \sum_i mval(t_i \in r(R_1 \cap R_2) \tag{8}$$

### *Cartesian Product*

The Cartesian product, also known as cross product, is an operator which works on two relations, just as set union and set difference. However, the Cartesian product is the costliest operator to evaluate [9], because it combines the tuples of one relation with all the tuples of the other relation – it pairs rows from both tables. Therefore, if the input relations $R_1$ and $R_2$ have $n$ and $m$ rows, respectively, the result set will contain $n * m$ rows and consist of columns of $R_1$ and the columns of $R_2$. Because, the number of tuples of the outgoing relations are known, the monetary value is a summation of all attribute valuations multiplied by number of rows of both relations given by Equation (9). We are fully aware that by a user mistake, e.g. using cross join instead of natural join, thresholds will be exceeded and the user will be classified as potentially suspicious. However, we recommend a multiplication of the monetary value of both source relations instead of a summation due to the fact that the calculation of the monetary value needs to be consistent also by combining different operators. For that reason, by following our recommendation, we ensure that an inner join is valuated with the same monetary value as the respective combination of a cross join (Cartesian product) and selection on the join condition.

$$\begin{aligned} &mval(r(R_1 \times R_2)) = \\ &\quad mval(t \in r(R_1)) * |r(R_1)| + mval(t \in r(R_2)) * |r(R_2)| \end{aligned} \tag{9}$$

## 3.2 Join Operations

In the context of relational databases, a join is a binary operation of two tables (resp. data sources). The result set of a join is an association of tuples from one table with tuples from another table by concatenating concerned attributes. Joining is an important operation and most often performance critical to certain queries that target tables whose relationships to each other cannot be followed directly. Because the type of join affects the number of resulting tuples and their attributes, the monetary value of each join needs to be calculated independently.

### *Inner Join*

An inner join produces a result table containing composite rows of involved tables that match some pre-defined, or explicitly specified, join condition. This join condition can be any simple or compound search condition, but does not have to contain a subquery reference. The valuation of an inner join is computed by the sum of the monetary values of all attributes of a composite row multiplied by the number of rows within the result set. Because the join attribute $A_{join}$ of two joined tables has to be counted only once, we need to subtract it (shown in Eq. (10)).

$$\begin{aligned} &mval(r(R_1 \bowtie R_2) = |r(R_1 \bowtie R_2)| * \\ &(mval(t \in r(R_1)) + (mval(t \in r(R_2)) - mval(A_{join})) \end{aligned} \tag{10}$$

### *Outer Join*

An outer join does not require matching records for each tuple of concerned tables. The joined result table retains all rows from at least one of the tables mentioned in the *FROM* clause, as long as those rows are consistent with the search condition. Outer joins are subdivided further into left, right, and full outer joins. The result set of a left outer join (or left join) includes all rows of the first mentioned table (left of the join keyword) merged with attribute values of the right table where the join attribute matches. In case there is no match, attributes of the right table are set to *NULL*. The right outer join (or right join) will return rows that have data in the right table, even if there's no matching rows in the left table enhanced by atteributes (with NULL values) of the left table. A full outer join is used to retain the non-matching information of all affected tables by including non-matching rows in the result set. To cumulate the monetary value for a query that contains a left or right outer join, we only need to compute the monetary value of an inner join of both tables and add the *mval* of an antijoin $r(R_1 \triangleright R_2) \subseteq r(R_1)$ which includes only tuples of $R_1$ that do not have a join partner in $R_2$(shown in Eq. (11)). For the monetary value of a full outer join, we additionally would consider an antijoin $r(R_2 \triangleright R_1) \subseteq r(R_2)$ which includes tuples of $R_2$ that do not have a join partner given by Equation (12)).

$$mval(r(R_1 ⟕ R_2)) = mval(r(R_1 \bowtie R_2)) + mval(r(R_1 \triangleright R_2)) \tag{11}$$

$$mval(r(R_1 ⟗ R_2)) = mval(r(R_1 \bowtie R_2)) + mval(r(R_1 \triangleright R_2)) + mval(r(R_2 \triangleright R_1)) \tag{12}$$

### *Semi Join*

A semi join is similar to the inner join, but with the addition that only attributes of one relation are represented in the result set. Semi joins are subdivided further into left and right semi joins. The left semi join operator returns each row from the first input relation (left of the join keyword) when there is a matching row in the second input relation (right of the join keyword). The right semi join is computed vice versa. The monetary value for a query that uses semi joins can be easily cumulated by multiplying the sum of monetary values for included attributes with number of matching rows of the outgoing relation (shown in Eq. (13)).

$$mval(r(R_1 \ltimes R_2)) = \sum_{A_i \in R_1} mval(A_i) * |r(R_1 \ltimes R_2)| \tag{13}$$

Nevertheless, we do have an information gain by knowing join attributes of $R_1$ have some join partners within $R_2$ which are not considered. But adding our uncertainty factor *UF* in this equation would lead to inconsistency by cumulating the *mval* of a semi join compared to the *mval* of a combination of a natural join and a projection. In future work, we will solve this issue by presenting a calculation that is based on a combination of projections and joins to cover such an implicit information gain.

## 3.3 Aggregate Functions

In computer science, an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning. The SQL aggregate functions are useful when mathematical operations must be performed on all or on a group of values. For that reason, they are frequently used with the *GROUP BY* clause within a *SELECT* statement. According to the SQL standard, the following aggregate function are implemented in most DBMS and the ones used most often: *COUNT*, *AVG*, *SUM*, MAX, and MIN.

All aggregate functions are deterministic, i.e. they return the same value any time they are called by using the same set of input values. SQL aggregate functions return a single value, calculated from values within one column of a arbitrary relation [10]. However, it should be noted that except for *COUNT*, these functions return a *NULL* value when no rows are selected. For example, the function *SUM* performed on no rows returns *NULL*, not zero as one might expect. Furthermore, except for *COUNT*, aggregate functions ignore *NULL* values at all during computation. All aggregate function are defined in SQL:2011 standard or ISO/IEC 9075:2011 (under the general title "Information technology - Database languages - SQL") which is the seventh revision of the ISO (1987) and ANSI (1986) standard for the SQL database query language.

To be able to compute the monetary value of a derived, aggregated attribute, we need to consider two more factors. First of all, we divided aggregate function into two groups: *informative* and *conservative*.

1. *Informative* are those aggregate functions where the aggregated value of a certain aggregate function leads to an information gain of the entire input of all attribute values. This means that every single attribute value participates in the computation of the aggregated attribute value. Representatives for *informative* aggregate functions are *COUNT*, *AVG* and *SUM*.
2. *Conservative*, on the contrary, are those functions where the aggregated value is represented by only one attribute value, but in consideration of all other attribute values. So if the aggregated value are again separated from the input set, all other attribute values will remain. *Conservative* aggregate functions are *MAX* and *MIN*.

The second factor that needs to be considered is the number of attributes that are used to compute the aggregated values. In case of a *conservative* aggregate function, it is simple, because only one attribute value is part of the output. For that reason we recommend to leave the *mval* of the source attribute unchanged (shown in Eq. (14)).

$$mval(A_i) = mval(MAX(A_i)) = mval(MIN(A_i)) \tag{14}$$

For the *informative* aggregate functions the computation is more challenging due to several participating attribute values. Because several input attribute values are concerned, we recommend the usage of our *uncertainty factor* which we already mentioned in a prior section. With the uncertainty factor it is possible to integrate the number of attribute values in a way that a higher number of concerned attributes leads to an increase in percentage terms of the monetary value of the aggregated attribute value given by Equation (15).

$$mval(COUNT(A_i)) = mval(SUM(A_i)) = mval(AVG(A_i)) = \frac{1}{30} log_{10}(|A_i| + 1) * mval(A_i) \tag{15}$$

## 3.4 Scalar Functions

Besides the SQL aggregate functions, which return a single value, calculated from values in a column, there are also scalar functions defined in SQL, that return a single value based on the input value. The possibly most commonly used and well known scalar functions are:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- LEN() - Returns the length of a text field
- ROUND() - Rounds a number to a specified degree
- FORMAT() - Formats how a field is to be displayed

Returned values of this scalar functions are always derived from one source attribute value, and some of them do not even change the main content of the attribute value. Therefore, we recommend that the monetary value of the source attribute stays untouched.

## 3.5 User-Defined Functions

User-defined functions (*UDF*) are subroutines made up of one or several SQL or programming extension statements that can be used to encapsulate code for reuse. Most database management systems (DBMS) allow users to create their own user-defined functions and do not limit them to the built-in functions of their SQL programming language (e.g., TSQL, PL/SQL, etc.). User-defined functions in most systems are created by using the *CREATE FUNCTION* statement and other users than the owner must be granted appropriate permissions on a function before they can use it. Furthermore, *UDFs* can be either deterministic or nondeterministic. A deterministic function always returns the same results if the input is the equal and a nondeterministic function returns different results every time it is called.

On the basis of the multiple possibilities offered by most DBMS, it is impossible to estimate all feasible results of a *UDF*. Also, due to several features like shrinking, concatenating, and encrypting of return values, a valuation of a single or an array of output values is practically impossible. For this reason we decided not to calculate the monetary value depending on the output of a *UDF*, much more we do consider the attribute values that are passed to an *UDF* (shown in Eq. (16)). This assumption is also the most reliable, because it does not matter what happens inside an *UDF* – like a black box – the information loss after inserting cannot get worse.

$$\begin{aligned} &mval(UDF_{output}(A_a, .., A_g)) = \\ &\quad mval(UDF_{input}(A_k, .., A_p)) = \sum_{i=k}^{p} mval(A_i) \end{aligned} \tag{16}$$

## 3.6 Stored Procedures

Stored procedures (*SP*) are stored similar to user-defined functions (*UDF*) within a database system. The major difference is that stored procedures have to be called and the return values of *UDFs* are used in other SQL statements in the same way pre-installed functions are used (e.g., *LEN*, *ROUND*, etc.). A stored procedure, which is depending on the DBMS also called *proc*, *sproc*, *StoredProc* or *SP*, is a group of SQL statements compiled into a single execution plan [13] and mostly developed for applications that need to access easily a relational database system. Furthermore, *SPs* combine and provide logic also for extensive or complex processing that requires execution of several SQL statement, which had to be implemented in an application before. Also a nesting of *SPs* is feasible by executing one stored procedure from within another. A typical use for *SPs* refers to data validation (integrated into the database) or access control mechanisms [13].

Because stored procedures have such a complex structure, nesting is also legitimate and *SPs* are "only" a group of SQL statements, we recommend to value each single statement within a *SP* and sum up all partial results (shown in Eq. (17). With this assumption we do follow the principal that single SQL statements are moved into stored procedures to provide a simple access for applications which only need to call the procedures.

$$mval(SP(r(R_j), .., r(R_k))) = \sum_{i=j}^{k} mval(r(R_i)) \tag{17}$$

Furthermore, by summing all partial result, we make sure that the worst case of information loss is considered, entirely in line with our general idea of a leakage resistant data valuation that should prevent a massive data extraction. However, since *SPs* represent a completed unit, by reaching the truncate threshold the whole *SP* will be blocked and rolled back. For that reason, we recommend smaller *SPs* resp. split existing *SPs* in DBS with an enabled leakage resistant data valuation.

## 4. RELATED WORK

Conventional database management systems mostly use access control models to face unauthorized access on data. However, these are insufficient when an authorized individual extracts data regardless whether she is the owner or has stolen that account. Several methods were conceived to eliminate those weaknesses. We refer to Park and Giordano [14], who give an overview of requirements needed to address the insider threat.

Authorization views partially achieve those crucial goals of an extended access control and have been proposed several times. For example, Rizvi et al. [15] as well as Rosenthal et al. [16] use authorization-transparent views. In detail, incoming user queries are only admitted, if they can be answered using information contained in authorization views. Contrary to this, we do not prohibit a query in its entirety. Another approach based on views was introduced by Motro [12]. Motro handles only conjunctive queries and answers a query only with a part of the result set, but without any indication why it is partial. We do handle information enhancing (e.g., joins), as well as coarsening operations (e.g., aggregation) and we do display a user notification. All authorization view approaches require an explicit definition of a view for each possible access need, which also imposes the burden of knowing and directly querying these views. In contrast, the monetary values of attributes are set while defining the tables and the user can query the tables or views she is used to. Moreover, the equivalence test of general relational queries is undecidable and equivalence for conjunctive queries is known to be NP complete [3]. Therefore, the leakage-resistant data valuation is more applicable, because it does not have to face those challenges.

However, none of these methods does consider the sensitivity level of data that is extracted by an authorized user. In the field of *privacy-preserving data publishing* (*PPDP*), on the contrary, several methods are provided for publishing useful information while preserving data privacy. In detail, multiple security-related measures (e.g., k-anonymity [17], l-Diversity [11]) have been proposed, which aggregate information within a data extract in a way that they can not lead to an identification of a single individual. We refer to Fung et al. [5], who give a detailed overview of recent developments in methods and tools of *PPDP*. However, these mechanisms are mainly used for privacy-preserving tasks and are not in use when an insider accesses data. They are not applicable for our scenario, because they do not consider a line by line extraction over time as well as the information loss by aggregating attributes.

To the best of our knowledge, there is only the approach of Harel et al. ([6], [7], [8]) that is comparable to our data valuation to prevent suspicious, authorized data extractions. Harel et al. introduce the *Misuseability Weight (M-score)* that desribes the sensitivity level of the data exposed to the user. Hence, Harel et al. focus on the protection of the quality of information, whereas our approach predominantly preserves the extraction of a collection of data (quantity of information). Harel et al. also do not consider extractions over time, logging of malicious requester and the backup process. In addition, mapping attributes to a certain monetary value is much more applicable and intuitive, than mapping to a artificial M-score.

Our extended authorization control does not limit the system to a simple query-authorization control without any protection against the insider threat, rather we allow a query to be executed whenever the information carried by the query is legitimate according to the specified authorizations and thresholds.

## 5. CONCLUSIONS AND FUTURE WORK

In this paper we described conceptual background details for a novel approach for database security. The key contribution is to derive valuations for query results by considering the most important operations of the relational algebra as well as SQL and providing specific *mval* functions for each of them. While some of these rules are straight forward, e.g. for core operations like selection and projection, other operations like specific join operations require some more thorough considerations. Further operations, e.g. grouping and aggregation or user-defined function, would actually require application specific valuations. To minimize the overhead for using valuation-based security, we discuss and recommend some reasonable valuation functions for these cases, too.

As the results presented here merely are of conceptual nature, our current and future research includes considering implementation alternatives, e.g. integrated with a given DBMS or as part of a middleware or driver as well as evaluating the overhead and the effectiveness of the approach. We will also come up with a detailed recommendation of how to set monetary values appropriate to different environments and situations. Furthermore, we plan to investigate further possible use cases for data valuation, such as billing of data-providing services on a fine-grained level and controlling benefit/cost trade-offs for data security and safety.

## 6. ACKNOWLEDGMENTS

This research has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract FKZ: 13N10818.

## 7. REFERENCES

[1] S. Barthel and E. Schallehn. The Monetary Value of Information: A Leakage-Resistant Data Valuation. In *BTW Workshops*, BTW'2013, pages 131–138. Köln Verlag, 2013.

[2] E. Bertino and R. Sandhu. Database Security - Concepts, Approaches, and Challenges. *IEEE Dependable and Secure Comp.*, 2(1):2 –19, Mar. 2005.

[3] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proc. of the 9th Annual ACM Symposium on Theory of Computing*, STOC'77, pages 77–90. ACM, 1977.

[4] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *ACM Communication*, 13(6):377–387, June 1970.

[5] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-Preserving Data Publishing: A Survey of Recent Developments. *ACM Comput. Surv.*, 42(4):14:1–14:53, June 2010.

[6] A. Harel, A. Shabtai, L. Rokach, and Y. Elovici. M-score: Estimating the Potential Damage of Data Leakage Incident by Assigning Misuseability Weight. In *Proc. of the 2010 ACM Workshop on Insider Threats*, Insider Threats'10, pages 13–20. ACM, 2010.

[7] A. Harel, A. Shabtai, L. Rokach, and Y. Elovici. Eliciting Domain Expert Misuseability Conceptions. In *Proc. of the 6th Int'l Conference on Knowledge Capture*, K-CAP'11, pages 193–194. ACM, 2011.

[8] A. Harel, A. Shabtai, L. Rokach, and Y. Elovici. M-Score: A Misuseability Weight Measure. *IEEE Trans. Dependable Secur. Comput.*, 9(3):414–428, May 2012.

[9] T. Helleseth and T. Klove. The Number of Cross-Join Pairs in Maximum Length Linear Sequences. *IEEE Transactions on Information Theory*, 37(6):1731–1733, Nov. 1991.

[10] P. A. Laplante. *Dictionary of Computer Science, Engineering and Technology.* CRC Pressl, London,England, 1st edition, 2000.

[11] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-Diversity: Privacy Beyond k-Anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):1–50, Mar. 2007.

[12] A. Motro. An Access Authorization Model for Relational Databases Based on Algebraic Manipulation of View Definitions. In *Proc. of the 5th Int'l Conference on Data Engineering*, pages 339–347. IEEE Computer Society, 1989.

[13] J. Natarajan, S. Shaw, R. Bruchez, and M. Coles. *Pro T-SQL 2012 Programmer's Guide.* Apress, Berlin-Heidelberg, Germany, 3rd edition, 2012.

[14] J. S. Park and J. Giordano. Access Control Requirements for Preventing Insider Threats. In *Proc. of the 4th IEEE Int'l Conference on Intelligence and Security Informatics*, ISI'06, pages 529–534. Springer, 2006.

[15] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending Query Rewriting Techniques for Fine-Grained Access Control. In *Proc. of the 2004 ACM SIGMOD Int'l Conference on Management of Data*, SIGMOD'04, pages 551–562. ACM, 2004.

[16] A. Rosenthal and E. Sciore. View Security as the Basis for Data Warehouse Security. In *CAiSE Workshop on Design and Management of Data Warehouses*, DMDW'2000, pages 5–6. CEUR-WS, 2000.

[17] L. Sweeney. K-Anonymity: A Model For Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, Oct. 2002.

# TrIMPI: A Data Structure for Efficient Pattern Matching on Moving Objects

Tsvetelin Polomski
Christian-Albrechts-University at Kiel
Hermann-Rodewald-Straße 3
24118 Kiel
tpo@is.informatik.uni-kiel.de

Hans-Joachim Klein
Christian-Albrechts-University at Kiel
Hermann-Rodewald-Straße 3
24118 Kiel
hjk@is.informatik.uni-kiel.de

## ABSTRACT

Managing movement data efficiently often requires the exploitation of some indexing scheme. Taking into account the kind of queries issued to the given data, several indexing structures have been proposed which focus on spatial, temporal or spatio-temporal data. Since all these approaches consider only raw data of moving objects, they may be well-suited if the queries of interest contain concrete trajectories or spatial regions. However, if the query consists only of a qualitative description of a trajectory, e.g. by stating some properties of the underlying object, sequential scans on the whole trajectory data are necessary to compute the property, even if an indexing structure is available.
The present paper presents some results of an ongoing work on a data structure for Trajectory Indexing using Motion Property Information (TrIMPI). The proposed approach is flexible since it allows the user to define application-specific properties of trajectories which have to be used for indexing. Thereby, we show how to efficiently answer queries given in terms of such qualitative descriptions. Since the index structure is built on top of ordinary data structures, it can be implemented in arbitrary database management systems.

## Keywords

Moving object databases, motion patterns, indexing structures

## 1. INTRODUCTION AND MOTIVATION

Most index structures for trajectories considered in the literature (e.g. [8]) concentrate on (time dependent) positional data, e.g. R-Tree [9] or TPR*-Tree [17]. There are different approaches (e.g. [1], [12]) exploiting transformation functions on the original data and thereby reducing the indexing overhead through “light versions” of the trajectories to be indexed. In these approaches only stationary data is being handled. In cases where the queries of interest consist of concrete trajectories or polygons covering them, such indexing schemata as well as trajectory compression techniques (e.g. [1], [6], [10], [12], [13]) may be well-suited. However, there are applications [14] where a query may consist only of a qualitative description, e.g. *return all trajectories where the underlying object slowed down (during any time interval) and after that it changed its course*. Obviously, the motion properties *slowdown* and *course alteration* as well as their temporal adjustment can be computed using formal methods. The crucial point is that, even if an indexing structure is used, the stated properties must be computed for each trajectory and this results in sequential scan(s) on the whole trajectory data. Time consuming processing of queries is not acceptable, however, in a scenario where fast reaction on incoming data streams is needed. An example of such a situation with so-called *tracks* computed from radar and sonar data as input is the detection of patterns of skiff movements typical for many piracy attacks [14]. A *track* comprises the position of an object at a time moment and can hold additional information e.g. about its current course and velocity. Gathering the tracks of a single object over a time interval yields its trajectory over this interval.
To address the efficiency problem, we propose an indexing scheme which is not primarily focused on the “time-position data” of trajectories but uses meta information about them instead.
We start with a discussion of related work in Section 2. Section 3 provides some formal definitions on trajectories and their motion properties. In section 4 we introduce the indexing scheme itself and illustrate algorithms for querying it. Section 5 summarizes the present work and outlines our future work.

## 2. RELATED WORK

In this section we provide a short overview on previous contributions which are related to our approach. We start the section by reviewing classical indexing structures for moving objects data. Next to this, we show an approach which is similar in general terms to the proposed one and finally we review literature related to semantical aspects of moving objects.

### 2.1 Indexing of Spatial, Temporal and Spatio-Temporal Data

The moving object databases community has developed several data structures for indexing movement data. According to [8], these structures can be roughly categorized as structures indexing only spatial data, also known as *spatial access methods* (*SAM*); indexing approaches for temporal data, also known as *temporal index structures*; and those which manage both - spatial and temporal data, also known as *spatio-temporal index structures*. One of the first structures developed for SAMs is the well-known R-Tree [9]. Several extensions of R-Trees have been provided over the years, thus yielding a variety of spatio-temporal index structures. An informal schematic overview on these extensions, including also new developments as the HTPR*-Tree [7] can be found in [11]. Since all of the proposed access methods focus mainly on the raw spatio-

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Illmenau, Germany.

temporal data, they are well-suited for queries on history of movement and predicting new positions of moving objects, or for returning most similar trajectories to a given one. If a query consists only of a qualitative description, however, all the proposed indexing structures are of no use.

## 2.2 Applying Dimensionality Reduction upon Indexing - the GEMINI Approach

The overall approach we consider in this work is similar to the GEMINI (GEneric Multimedia INdexIng method) indexing scheme presented in [6]. This approach was originally proposed for time series and has been applied later for other types of data, e.g. for motion data in [16]. The main idea behind GEMINI is to reduce the dimensionality of the original data before indexing. Therefor, representatives of much lower dimensionality are created for the data (trajectory or time series) to be indexed by using an appropriate transform and used for indexing. A crucial result in [6] is that the authors proved that in order to guarantee no false dismissals [12], the exploited transform must retain the distance (or similarity) of the data to be indexed, that is, the distance between representatives should not exceed the distance of the original time series.
In the mentioned approaches, the authors achieve encouraging results on querying most similar trajectories (or time series) to a given one. However, since the representatives of the original data are trajectories or time series, respectively, evaluating a query which only describes a motion behavior would result in the inspection of all representatives.

## 2.3 Semantical Properties of Movement

Semantical properties of movement data have been considered in various works, e.g. in [2], [5], and [15].
The authors of [2] propose a spatio-temporal representation scheme for moving objects in the area of video data. The considered representation scheme distinguishes between spatio-temporal data of trajectories and their topological information, and also utilizes information about distances between pairs of objects. The topological information itself is defined through a set of *topological relations operators* expressing spatial relations between objects over some time interval, including *faraway*, *disjoint*, *meet*, *overlap*, *is-included-by*/*includes* and *same*.
In [5], a comprehensive study on the research that has been carried out on data mining and visual analysis of movement patterns has been provided. The authors propose a conceptual framework for movement behavior of different moving objects. The extracted behavior patterns are classified according to a taxonomy.
In [15], the authors provide some aspects related to a semantic view of trajectories. They show a conceptual approach for how trajectory behaviors can be described by predicates that involve movement attributes and/or semantic annotations. The provided approach is rather informal and considers behavior analysis of moving objects on a general level.

## 3. FORMAL BACKGROUND

This section provides the formal notions as well as the definitions needed throughout the rest of the paper. We start with the term *trajectory* and then direct later our attention to motion properties and patterns.

## 3.1 Trajectories

In our approach we consider the trajectory $\tau_o$ of an object $o$ simply as a function of time which assigns a position to $o$ at any point in time. Since time plays only a role for the determination of temporal causality between the positions of an object, we abstract from "real time" and use any *time domain* instead. A *time domain* is any set which is interval scaled and countably infinite. The first requirement ensures that timestamps can be used for ordering and, furthermore, that the "delay" between two time assignments can be determined. The second requirement ensures that we have an infinite number of "time moments" which can be unambiguously indexed by elements of $\mathbb{N}$. In the following we denote a time domain by T.

Since objects move in a space, we also need a notion for a *spatial domain*. In the following, let S denote the spatial domain. We require that S is equipped with an adequate metric, such as the Euclidean distance (e.g. for $S = \mathbb{R} \times \mathbb{R}$), which allows us to measure the spatial distance between objects.

Having the notions of time and space we can define formally the term *trajectory*.

*Definition 1.* Let T, S and $O$ denote a time domain, a space domain and a set of distinct objects, respectively. Then, *the trajectory $\tau_o$ of an object $o \in O$* is a function $\tau_o : \mathrm{T} \to \mathrm{S}$.

For brevity, we can also write the trajectory of an object $o \in O$ in the form $(o, t_0, s_0), (o, t_1, s_1) \ldots$ for those $t \in \mathrm{T}$ where $\tau_o(t) = s$ is defined. A single element $(o, t_i, s_i)$ is called the *track of object o at time $t_i$*.

## 3.2 Motion Patterns

We consider a motion pattern as a sequence of properties of trajectories which reveal some characteristics of the behavior of the underlying moving objects. Such properties may be expressed through any predicates which are important for the particular analysis, such as *start*, *stop*, *turn*, or *speedup*.

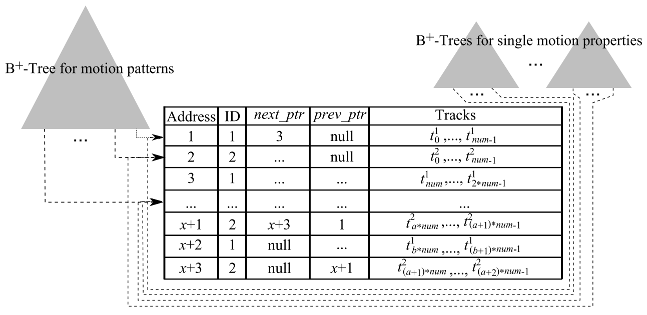
*Definition 2.* Let T be a time domain, $\mathfrak{T}$ be the set of trajectories of an object set $O$ over T, and $I_\mathrm{T}$ be the set of all closed intervals over T. A *motion property* on $\mathfrak{T}$ is a function $p : 2^{\mathfrak{T}} \times I_\mathrm{T} \to \{true, false\}$.

That is, a motion property is fulfilled for a set of trajectories and a certain time interval if the appropriate predicate is satisfied. To illustrate this definition, some examples of motion properties are provided below:

- **Appearance**: Let $t \in \mathrm{T}$. Then we define appear$(\cdot,\cdot)$ as follows: appear$(\{\tau_o\}, [t,t]) = true \Leftrightarrow \forall t' \in \mathrm{T} : \tau_o(t') \neq$ undefined $\to t \leq t'$. That is, an object "appears" only in the "first" moment it is being observed.

- **Speedup**: Let $t_1, t_2 \in \mathrm{T}$ and $t_1 < t_2$. Then speedup$(\cdot,\cdot)$ is defined as follows: speedup$(\{\tau_o\}, [t_1, t_2]) = true \Leftrightarrow v(\tau_o, t_1) < v(\tau_o, t_2) \wedge \forall t \in \mathrm{T} : t_1 \leq t \leq t_2 \to v(\tau_o, t_1) \leq v(\tau_o, t) \leq v(\tau_o, t_2)$ where $v(\tau_o, t)$ denotes the velocity of the underlying moving object $o$ at time $t$. That is, the predicate speedup is satisfied for a trajectory and a time interval if and only if the velocity of the underlying object is increasing in the considered time interval. Note that the increase may not be strictly monotonic.

- **Move away**: Let $t_1, t_2 \in \mathrm{T}$ and $t_1 < t_2$. Then we define: moveaway$(\{\tau_{o_1}, \tau_{o_2}\}, [t_1, t_2]) = true \Leftrightarrow \forall t, t' \in \mathrm{T} : t_1 \leq t < t' \leq t_2 \to dist(\tau_{o_1}, \tau_{o_2}, t) < dist(\tau_{o_1}, \tau_{o_2}, t')$ where the term $dist(\tau_{o_1}, \tau_{o_2}, t)$ denotes the distance between the underlying moving objects $o_1$ and $o_2$ at time $t$. That is, two objects are moving away from each other for a time interval, if their distance is increasing during the considered time interval.


B$^+$-Tree for motion patterns
B$^+$-Trees for single motion properties
...

| Address | ID | *next_ptr* | *prev_ptr* | Tracks |
|---|---|---|---|---|
| 1 | 1 | 3 | null | $t^1_0, \ldots, t^1_{num-1}$ |
| 2 | 2 | ... | null | $t^2_0, \ldots, t^2_{num-1}$ |
| 3 | 1 | ... | ... | $t^1_{num}, \ldots, t^1_{2*num-1}$ |
| ... | ... | ... | ... | ... |
| $x$+1 | 2 | $x$+3 | 1 | $t^2_{a*num}, \ldots, t^2_{(a+1)*num-1}$ |
| $x$+2 | 1 | null | ... | $t^1_{b*num}, \ldots, t^1_{(b+1)*num-1}$ |
| $x$+3 | 2 | null | $x$+1 | $t^2_{(a+1)*num}, \ldots, t^2_{(a+2)*num-1}$ |

**Figure 1: Overview of the index structure**

Using motion properties, a *motion pattern* of a single trajectory or a set of trajectories is defined as *a sequence of motion properties ordered by the time intervals in which they are fulfilled*. It is important to note, that this common definition of a motion pattern allows multiple occurrences of the same motion property in the sequence. In order to get a well-defined notion it has to be required that the time intervals in which the motion properties are fulfilled are disjoint or that meaningful preferences on the motion properties are specified in order to allow ordering in case the time intervals overlap.

## 4. TRAJECTORY INDEXING USING MOTION PROPERTIES

In this section we explain how the proposed index is being created and used. Index creation starts with the determination of the motion pattern of each trajectory to be indexed. For this purpose, the motion predicates specified by the user are computed. The resulting motion patterns are indexed with references to the original trajectories.

The resulting index is schematically depicted in Figure 1. TrIMPI consists mainly of a data structure holding the raw trajectory data, and secondary index structures for maintaining motion patterns. Thereby, we differentiate between indexing single motion properties and indexing motion patterns.
A query to the index can be stated either through a motion pattern or through a concrete trajectory. The index is searched for motion patterns containing the given one or the computed one, respectively. In both cases, the associated trajectories are returned. The following subsections consider the outlined procedures more precisely.

### 4.1 Indexing Trajectory Raw Data

Since the focus of TrIMPI is not on querying trajectories by example, the index structure for the raw trajectory data can be rather simple. For our implementation, we considered a *trajectory record file* as proposed by [3]. This structure (Figure 1) stores trajectories in records of fixed length. The overall structure of the records is as follows

| $ID_o$ | $next\_ptr$ | $prev\_ptr$ | $\{track_0, \ldots, track_{num-1}\}$ |
|---|---|---|---|

.

$ID_o$ denotes the identifier of the underlying moving object, $next\_ptr$ and $prev\_ptr$ are references to the appropriate records holding further parts of the trajectory, and $\{track_0, \ldots, track_{num-1}\}$ is a list of tracks of a predefined fixed length $num$. If a record $r_i$ for a trajectory $\tau_o$ gets filled, a new record $r_j$ is created for $\tau_o$ holding its further tracks. In this case, $next\_ptr_{r_i}$ is set up to point to $r_j$, and $prev\_ptr_{r_j}$ is set up to point to $r_i$.
Using a trajectory record file, the data is not completely clustered, but choosing appropriate record size leads to partial clustering of the trajectory data in blocks. This has the advantage that extracting the complete trajectory requires only loading as much blocks as needed for storing a trajectory.

### 4.2 Indexing Motion Patterns

For the maintenance of motion patterns we consider two cases - single motion properties and sequences of motion properties. Storing single motion properties allows the efficient finding of trajectories which contain the considered motion property. This is advantageous if the searched property is not often satisfied. Thus, for each motion property $p$ a “list” $DBT_p$ holding all trajectories satisfying this property is maintained. As we shall see in Algorithm 4.3, we have to combine such lists and, thus, a simple unsorted list would not be very favourable. Therefore, we implement these lists through B$^+$-Trees (ordered by the trajectory/object identifiers). An evaluation of union and intersection of two B$^+$-Trees with $m$ and $n$ leaves can be performed in $O(m \log \frac{m+n}{m})$[4].
The search for motion patterns with more than one motion property can be conducted through the single $DBT_p$ structures. However, if the query motion pattern is too long, too many intersections of the $DBT_p$ structures will happen and the resulting trajectories will have to be checked for containing properties that match the given order, as well. To overcome this problem, sequences of motion properties are stored in an additional B$^+$-Tree structure $DBT$. The elements of $DBT$ have the form $(p, \tau_o)$ where $p$ is a motion pattern, and $o \in O$. To sort the elements of $DBT$, we apply lexicographical ordering. As a result, sequences with the same prefix are stored consecutively. Thus, storing of motion patterns that are prefixes of other motion patterns can be omitted.

### 4.3 Building the Index

The algorithm for the index creation is quite simple. It consists primarily of the following steps:

- Determine the motion properties for each trajectory $\tau_o$. Consider, if needed, a sliding window or some reduction or segmenting technique as proposed in [1], [6], [10], [12], [13], for example. Generate a list $f$ of the motion properties of $\tau_o$, ordered by their appearance in $\tau_o$.
- Store $\tau_o$ into the trajectory record file.
- Apply Algorithm 4.1 to $f$ to generate access keys relevant for indexing.
- For each generated access key, check whether it is already contained in the index. If this is not the case, store it in the index. Link the trajectory record file entry of $\tau_o$ to the access key.

Algorithm 4.1 is used to generate index keys of a pattern. An index key is any *subpattern* $p' = (p'_j)_{j=0}^{m-1}$ of a pattern $p = (p_i)_{i=0}^{n-1}$ which is defined as follows:

- For each $j \leq m-1$ exists $i \leq n-1$ such that $p'_j = p_i$
- For each $j, k$ such that $0 \leq j < k \leq m-1$ exist $i, l$ such that $0 \leq i < l \leq n-1$ and $p'_j = p_i$ and $p'_k = p_l$.

To generate the list of index keys, algorithm 4.1 proceeds iteratively. At each iteration of the outer loop (lines 3 to 16) the algorithm considers a single element $p$ of the input sequence $f$. On the one hand, $p$ is being added as an index key to the (interim) result (lines 14 and 15) and on the other hand it is being appended as a suffix to each previously generated index key (inner loop - lines 5 to 13). Algorithm 4.1 utilizes two sets whose elements are lists of motion properties - `supplist` and `entries`. The set `supplist` contains at each iteration the complete set of index keys, including those which are prefixes of other patterns. The set `entries` is built in each iteration of the inner loop (lines 5 to 13) by appending the current motion property of the input sequence to any element of `supplist`. Thereby, at line 14 `entries` holds only index keys which are no prefixes of other index keys. Since the resulting lists of index keys are stored in a $B^+$-Tree by applying a lexicographical order, sequences of motion properties which are prefixes of other sequences can be omitted. Therefore, the set `entries` is returned as final result (line 17).

Since the given procedure may result in the computation of up to $2^{k_0}$ different indexing keys for an input sequence with $k_0$ motion properties, a global constant $G$ is used to limit the maximal length of index keys. Using an appropriate value for $G$ leads to no drawbacks for the application. Furthermore, the proposed querying algorithm can handle queries longer than $G$.

**Algorithm 4.1** Building the indexing keys

```
Require: $f$ is a sequence of motion properties
Require: $G$ is the maximal length of sequences to be indexed
 1 function createIndexKeys($f$)
 2     $supplist \leftarrow$ empty set of lists
 3     for all $a \in f$ do
 4         $entries \leftarrow$ empty set of lists
 5         for all $l \in supplist$ do
 6             $new \leftarrow$ empty list
 7             if $|l| \leq G$ then
 8                 $new \leftarrow l$.append($a$)
 9             else
10                 $new \leftarrow l$
11             end if
12             $entries \leftarrow entries \cup \{new\}$
13         end for
14         $entries \leftarrow entries \cup \{[a]\}$
15         $supplist \leftarrow entries \cup supplist$
16     end for
17     return $entries$
18 end function
```

## 4.4 Searching for Motion Patterns

Since the index is primarily considered to support queries on sequences of motion properties, the appropriate algorithm for evaluating such queries given in the following is rather simple. In its "basic" version, query processing is just traversing the index and returning all trajectories referenced by index keys which contain the queried one (as a subpattern). This procedure is illustrated in algorithm 4.2. There are, however, some special cases which have to be taken into account. The first of them considers query sequences which are "too short". As stated in Section 4.2, it can be advantageous to evaluate queries containing only few motion properties by examination of the index structures for single motion properties. To be able to define an application specific notion of "short" queries, we provide besides $G$ an additional global parameter $\alpha$ for which holds $1 \leq \alpha < G$. In algorithm 4.3, which evaluates queries of patterns of arbitrary length, each pattern of length shorter than $\alpha$ is being handled in the described way (lines 3 to 8). It is important that each trajectory of the interim result has to be checked whether it matches the queried pattern (lines 9 to 13).

**Algorithm 4.2** Basic querying of trajectories with a sequence of motion properties

```
Require: s is a sequence of motion properties; $|s| \leq G$
Require: DBT is the index containing motion patterns
 1 function GetEntriesFromDBT(s)
 2     $result \leftarrow \{\tau_o \mid \exists p$ s.t. $s \leq p \wedge (p, \tau_o) \in DBT\}$
 3     return $result$
 4 end function
```

The other special case are queries longer than $G$ (lines 16 to 24). As we have seen in algorithm 4.1, in such cases the index keys are cut to prefixes of length $G$. Thus, the extraction in this case considers the prefix of length $G$ of the query sequence (lines 17) and extracts the appropriate trajectories (line 18). Since these trajectories may still not match the query sequence, e.g. by not fulfilling some of the properties appearing on a position after $G-1$ in the input sequence, an additional check of the trajectories in the interim result is made (lines 19 to 23).

The last case to consider are query sequences with length between $\alpha$ and $G$. In these cases, the index $DBT$ holding the index keys is searched through a call to algorithm 4.2 and the result is returned. Finally, the function `Match` (algorithm 4.4) checks whether a trajectory $\tau_o$ fulfills a pattern $s$. For this purpose, the list of motion properties of $\tau_o$ is being generated (line 2). Thereafter, $s$ and the generated pattern of $\tau_o$ are traversed (lines 5 to 14) so that it can be checked whether the elements of $s$ can be found in the trajectory pattern of $\tau_o$ in the same order. In this case the function `Match` returns `true`, otherwise it returns `false`.

**Algorithm 4.3** Querying trajectories with a sequence of arbitrary length

```
Require: $s$ is a sequence of motion properties
Require: $G$ is the maximal length of stored sequences
Require: $DBT_p$ is the index of the property $p$
Require: $1 \leq \alpha < G$ maximal query length for searching single property indexes
 1 function GetEntries($s$)
 2     $result \leftarrow$ empty set
 3     if $|s| < \alpha$ then
 4         $result \leftarrow \mathfrak{T}$
 5         for all $p \in s$ do
 6             $suppset \leftarrow DBT_p$
 7             $result \leftarrow result \cap suppset$
 8         end for
 9         for all $\tau_o \in result$ do
10             if ! match($\tau_o, s$) then
11                 $result \leftarrow result \backslash \{\tau_o\}$
12             end if
13         end for
14     else if $|s| \leq G$ then
15         $result \leftarrow GetEntriesFromDBT(s)$
16     else
17         $k \leftarrow s[0..G-1]$
18         $result \leftarrow GetEntriesFromDBT(k)$
19         for all $\tau_o \in result$ do
20             if ! match($\tau_o, s$) then
21                 $result \leftarrow result \backslash \{\tau_o\}$
22             end if
23         end for
24     end if
25     return $result$
26 end function
```

## 5. CONCLUSIONS AND OUTLOOK

In this paper we provided some first results of an ongoing work on an indexing structure for trajectories of moving objects called TrIMPI. The focus of TrIMPI lies not on indexing spatio-temporal data but on the exploitation of motion properties of moving objects. For this purpose, we provided a formal notion of motion properties and showed how they form a motion pattern. Furthermore, we showed how these motion patterns can be used to build a meta index. Algorithms for querying the index were also provided. In the next steps, we will finalize the implementation of TrIMPI and perform tests in the scenario of the automatic detection of piracy attacks mentioned in the Introduction. As a conceptual improvement of the work provided in this paper, we consider a flexibilisation of

**Algorithm 4.4** Checks whether a trajectory matches a motion pattern

```
Require: $\tau_o$ is a valid trajectory
Require: $s$ is a sequence of motion properties
1  function MATCH($\tau_o$, $s$)
2      $motion\_properties \leftarrow$ compute the list of motion properties of $\tau_o$
3      $index\_s \leftarrow 0$
4      $index\_props \leftarrow 0$
5      while $index\_props < motion\_properties.length$ do
6          if $motion\_properties[index\_props] = s[index\_s]$ then
7              $index\_s \leftarrow index\_s + 1$
8          else
9              $index\_props \leftarrow index\_props + 1$
10         end if
11         if $index\_s = s.length$ then
12             return true
13         end if
14     end while
15     return false
16 end function
```

the definition of motion patterns including arbitrary temporal relations between motion predicates.

## 6. ACKNOWLEDGMENTS

The authors would like to give special thanks to their former student Lasse Stehnken for his help in implementing TrIMPI.

## 7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In D. B. Lomet, editor, *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms, FODO'93, Chicago, Illinois, USA, October 13-15, 1993*, volume 730 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 1993.

[2] J.-W. Chang, H.-J. Lee, J.-H. Um, S.-M. Kim, and T.-W. Wang. Content-based retrieval using moving objects' trajectories in video data. In *IADIS International Conference Applied Computing*, pages 11–18, 2007.

[3] J.-W. Chang, M.-S. Song, and J.-H. Um. TMN-Tree: New trajectory index structure for moving objects in spatial networks. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1633–1638. IEEE Computer Society, July 2010.

[4] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 743–752, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[5] S. Dodge, R. Weibel, and A.-K. Lautenschütz. Towards a taxonomy of movement patterns. *Information Visualization*, 7(3):240–252, June 2008.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, SIGMOD '94, pages 419–429, New York, NY, USA, 1994. ACM. 472940.

[7] Y. Fang, J. Cao, J. Wang, Y. Peng, and W. Song. HTPR*-Tree: An efficient index for moving objects to support predictive query and partial history query. In L. Wang, J. Jiang, J. Lu, L. Hong, and B. Liu, editors, *Web-Age Information Management*, volume 7142 of *Lecture Notes in Computer Science*, pages 26–39. Springer Berlin Heidelberg, 2012.

[8] R. H. Güting and M. Schneider. *Moving Object Databases*. Data Management Systems. Morgan Kaufmann, 2005.

[9] A. Guttman. R-Trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[10] J. Hershberger and J. Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In P. Bresnahan, editor, *Proceedings of the 5th International Symposium on Spatial Data Handling, SDH'92, Charleston, South Carolina, USA, August 3-7, 1992*, pages 134–143. University of South Carolina. Humanities and Social Sciences Computing Lab, August 1992.

[11] C. S. Jensen. TPR-Tree Successors 2000–2012. `http://cs.au.dk/~csj/tpr-tree-successors`, 2013. Last accessed 24.03.2013.

[12] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal Of Knowledge And Information Systems*, 3(3):263–286, 2001.

[13] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM'01, San Jose, California, USA, 29 November - 2 December 2001*, pages 289–296. IEEE Computer Society, 2001.

[14] T. Polomski and H.-J. Klein. How to Improve Maritime Situational Awareness using Piracy Attack Patterns. 2013. submitted.

[15] S. Spaccapietra and C. Parent. Adding meaning to your steps (keynote paper). In M. Jeusfeld, L. Delcambre, and T.-W. Ling, editors, *Conceptual Modeling - ER 2011, 30th International Conference, ER 2011, Brussels, Belgium, October 31 - November 3, 2011. Proceedings*, ER'11, pages 13–31. Springer, 2011.

[16] Y.-S. Tak, J. Kim, and E. Hwang. Hierarchical querying scheme of human motions for smart home environment. *Eng. Appl. Artif. Intell.*, 25(7):1301–1312, Oct. 2012.

[17] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, editors, *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 790–801. VLDB Endowment, 2003.

# Complex Event Processing in Wireless Sensor Networks

Omran Saleh
Faculty of Computer Science and Automation
Ilmenau University of Technology
Ilmenau, Germany
omran.saleh@tu-ilmenau.de

## ABSTRACT

Most of the WSN applications need the number of sensor nodes deployed to be in order of hundreds, thousands or more to monitor certain phenomena and capture measurements over a long period of time. The large volume of sensor networks would generate continuous streams of raw events[1] in case of centralized architecture, in which the sensor data captured by all the sensor nodes is sent to a central entity.

In this paper, we describe the design and implementation of a system that carries out complex event detection queries inside wireless sensor nodes. These queries filter and remove undesirable events. They can detect complex events and meaningful information by combining raw events with logical and temporal relationship, and output this information to external monitoring application for further analysis. This system reduces the amount of data that needs to be sent to the central entity by avoiding transmitting the raw data outside the network. Therefore, it can dramatically reduce the communication burden between nodes and improve the lifetime of sensor networks.

We have implemented our approach for the TinyOS Operating System, for the TelosB and Mica2 platforms. We conducted a performance evaluation of our method comparing it with a naive method. Results clearly confirm the effectiveness of our approach.

## Keywords

Complex Event Processing, Wireless Sensor Networks, In-network processing, centralized processing, Non-deterministic Finite state Automata

## 1. INTRODUCTION

Wireless sensor networks are defined as a distributed and cooperative network of devices, denoted as sensor nodes that are densely deployed over a region especially in harsh environments to gather data for some phenomena in this monitored region. These nodes can sense the surrounding environment and share the information with their neighboring nodes. They are gaining adoption on an increasing scale for tracking and monitoring purposes. Furthermore, sensor nodes are often used in control purposes. They are capable of performing simple processing.

[1]The terms data, events and tuples are used interchangeably.

25th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Ilmenau, Germany.

In the near future, it is prospective that wireless sensor networks will offer and make conceivable a wide range of applications and emerge as an important area of computing. WSN technology is exciting with boundless potential for various application areas. They are now found in many industrial and civilian application areas, military and security applications, environmental monitoring, disaster prevention and health care applications, etc.

One of the most important issues in the design of WSNs is energy efficiency. Each node should be as energy efficient as possible. Processing a chunk of information is less costly than wireless communication; the ratio between them is commonly supposed to be much smaller than one [19]. There is a significant link between energy efficiency and superfluous data. The sensor node is going to consume unnecessary energy for the transmission of superfluous data to the central entity, which means minimizing the energy efficiency.

Furthermore, traditional WSN software systems do not apparently aim at efficient processing of continuous data or event streams. According to previous notions, we are looking for an approach that makes our system gains high performance and power saving via preventing the generation and transmission of needless data to the central entity. Therefore, it can dramatically reduce the communication burden between nodes and improve the lifetime of sensor networks. This approach takes into account the resource limitations in terms of computation power, memory, and communication. Sensor nodes can employ their processing capabilities to perform some computations. Therefore, an in-network complex event processing [2] based solution is proposed.

We have proposed to run a complex event processing engine inside the sensor nodes. CEP engine is implemented to transform the raw data into meaningful and beneficial events that are to be notified to the users after detecting them. It is responsible for combining primitive events to identify higher level complex events. This engine provides an efficient Non-deterministic Finite state Automata (NFA) [1] based implementation to lead the evaluation of the complex event queries where the automaton runs as an integral part of the in-network query plan. It also provides the theoretical basis of CEP as well as supports us with particular

[2]CEP is discussed in reference [15]

operators (conjunction, negation, disjunction and sequence operators, etc.).

Complex event processing over data stream has increasingly become an important field due to the increasing number of its applications for wireless sensor networks. There have been various event detection applications proposed in the WSNs, e.g. for detecting eruptions of volcanoes [18], forest fires, and for the habitat monitoring of animals [5]. An increasing number of applications in such networks is confronted with the necessity to process voluminous data streams in real time fashion.

The rest of the paper is organized as follows: section 2 provides an overview of the naive approaches for normal data and complex event processing in WSNs. Related works are briefly reviewed in section 3. Then we introduce the overall system architecture in order to perform complex event processing in sensor networks in section 4. Section 5 discusses how to create logical query plans to evaluate sensor portion queries. Section 6 explains our approach and how queries are implemented by automata. In section 7, the performance of our system is evaluated using a particular simulator. Finally, section 8 presents our concluding remarks and future works.

## 2. NAIVE APPROACHES IN WSNS

The ideas behind naive approaches which are definitely different from our approach lie in the processing of data as the central architectural concept. For **normal sensor data processing**, the centralized approach proceeds in two steps; the sensor data captured by all the sensor nodes is sent to the sink node and then routed to the central server (base station) where it is stored in centralized database. High volume data are arriving at the server. Subsequently, query processing takes place on this database by running queries against stored data. Each query executes one time and returns a set of results.

Another approach which adopts the idea of centralized architecture is the use of a central data stream management system (DSMS), which simply takes the sensor data stream as input source. Sending all sensor readings to DSMS is also an option for WSN data processing. DSMS is defined as a system that manages a data stream, executes a continuous query against a data stream and supports on-line analysis of rapidly changing data streams [10]. Traditional stream processing systems such as Aurora [2], NiagraCQ [7], and AnduIN [12] extend the relational query processing to work with stream data. Generally the select, project, join and aggregate operations are supported in these stream systems.

The naive approach for **Complex Event Processing** in WSNs is similar to the central architectural idea of normal data processing, but instead of using traditional database and data stream engine, CEP uses a dedicated engine for processing complex events such as Esper [8], SASE [11] and Cayuga [4], in which sensor data or events streams need to be filtered, aggregated, processed and analyzed to find the events of interest and identify some patterns among them, finally take actions if needed.

Reference [11] uses SASE in order to process RFID stream data for a real-world retail management scenario. Paper [3] demonstrates the use of Esper engine for object detection tracking in sensor networks. All the aforementioned engines use some variant of a NFA model to detect the complex event. Moreover, there are many CEP engines in the field of active databases. Most of the models in these engines are based on fixed data structures such as tree, graph, finite automaton or petri net. The authors of [6] used a tree based model. Paper [9] used petri net based model to detect complex events from active database. Reference [17] used Timed Petri-Net (TPN) to detect complex events from RFID stream.

## 3. RELATED WORKS

It is preferable to perform **In-Network Processing** inside sensor network to reduce the transmission cost between neighboring nodes. This concept is proposed by several systems such as TinyDB [16], and Cougar [19]. Cougar project applies a database system concept to sensor networks. It uses the declarative queries that are similar to SQL to query sensor nodes. Additionally, sensor data in cougar is considered like a "virtual" relational database. Cougar places on each node an additional query layer that lies between the network and application layers which has the responsibility of in-network processing. This system generates one plan for the leader node to perform aggregation and send the data to a sink node. Another plan is generated for non-leader nodes to measure the sensors status. The query plans are disseminated to the query layers of all sensor nodes. The query layer will register the plan inside the sensor node, enable desired sensors, and return results according to this plan.

TinyDB is an acquisitional query processing system for sensor networks which maintains a single, infinitely-long virtual database table. It uses an SQL-like interface to ask for data from the network. In this system, users specify the data they want and the rate at which the data should be refreshed, and the underlying system would decide the best plan to be executed. Several in-network aggregation techniques have been proposed in order to extend the life time of sensor network such as tree-based aggregation protocols i.e., directed diffusion.

Paper [13] proposes a framework to detect complex events in wireless sensor networks by transforming them into subevents. In this case, the sub-events can easily be detected by sensor nodes. Reference [14] splits queries into server and node queries, where each query can be executed. The final results from both sides are combined by the results merger. In [20], symbolic aggregate approximation (SAX) is used to transform sensor data to symbolic representations. To detect complex events, a distance metric for string comparison is utilized. These papers are the closer works to our system.

Obviously, there is currently little work into how the idea of in-network processing can be extended and implemented to allow more complex event queries to be resolved within the network.

## 4. SYSTEM ARCHITECTURE

We have proposed a system architecture in which collected data at numerous, inexpensive sensor nodes are processed locally. The resulting information is transmitted to larger, more capable and more expensive nodes for further analysis and processing through specific node called sink node.

The architecture has three main parts that need to be modified or created to make our system better suited to queries over sensor nodes: 1- **Server side**: queries will be originated at server side and then forwarded to the nearest sink node. Additionally, this side mainly contains an application that runs on the user's PC (base station). Its main purpose is to collect the results stream over the sensor network and display them. Server side application can offer more functions i.e., further filtering for the collected data, perform joining on sensor data, extract, save, manage, and search the semantic information and apply further complex event processing on incoming events after processing them locally in sensor nodes. Because sensor data can be considered as a data stream, we proposed to use a data stream management system to play a role of server side, for that we selected AnduIN data stream engine. 2- **Sink side**: sink node (also known as root or gateway node) is one of the motes in the network which communicates with the base station directly, all the data collected by sensors is forwarded to a sink node and then to server side. This node will be in charge of disseminating the query down to all the sensor nodes in the network that comes from server side. 3- **Node side**: in this side, we have made huge changes to the traditional application which runs on the nodes themselves to enable database manner queries involving filters, aggregates, complex event processing operator (engine) and other operators to be slightly executed within sensor networks. These changes are done in order to reduce communication costs and get useful information instead of raw data.

When combining on-sensor portions of the query with the server side query, most of the pieces of the sensor data query are in place. This makes our system more advanced.

## 5. LOGICAL PLAN

Each and every sensor node of a network generates tuples. Every tuple may consist of information about the node id, and sensor readings. Query plan can specify the tuples flow between all necessary operators and a precise computation plan for each sensor node. Figure 1 (lower plan) illustrates how our query plan can be employed. It corresponds to an acyclic directed graph of operators. We assume the dataflow being upward. At the bottom, there is a homogeneous data source which generates data tuples that must be processed by operators belonging to query plans. Tuples are flowed through intermediate operators composed in the query graph. The operators perform the actual processing and eventually forward the data to the sink operator for transmitting the resulting information to the server side (base station). These operators adopt publish/subscribe mechanism to transfer tuples from one operator to next operator.

We differ between three different types of operators within a query graph [12]: 1- **Source operator**: produces tuples and transfers them to other operators. 2- **Sink operator**: receives incoming tuples from other operators. 3- **Inner operators**: receive incoming tuples from source operator, process them, and transfer the result to sink operator or other inner operators.

A query plan consists of one source at the bottom of a logical query graph, several inner operators, and one sink at the top and the tuples are flowing strictly upward. In our system, we have extended this plan to give the system the capability to perform the complex event processing and detecting by adding new operators. We have separated the mechanism for detecting complex events from the rest of normal processing side. We have a particular component working as an extra operator or engine within the main process, as we can see from figure 1 (upper plan). The detection mechanism takes as input primitive events from lower operators and detects occurrences of composite events which are used as an output to the rest of the system.

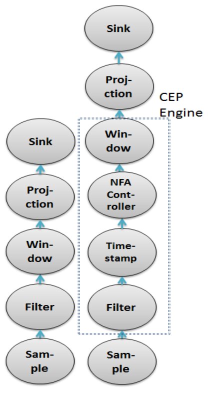


**Figure 1: Logical query plan**

## 6. IN-NETWORK CEP SYSTEM

Various applications including WSNs require the ability to handle complex events among apparently unrelated events and find interesting and/or special patterns. Users want to be notified immediately as soon as these complex events are detected. Sensor node devices generate massive sensor data streams. These streams generate a variety of primitive events continuously. The continuous events form a sequence of primitive events, and recognition of the sequence supplies us a high level event, which the users are interested in.

Sensor event streams have to be automatically filtered, processed, and transformed into significative information. In non-centralized architecture, CEP has to be performed as close to real time as possible (inside the node). The task of identifying composite events from primitive ones is performed by the Complex Event Processing engine. CEP engine provides the runtime to perform complex event processing where they accept queries provided by the user, match those queries against continuous event streams, and trigger an event or an execution when the conditions specified in the queries have been satisfied. The idea of this concept is close to Event-Condition-Action (ECA) concept in conventional database systems where an action has to be carried out in response to an event and one or more conditions are satisfied.

Each data tuple from the sensor node is viewed as a primitive event and it has to be processed inside the node. We have proposed an event detection system that specifically targets applications with limited resources, such in our system. There are four phases for complex event processing in our in-network model: NFA creation, Filtering, Sequence scan and Response as shown in figure 2.

### 6.1 NFA Creation Phase

The first phase is NFA creation. NFA's structure is created by the translation from the sequence pattern through mapping the events to NFA states and edges, where the conditions of the events (generally called event types) are associated with edges. For pattern matching over sensor node streams, NFA is employed to represent the structure of an event sequence. For a concrete example, consider the query

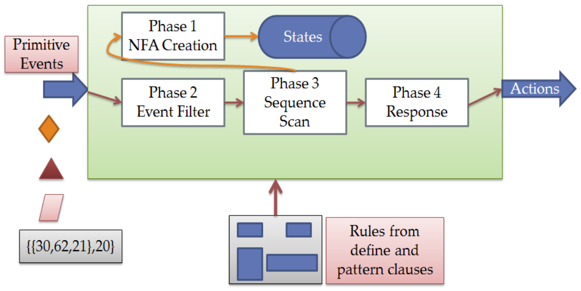


**Figure 2: CEP Phases**

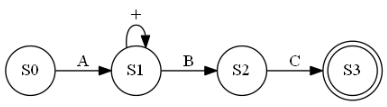


**Figure 3: NFA for SEQ(A a, B+ b, C c)**

pattern: **SEQ(A a, B+ b, C c)**[3]. Figure 3 shows the NFA created for the aforementioned pattern **(A, B+, C)**, where state $S0$ is the starting state, state $S1$ is for the successful detection of an $A$ event, state $S2$ is for the detection of a $B$ event after event $A$, also state $S3$ is for the detection of a $C$ event after the $B$ event. State $S1$ contains a self-loop with the condition of a $B$ event. State $S3$ is the accepting state, reaching this state indicates that the sequence is detected.

## 6.2 Filtering Phase

The second phase is to filter primitive events at early stage, generated by sensor nodes. Sensor nodes cannot understand whether a particular event is necessary or not. When additional conditions are added to the system, possible event instances might be pruned at the first stage.

After filtering, timestamp operator will add the occurrence time of the event $t$. A new operator is designed for adding a timestamp $t$ to the events (tuples) before entering the complex event processing operator. We can notice that from figure 1. The timestamp attribute value of an event $t$ records the reading of a clock in the system in which the event was created, in this case it can reflect the true order of the occurrences of primitive events.

## 6.3 Sequence Scan Phase

The third phase is sequence scan to detect a pattern match. We have three modes state the way in which events may contribute to scan a sequence: UNRESTRICTED, RECENT and FIRST. Every mode has a different behavior. The selection between them depends on the users and the application domain. These modes have advantages and disadvantages. We will illustrate them below.

In the UNRESTRICTED mode, each start event $e$, which allows a sequence to move from the initial state to the next state, starts a separate sequence detection. In this case any event occurrence combination that matches the definition of the sequence can be considered as an output. By using this mode, we can get all the possibilities of event combination which satisfy the sequence. When the sequence is created, it

[3]Notice: In this paper, we are going to only focus on sequence operator **SEQ** because of the limited number of pages.

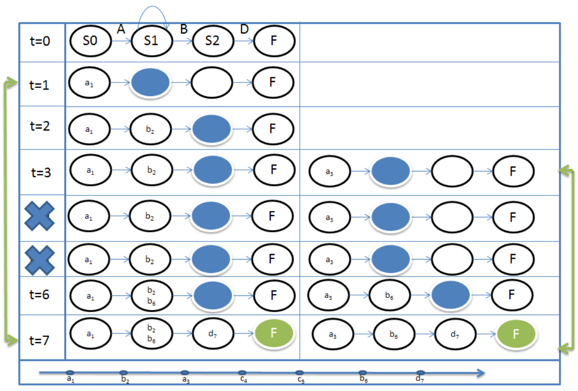


**Figure 4: Sequence Scan for SEQ (A, B+, D) within 6 Time Unit Using UNRESTRICTED Mode**

is waiting for the arrival of events in its starting state. Once a new instance event $e$ arrives, the sequence scan responds as follows: 1- It checks whether the type of instance (from attributes) and occurrence time of $e$ satisfy a transition for one of the logical existing sequences. If not, the event is directly rejected. 2- If yes, $e$ is registered in the system (the registration is done in the sliding window) and the sequence advances to next state. 3- If $e$ allows for a sequence to move from the starting state to next state, the engine will create other logical sequence to process further incoming events while keeping the original sequence in its current state to receive new event. Therefore, multiple sequences work on the events at the same time. 4- Delete some sequences when their last received items are not within a time limit. It becomes impossible for them to proceed to the next state since the time limits for future transitions have already expired.

Next, we use an example to illustrate how UNRESTRICTED sequence scan works. Suppose we have the following pattern[4] **SEQ (A, B+, D)** and sequence of events (tuples) presented as **[a1, b2, a3, c4, c5, b6, d7 ...]** within 6 time unit. Figure 4 shows, step by step, how the aforementioned events are processed. Once the sequence has reached the accepting state ($F$), the occurrences of **SEQ (A, B+, D)** will be established at : **{{a1, b2, d7 }, {a1, b6, d7 }, {a3, b6, d7 }}**.

The drawback of this mode is the use of high storage to accumulate all the events that participate in the combinations in addition to computation overhead for the detection. It consumes more energy. On other hand, it gives us all the possibilities of event combination which can be used (e.g. for further analysis). In our system, we only output one of these possibilities to reduce transmission cost overhead. All registered events are stored in a sliding window. Once the overflow has occurred, the candidate events would be the newest registered ones from the first sequence. The engine will continue to replace the events from the first sequence as long as there is no space. When the initial event (first event in the first sequence combination) is replaced, the engine starts the replacement from the second sequence and so on. The engine applies this replacement policy to ensure that the system still has several sequences to detect a composite event, because replacing the initial events would destroy the

[4]The terms complex event, composite event, pattern and sequence are used interchangeably.

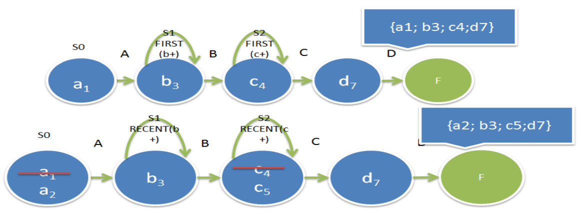


**Figure 5: First and Recent Modes**

whole sequence.

In the FIRST mode, the earliest occurrence of each contributing event type is used to form the composite event output. Only the first event from a group of events which have the same type advances the sequence to the next state. In this mode, we have just one sequence in the system. The automaton engine will examine every incoming instance $e$, whether the type of it and occurrence time of $e$ satisfy a transition from the current state to next state. If it is, the sequence will register the event in the current state and advance to next state. If not, the event is directly rejected. Suppose we have the following pattern **SEQ (A, B+, C+, D)** and sequence of tuples presented as **[a1, a2, b3, c4, c5, b6, d7 ...]** within 6 time unit. The result as shown in the upper part of figure 5 .

In the RECENT mode (as the lower part of figure 5 which has FIRST pattern and the same sequence of tuples), the most recent event occurrences of contributing event types are used to form the composite event. In RECENT mode, once an instance satisfies the condition and timing constraint to jump from a state to next state, the engine will stay in the current state unlike FIRST mode. This mode tries to find the most recent instance from consecutive instances for that state before moving to next state. When $a1$ enters the engine. It satisfies the condition to move from $S0$ to $S1$. The engine registers it, stays in $S0$ and does not jump to the next state. Perhaps the new incoming instance is more recent from the last one in the current state.

The advantages of FIRST and RECENT modes are the use of less storage to accumulate all the events that participate in the combinations. Only a few events will be registered in the system in addition to low computation overhead for the detection. They consume less energy. Unlike UNRESTRICTED, they do not give all possible matches.

## 6.4 Response Phase

Once an accepting state $F$ is reached by the engine, the engine should immediately output the event sequence. This phase is responsible for preparing the output sequence to pass it to the sink operator. The output sequence depends on the mode of the scan. This phase will start to create the response by reading the sliding window contents. In case of FIRST and RECENT modes, the sliding window contains only the events which contribute in sequence detection. In UNRESTRICTED mode, the engine randomly selects a combination of events which matches the pattern in order to reduce transmission cost.

## 7. EVALUATION

We have completed an initial in-network complex event processing implementation. All the source code, implementing the in-network complex event processing techniques as well as base station functionality, is written in TinyOS. Our code runs successfully on both real motes and the TinyOS Avrora simulator. The aim of the proposed work is to compare the performance of our system, in-network processor which includes complex event engine in comparison with centralized approach in wireless sensor networks and to assess the suitability of our approach in an environment where resources are limited. The comparison would be done in terms of energy efficiency (amount of energy consumed) and the number of messages transmitted per particular interval, in the entire network. The experiment was run for varying the SEQ length. We started with length 2 then 3 and finally 5. Simulations were run for 60 seconds with one event per second. The performance for different SEQ lengths and different modes with a network of 75 nodes is shown in figure 6. The centralized architecture led to higher energy consumption because sensor nodes transmitted events to the sink node at regular periods. In our system, we used in-network complex event processing to decrease the number of transmissions of needless events at each sensor node. What we can notice from figure 6 is summarized as: 1- By increasing the SEQ length in our approach, the RAM size is increased while energy consumption is reduced. The reason is: the transmission will not occur until the sequence reaches the accepting state, few events (tuples) will be relatively satisfied. Hence, the number of transmissions after detections will be decreased. 2- FIRST is a little bit better than RECENT, and both of them are better than UNRESTRICTED in energy consumption. The gap between them is resulting from processing energy consumption, that is because UNRESTRICTED needs more processing power while the other needs less, as shown in figure 6.

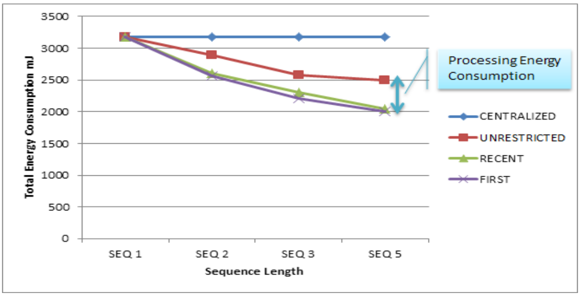


**Figure 6: Total Energy Consumption**

Figure 7 shows the radio energy consumption for each sensor node and the total number of messages when SEQ length was 3. The nodes in the centralized architecture sent more messages than our approach (nearly three times more). Hence, it consumed more radio energy. Additionally, the gateway nodes consumed more radio energy due to receiving and processing the messages from other sensor nodes. In a 25 nodes network, the centralized approach consumed energy nearly 4203mJ in sink side, while our approach consumed around 2811mJ. Thus, our system conserved nearly 1392mJ (33% of the centralized approach) of the energy. In our architecture, the number of transmissions was reduced. Therefore, the radio energy consumption is reduced not only at the sensor nodes but also at the sink nodes.

## 8. CONCLUSIONS

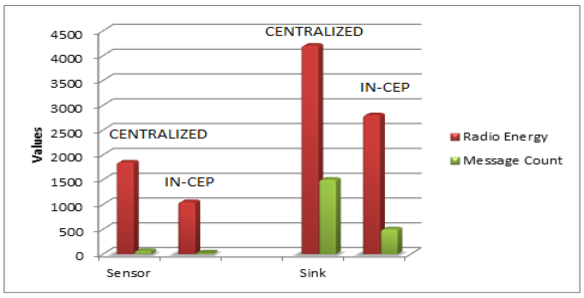


**Figure 7: Energy Consumption vs. Radio Message**

Sensor networks provide a considerably challenging programming and computing environment. They require advanced paradigms for software design, due to their characteristics such as limited computational power, limited memory and battery power which WSNs suffer from. In this paper, we presented our system, an in-network complex event processing, a system that efficiently carries out complex event queries inside network nodes.

We have proposed an engine to allow the system to detect complex events and valuable information from primitive events.

We developed a query plan based approach to implement the system. We provided the architecture to collect the events from sensor network, this architecture includes three sides; sensor side to perform in-network complex event processing, sink side to deliver the events from the network to AnduIN server side which has the responsibility to display them and perform further analysis.

We demonstrated the effectiveness of our system in a detailed performance study. Results obtained from a comparison between centralized approach and our approach confirms that our in-network complex event processing in small-scale and large-scale sensor networks has shown to increase the lifetime of the network. We plan to continue our research to build distributed in-network complex event processing, in which each sensor node has a different complex event processing plan and can communicate directly between them to detect complex events.

## 9. REFERENCES

[1] Nondeterministic finite automaton. `http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton`.

[2] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J.-h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: a data stream management system. In *ACM SIGMOD Conference*, page 666, 2003.

[3] R. Bhargavi, V. Vaidehi, P. T. V. Bhuvaneswari, P. Balamuralidhar, and M. G. Chandra. Complex event processing for object tracking and intrusion detection in wireless sensor networks. In *ICARCV*, pages 848–853. IEEE, 2010.

[4] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White. Cayuga: a high-performance event processing engine. In *ACM SIGMOD*, pages 1100–1102, New York, NY, USA, 2007. ACM.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. *SIGCOMM Comput. Commun. Rev.*, 31(2 supplement):20–41, Apr. 2001.

[6] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: semantics, contexts and detection. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 606–617, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *ACM SIGMOD*, pages 379–390, New York, NY, USA, 2000. ACM.

[8] EsperTech. Event stream intelligence: Esper & NEsper. `http://www.esper.codehaus.org/`.

[9] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system, 1993.

[10] V. Goebel and T. Plagemann. Data stream management systems - a technology for network monitoring and traffic analysis? In *ConTEL 2005*, volume 2, pages 685–686, June 2005.

[11] D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: complex event processing over streams (Demo). In *CIDR*, pages 407–411, 2007.

[12] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K. Sattler. Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIN, distributed and parallel databases. *Distributed and Parallel Databases*, 29(1):151–183, Jan. 2011.

[13] Y. Lai, W. Zeng, Z. Lin, and G. Li. LAMF: framework for complex event processing in wireless sensor networks. In *2nd International Conference on (ICISE)*, pages 2155–2158, Dec. 2010.

[14] P. Li and W. Bingwen. Design of complex event processing system for wireless sensor networks. In *NSWCTC*, volume 1, pages 354–357, Apr. 2010.

[15] D. C. Luckham. *The power of events*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[16] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, Mar. 2005.

[17] J. Xingyi, L. Xiaodong, K. Ning, and Y. Baoping. Efficient complex event processing over RFID data stream. In *IEEE/ACIS*, pages 75–81, May 2008.

[18] X. Yang, H. B. Lim, T. M. Özsu, and K. L. Tan. In-network execution of monitoring queries in sensor networks. In *ACM SIGMOD*, pages 521–532, New York, NY, USA, 2007. ACM.

[19] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, Sept. 2002.

[20] M. Zoumboulakis and G. Roussos. Escalation: complex event detection in wireless sensor networks. In *EuroSSC*, pages 270–285, 2007.

# XQuery processing over NoSQL stores

Henrique Valer
University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
valer@cs.uni-kl.de

Caetano Sauer
University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
csauer@cs.uni-kl.de

Theo Härder
University of Kaiserslautern
P.O. Box 3049
67653 Kaiserslautern, Germany
haerder@cs.uni-kl.de

## ABSTRACT

Using NoSQL stores as storage layer for the execution of declarative query processing using XQuery provides a high-level interface to process data in an optimized manner. The term NoSQL refers to a plethora of new stores which essentially trades off well-known ACID properties for higher availability or scalability, using techniques such as eventual consistency, horizontal scalability, efficient replication, and schema-less data models. This work proposes a mapping from the data model of different kinds of NoSQL stores—key/value, columnar, and document-oriented—to the XDM data model, thus allowing for standardization and querying NoSQL data using higher-level languages, such as XQuery. This work also explores several optimization scenarios to improve performance on top of these stores. Besides, we also add updating semantics to XQuery by introducing simple CRUD-enabling functionalities. Finally, this work analyzes the performance of the system in several scenarios.

## Keywords

NoSQL, Big Data, key/value, XQuery, ACID, CAP

## 1. INTRODUCTION

We have seen a trend towards specialization in database markets in the last few years. There is no more one-size-fits-all approach when comes to storing and dealing with data, and different types of DBMSs are being used to tackle different types of problems. One of these being the *Big Data* topic.

It is not completely clear what Big Data means after all. Lately, it is being characterized by the so-called 3 V's: *volume*—comprising the actual size of data; *velocity*—comprising essentially a time span in which data data must be analyzed; and *variety*—comprising types of data. Big Data applications need to understand how to create solutions in these data dimensions.

RDBMS have had problems when facing Big Data applications, like in web environments. Two of the main reasons for that are scalability and flexibility. The solution RDBMS provide is usually twofold: either (i) a horizontally-scalable architecture, which in database terms generally means giving up joins and also complex multi-row transactions; or (ii) by using parallel databases, thus using multiple CPUs and disks in parallel to optimize performance. While the latter increases complexity, the former just gives up operations because they are too hard to implement in distributed environments. Nevertheless, these solutions are neither scalable nor flexible.

NoSQL tackles these problems with a mix of techniques, which involves either weakening ACID properties or allowing more flexible data models. The latter is rather simple: some scenarios—such as web applications—do not conform to a rigid relational schema, cannot be bound to the structures of a RDBMS, and need flexibility. Solutions exist, such as using XML, JSON, pure key/value stores, etc, as data model for the storage layer. Regarding the former, some NoSQL systems relax consistency by using mechanisms such as multi-version concurrency control, thus allowing for eventually consistent scenarios. Others support atomicity and isolation only when each transaction accesses data within some convenient subset of the database data. Atomic operations would require some distributed commit protocol—like two-phase commit—involving all nodes participating in the transaction, and that would definitely not scale. Note that this has nothing to do with SQL, as the acronym NoSQL suggests. Any RDBMS that relaxes ACID properties could scale just as well, and keep SQL as querying language.

Nevertheless, when it comes to performance, NoSQL systems have shown some interesting improvements. When considering update- and lookup-intensive OLTP workloads—scenarios where NoSQL are most often considered—the work of [13] shows that the total OLTP time is almost evenly distributed among four possible overheads: *logging*, *locking*, *latching*, and *buffer management*. In essence, NoSQL systems improve locking by relaxing atomicity, when compared to RDBMS.

When considering OLAP scenarios, RDBMS require rigid schema to perform usual OLAP queries, whereas most NoSQL stores rely on a brute-force processing model called *MapReduce*. It is a linearly-scalable programming model for processing and generating large data sets, and works with any data format or shape. Using MapReduce capabilities, parallelization details, fault-tolerance, and distribution aspects are transparently offered to the user. Nevertheless, it requires implementing queries from scratch and still suffers from the lack of proper tools to enhance its querying capa-

24th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

bilities. Moreover, when executed atop raw files, the processing is inefficient. NoSQL stores provide this structure, thus one could provide a higher-level query language to take full advantage of it, like Hive [18], Pig [16], and JAQL [6].

These approaches require learning separated query languages, each of which specifically made for the implementation. Besides, some of them require schemas, like Hive and Pig, thus making them quite inflexible. On the other hand, there exists a standard that is flexible enough to handle the offered data flexibility of these different stores, whose compilation steps are directly mappable to distributed operations on MapReduce, and is been standardized for over a decade: XQuery.

## Contribution

Consider employing XQuery for implementing the large class of query-processing tasks, such as aggregating, sorting, filtering, transforming, joining, etc, on top of MapReduce as a first step towards standardization on the realms of NoSQL [17]. A second step is essentially to incorporate NoSQL systems as storage layer of such framework, providing a significant performance boost for MapReduce queries. This storage layer not only leverages the storage efficiency of RDBMS, but allows for pushdown projections, filters, and predicate evaluations to be done as close to the storage level as possible, drastically reducing the amount of data used on the query processing level.

This is essentially the contribution of this work: allowing for NoSQL stores to be used as storage layer underneath a MapReduce-based XQuery engine, Brackit[**?**]—a generic XQuery processor, independent of storage layer. We rely on Brackit's MapReduce-mapping facility as a transparently distributed execution engine, thus providing scalability. Moreover, we exploit the XDM-mapping layer of Brackit, which provides flexibility by using new data models. We created three XDM-mappings, investigating three different implementations, encompassing the most used types of NoSQL stores: *key/value*, *column-based*, and *document-based*.

The remainder of this paper is organized as follows. Section 2 introduces the NoSQL models and their characteristics. Section 3 describes the used XQuery engine, Brackit, and the execution environment of XQuery on top of the MapReduce model. Section 4 describes the mappings from various stores to XDM, besides all implemented optimizations. Section 5 exposes the developed experiments and the obtained results. Finally, Section 6 concludes this work.

## 2. NOSQL STORES

This work focuses on three different types of NoSQL stores, namely *key/value*, *columnar*, and *document-oriented*, represented by Riak [14], HBase[11], and MongoDB[8], respectively.

Riak is the simplest model we dealt with: a pure key/-value store. It provides solely read and write operations to uniquely-identified values, referenced by key. It does not provide operations that span across multiple data items and there is no need for relational schema. It uses concepts such as *buckets*, *keys*, and *values*. Data is stored and referenced by bucket/key pairs. Each bucket defines a virtual key space and can be thought of as tables in classical relational databases. Each key references a unique value, and there are no data type definitions: *objects* are the only unit of data storage. Moreover, Riak provides automatic load balancing and data replication. It does not have any relationship between data, even though it tries by adding link between key/value pairs. It provides the most flexibility, by allowing for a per-request scheme on choosing between availability or consistency. Its distributed system has no master node, thus no single point of failure, and in order to solve partial ordering, it uses *Vector Clocks* [15].

HBase enhances Riak's data model by allowing columnar data, where a table in HBase can be seen as a map of maps. More precisely, each key is an arbitrary string that maps to a row of data. A row is a map, where columns act as keys, and values are uninterpreted arrays of bytes. Columns are grouped into column families, and therefore, the full key access specification of a value is through column family concatenated with a column—or using HBase notation: a *qualifier*. Column families make the implementation more complex, but their existence enables fine-grained performance tuning, because (i) each column family's performance options are configured independently, like read and write access, and disk space consumption; and (ii) columns of a column family are stored contiguously in disk. Moreover, operations in HBase are atomic in the row level, thus keeping a consistent view of a given row. Data relations exist from column family to qualifiers, and operations are atomic on a per-row basis. HBase chooses consistency over availability, and much of that reflects on the system architecture. Auto-sharding and automatic replication are also present: shardling is automatically done by dividing data in regions, and replication is achieved by the master-slave pattern.

MongoDB fosters functionality by allowing more RDBMS-like features, such as secondary indexes, range queries, and sorting. The data unit is a *document*, which is an ordered set of keys with associated values. *Keys* are strings, and *values*, for the first time, are not simply objects, or arrays of bytes as in Riak or HBase. In MongoDB, values can be of different data types, such as strings, date, integers, and even embedded documents. MongoDB provides *collections*, which are grouping of documents, and *databases*, which are grouping of collections. Stored documents do not follow any predefined schema. Updates within a single document are transactional. Consistency is also taken over availability in MongoDB, as in HBase, and that also reflects in the system architecture, that follows a master-worker pattern.

Overall, all systems provide scaling-out, replication, and parallel-computation capabilities. What changes is essentially the data-model: Riak seams to be better suited for problems where data is not really relational, like logging. On the other hand, because of the lack of scan capabilities, on situations where data querying is needed, Riak will not perform that well. HBase allows for some relationship between data, besides built-in compression and versioning. It is thus an excellent tool for indexing web pages, which are highly textual (thus benefiting from compression), as well as inter-related and updatable (benefiting from built-in versioning). Finally, MongoDB provides documents as granularity unit, thus fitting well when the scenario involves highly-variable or unpredictable data.

## 3. BRACKIT AND MAPREDUCE

Several different XQuery engines are available as options for querying XML documents. Most of them provide either (i) a lightweight application that can perform queries on documents, or collections of documents, or (ii) an XML database that uses XQuery to query documents. The former lacks any sort of storage facility, while the latter is just not flexible enough, because of the built-in storage layer. Brackit[1] provides intrinsic flexibility, allowing for different storage levels to be "plugged in", without lacking the necessary performance when dealing with XML documents [5]. By dividing the components of the system into different modules, namely *language*, *engine*, and *storage*, it gives us the needed flexibility, thus allowing us to use any store for our storage layer.

## Compilation

The compilation process in Brackit works as follows: the *parser* analyzes the query to validate the syntax and ensure that there are no inconsistencies among parts of the statement. If any syntax errors are detected, the query compiler stops processing and returns the appropriate error message. Throughout this step, a data structure is built, namely an *AST* (Abstract Syntax Tree). Each node of the tree denotes a construct occurring in the source query, and is used through the rest of the compilation process. Simple rewrites, like *constant folding*, and the introduction of let bindings are also done in this step.

The *pipelining* phase transforms FLWOR expressions into *pipelines*—the internal, data-flow-oriented representation of FLWORs, discussed later. Optimizations are done atop pipelines, and the compiler uses global semantics stored in the AST to transform the query into a more-easily-optimized form. For example, the compiler will move predicates if possible, altering the level at which they are applied and potentially improving query performance. This type of operation movement is called *predicate pushdown*, or *filter pushdown*, and we will apply them to our stores later on. More optimizations such as *join recognition*, and *unnesting* are present in Brackit and are discussed in [4]. In the *optimization* phase, optimizations are applied to the AST. The *distribution* phase is specific to distributed scenarios, and is where MapReduce translation takes place. More details about the distribution phase are presented in [17]. At the end of the compilation, the *translator* receives the final AST. It generates a tree of executable physical operators. This compilation process chain is illustrated in Figure 1.

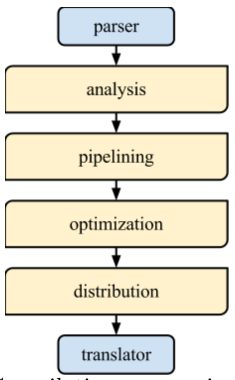


Figure 1: Compilation process in Brackit [5].

[1]Available at `http:\\www.brackit.org`

## XQuery over MapReduce

Mapping XQuery to the MapReduce model is an alternative to implementing a distributed query processor from scratch, as normally done in parallel databases. This choice relies on the MapReduce middleware for the distribution aspects. *BrackitMR* is one such implementation, and is more deeply discussed in [17]. It achieves a distributed XQuery engine in Brackit by scaling out using MapReduce.

The system hitherto cited processes collections stored in HDFS as text files, and therefore does not control details about encoding and management of low-level files. If the DBMS architecture [12] is considered, it implements solely the topmost layer of it, the set-oriented interface. It executes processes using MapReduce functions, but abstracts this from the final user by compiling XQuery over the MapReduce model.

It represents each query in MapReduce as sequence of *jobs*, where each job processes a section of a FLWOR pipeline. In order to use MapReduce as a query processor, (i) it breaks FLWOR pipelines are into map and reduce functions, and (ii) groups these functions to form a MapReduce job. On (i), it converts the logical-pipeline representation of the FLWOR expression—AST—to a MapReduce-friendly version. MapReduce uses a tree of *splits*, which represents the logical plan of a MapReduce-based query. Each *split* is a non-blocking operator used by MapReduce functions. The structure of splits is rather simple: it contains an AST and pointers to successor and predecessor splits. Because splits are organized in a bottom-up fashion, leaves of the tree are *map* functions, and the root is a *reduce* function—which produces the query output.

On (ii), the system uses the split tree to generate possibly multiple MapReduce job descriptions, which can be executed in a distributed manner. *Jobs* are exactly the ones used on Hadoop MapReduce [20], and therefore we will not go into details here.

## 4. XDM MAPPINGS

This section shows how to leverage NoSQL stores to work as storage layer for XQuery processing. First, we present mappings from NoSQL data models to XDM, adding XDM-node behavior to these data mappings. Afterwards, we discuss possible optimizations regarding data-filtering techniques.

## Riak

Riak's mapping strategy starts by constructing a *key/value tuple* from its low-level storage representation. This is essentially an abstraction and is completely dependent on the storage used by Riak. Second, we represent XDM operations on this key/value tuple. We map data stored within Riak utilizing Riak's linking mechanism. A key/value pair *kv* represents an XDM *element*, and key/value pairs linked to *kv* are addressed as children of *kv*. We map key/value tuples as XDM elements. The name of the element is simply the name of the bucket it belongs to. We create one bucket for the element itself, and one extra bucket for each link departing from the element. Each child element stored in a separated bucket represents a nested element within the key/value tuple. The name of the element is the name of the link between key/values. This does not necessarily decrease data locality: buckets are stored among distributed nodes based on hashed keys, therefore uniformly distributing the

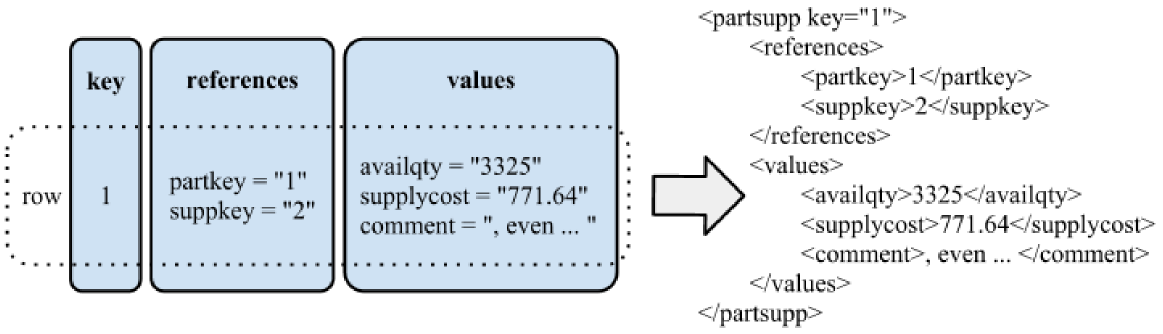


Figure 2: Mapping between an HBase row and an XDM instance.

load on the system. Besides, each element has an attribute *key* which Riak uses to access key/value pairs on the storage level.

It allows access using key/value as granularity, because every single element can be accessed within a single *get* operation. Full reconstruction of an element *el* requires one access for each key/value linked to *el*. Besides, Riak provides atomicity using single key/value pairs as granularity, therefore consistent updates of multiple key/value tuples cannot be guaranteed.

## HBase

HBase's mapping strategy starts by constructing a *columnar tuple* from the HDFS low-level-storage representation. HBase stores column-family data in separated files within HDFS, therefore we can use this to create an efficient mapping. Figure 2 presents this XDM mapping, where we map a table *partsupp* using two column families: *references* and *values*, five qualifiers: *partkey*, *suppkey*, *availqty*, *supplycost*, and *comment*. We map each row within an HBase table to an XDM element. The name of the element is simply the name of the table it belongs to, and we store the key used to access such element within HBase as an attribute in the element. The figure shows two *column families*: *references* and *values*. Each column family represents a child element, whose name is the name of the column family. Accordingly, each qualifier is nested as a child within the column-family element from which it descends.

## MongoDB

MongoDB's mapping strategy is straight-forward. Because it stores JSON-like documents, the mapping consists essentially of a *document field* → *element* mapping. We map each document within a MongoDB collection to an XDM element. The name of the element is the name of the collection it belongs to. We store the *id*—used to access the document within MongoDB—as an attribute on each element. Nested within the collection element, each field of the document represents a child element, whose name is the name of the field itself. Note that MongoDB allows fields to be of type *document*, therefore more complex nested elements can be achieved. Nevertheless, the mapping rules work recursively, just as described above.

## Nodes

We describe XDM mappings using object-oriented notation. Each store implements a *Node* interface that provides node behavior to data. Brackit interacts with the storage using this interface. It provides general rules present in XDM [19], Namespaces [2], and Xquery Update Facility [3] standards, resulting in navigational operations, comparisons, and other functionalities. *RiakRowNode* wraps Riak's *buckets*, *key/-values*, and *links*. *HBaseRowNode* wraps HBase's *tables*, *column families*, *qualifiers*, and *values*. Finally, *MongoRowNode* wraps MongoDB's *collections*, *documents*, *fields*, and *values*.

Overall, each instance of these objects represents one unit of data from the storage level. In order to better grasp the mapping, we describe the HBase abstraction in more details, because it represents the more complex case. Riak's and MongoDB's representation follow the same approach, but without a "second-level node". *Tables* are not represented within the Node interface, because their semantics represent where data is logically stored, and not data itself. Therefore, they are represented using a separated interface, called *Collection*. *Column families* represent a *first-level-access*. *Qualifiers* represent a *second-level-access*. Finally, *values* represent a *value-access*. Besides, first-level-access, second-level-access, and value-access must keep track of current indexes, allowing the node to properly implement XDM operations. Figure 3 depicts the mapping. The upper-most part of the picture shows a node which represents a data row from any of the three different stores. The first layer of nodes—with *level = 1st*—represents the *first-level-access*, explained previously. The semantic of first-level-access differs within different stores: while Riak and MongoDB interpret it as a value wrapper, HBase prefers a column family wrapper. Following, HBase is the only implementation that needs a *second-level-access*, represented by the middle-most node with *level = 2nd*, in this example accessing the wrapper of *regionkey = "1"*. Finally, lower-level nodes with *level = value* access values from the structure.

## Optimizations

We introduce projection and predicate pushdowns optimizations. The only storage that allows for predicate pushdown is MongoDB, while filter pushdown is realized on all of them. These optimizations are fundamental advantages of this work, when compared with processing MapReduce over raw files: we can take "shortcuts" that takes us directly to the bytes we want in the disk.

*Filter and projections pushdown* are an important optimization for minimizing the amount of data scanned and processed by storage levels, as well as reducing the amount

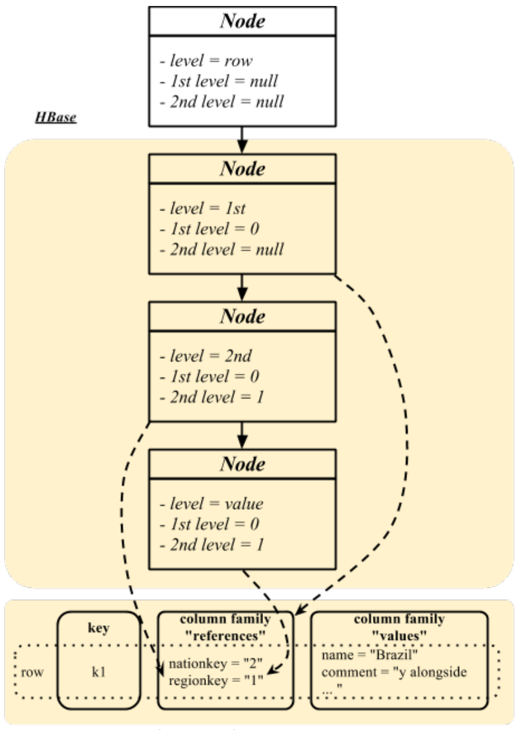


Figure 3: Nodes implementing XDM structure.

of data passed up to the query processor. *Predicate push-down* is yet another optimization technique to minimize the amount of data flowing between storage and processing layers. The whole idea is to process predicates as early in the plan as possible, thus pushing them to the storage layer.

On both cases we traverse the AST, generated in the beginning of the compilation step, looking for specific nodes, and when found we annotate the collection node on the AST with this information. The former looks for path expressions (*PathExpr*) that represent a *child* step from a collection node, or for *descendants* of collection nodes, because in the HBase implementation we have more than one access level within storage. The later looks for general-comparison operators, such as *equal*, *not equal*, *less than*, *greater than*, *less than or equal to*, and *greater than or equal to*. Afterwards, when accessing the collection on the storage level, we use the marked collections nodes to filter data, without further sending it to the query engine.

## NoSQL updates

The used NoSQL stores present different API to persist data. Even though XQuery does not provide data-storing mechanisms on its recommendation, it does provide an extension called *XQuery Update Facility* [3] for that end. It allows to add new nodes, delete or rename existing nodes, and replace existing nodes and their values. XQuery Update Facility adds very natural and efficient persistence-capabilities to XQuery, but it adds lots of complexity as well. Moreover, some of the constructions need document-order, which is simply not possible in the case of Riak. Therefore, simple-semantic functions such as "insert" or "put" seem more attractive, and achieve the goal of persisting or updating data.

The *insert* function stores a value within the underlying store. We provide two possible signatures: with or without a *$key*, therefore allowing for both insertions and updates.

```
db:insert($table as xs:string,
          $key as xs:string,
          $value as node()) as xs:boolean
```

The *delete* function deletes a values from the store. We also provide two possible signatures: with or without *$key*, therefore allowing for deletion of a giveng key, or droping a given table.

```
db:delete($table as xs:string,
          $key as xs:string) as xs:boolean
```

## 5. EXPERIMENTS

The framework we developed in this work is mainly concerned with the feasibility of executing XQuery queries atop NoSQL stores. Therefore, our focus is primarily on the proof of concept. The data used for our tests comes from the *TPC-H* benchmark [1]. The dataset size we used has 1GB, and we essentially scanned the five biggest tables on TPC-H: *part*, *partsupp*, *order*, *lineitem*, and *customer*. The experiments were performed in a single Intel Centrino Duo dual-core CPU with 2.00 GHz, with 4GB RAM, running Ubuntu Linux 10.04 LTS. HBase used is version 0.94.1, Riak is 1.2.1, and MongoDB is 2.2.1. It is not our goal to assess the scalability of these systems, but rather their query-procedure performance. For scalability benchmarks, we refer to [9] and [10].

### 5.1 Results

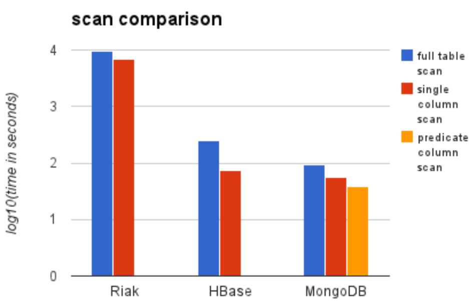


Figure 4: Latency comparison among stores.

Figure 4 shows the gathered latency times of the best schemes of each store, using log-scale. As we can see, all approaches take advantage from the optimization techniques. The blue column of the graph—*full table scan*—shows the latency when scanning all data from TPC-H tables. The red column —*single column scan*—represents the latency when scanning a simple column of each table. Filter pushdown optimizations explain the improvement in performance when compared to the first scan, reducing the amount of data flowing from storage to processing level. The orange column—*predicate column scan*—represents the latency when scanning a single column and where results were filtered by a predicate. We have chosen predicates to cut in half the amount of resulting data when compared with *single column scan*. The querying time was reduced in approximately 30%, not reaching the 50% theoretically-possible-improvement rate, essentially because of processing overhead. Nevertheless, it shows how efficient the technique is.

In scanning scenarios like the ones on this work, MongoDB has shown to be more efficient than the other stores, by always presenting better latency. MongoDB was faster by design: trading of data-storage capacity for data-addressability has proved to be a very efficiency-driven solution, although being a huge limitation. Moreover, MongoDB uses pre-caching techniques. Therefore, at run-time it allows working with data almost solely from main memory, specially in scanning scenarios.

## 6. CONCLUSIONS

We extended a mechanism that executes XQuery to work with different NoSQL stores as storage layer, thus providing a high-level interface to process data in an optimized manner. We have shown that our approach is generic enough to work with different NoSQL implementations.

Whenever querying these systems with MapReduce—taking advantage of its linearly-scalable programming model for processing and generating large-data sets—parallelization details, fault-tolerance, and distribution aspects are hidden from the user. Nevertheless, as a data-processing paradigm, MapReduce represents the past. It is not novel, does not use schemas, and provides a low-level record-at-a-time API: a scenario that represents the 1960's, before modern DBMS's. It requires implementing queries from scratch and still suffers from the lack of proper tools to enhance its querying capabilities. Moreover, when executed atop raw files, the processing is inefficient—because brute force is the only processing option. We solved precisely these two MapReduce problems: XQuery works as the higher-level query language, and NoSQL stores replace raw files, thus increasing performance. Overall, MapReduce emerges as solution for situations where DBMS's are too "hard" to work with, but it should not overlook the lessons of more than 40 years of database technology.

Other approaches cope with similar problems, like *Hive*, and *Scope*. Hive [18] is a framework for data warehousing on top of Hadoop. Nevertheless, it only provides equi-joins, and does not fully support point access, or CRUD operations—inserts into existing tables are not supported due to simplicity in the locking protocols. Moreover, it uses raw files as storage level, supporting only CSV files. Moreover, Hive is not flexible enough for Big Data problems, because it is not able to understand the structure of Hadoop files without some catalog information. Scope [7] provides a declarative scripting language targeted for massive data analysis, borrowing several features from SQL. It also runs atop a distributed computing platform, a MapReduce-like model, therefore suffering from the same problems: lack of flexibility and generality, although being scalable.

## 7. REFERENCES

[1] The tpc-h benchmark. `http://www.tpc.org/tpch/`, 1999.

[2] Namespaces in xml 1.1 (second edition). `http://www.w3.org/TR/xml-names11/`, August 2006.

[3] Xquery update facility 1.0. `http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/`, June 2009.

[4] S. Bächle. *Separating Key Concerns in Query Processing - Set Orientation, Physical Data Independence, and Parallelism.* PhD thesis, University of Kaiserslautern, 12 2012.

[5] S. Bächle and C. Sauer. Unleashing xquery for data-independent programming. *Submitted*, 2011.

[6] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C.-C. Kanne, F. Özcan, and E. J. Shekita. Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283, 2011.

[7] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1(2):1265–1276, Aug. 2008.

[8] K. Chodorow and M. Dirolf. *MongoDB: The Definitive Guide.* Oreilly Series. O'Reilly Media, Incorporated, 2010.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[10] T. Dory, B. Mejhas, P. V. Roy, and N. L. Tran. Measuring elasticity for cloud databases. In *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011.

[11] L. George. *HBase: The Definitive Guide.* O'Reilly Media, 2011.

[12] T. Härder. Dbms architecture - new challenges ahead. *Datenbank-Spektrum*, 14:38–48, 2005.

[13] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. Oltp through the looking glass, and what we found there, 2008.

[14] R. Klophaus. Riak core: building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming*, CUFP '10, pages 14:1–14:1, New York, NY, USA, 2010. ACM.

[15] F. Mattern. Virtual time and global states of distributed systems. In C. M. et al., editor, *Proc. Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland / Elsevier, 1989.

[16] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[17] C. Sauer. Xquery processing in the mapreduce framework. Master thesis, Technische Universität Kaiserslautern, 2012.

[18] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

[19] N. Walsh, M. Fernández, A. Malhotra, M. Nagy, and J. Marsh. XQuery 1.0 and XPath 2.0 data model (XDM). `http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/`, January 2007.

[20] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, 2012.