Proceedings

24. GI-Workshop „Grundlagen von Datenbanken“

29.05.2012 – 01.06.2012

Lübbenau, Deutschland

Ingo Schmitt, Sascha Saretz, Marcel Zierenberg (Hrsg.)

{schmitt | sascha.saretz | zieremar}@tu-cottbus.de

Brandenburgische
Technische Universität
Cottbus

Vorwort

Liebe Teilnehmerinnen und Teilnehmer,

der 24. Workshop „Grundlagen von Datenbanken“ (GvD) 2012 fand in Lübbenau, Spreewald, statt. Dieser viertägige Workshop wird alljährlich vom GI-Arbeitskreis Grundlagen von Informationssystemen im Fachbereich Datenbanken und Informationssysteme (DBIS) veranstaltet und hat die theoretischen, konzeptionellen und methodischen Grundlagen von Datenbanken und Informationssystemen zum Thema. Organisiert wurde der Workshop 2012 von der Forschungsgruppe Datenbank- und Informationssysteme am Institut für Informatik, Informations- und Medientechnik an der Brandenburgischen Technischen Universität Cottbus.

Der Workshop soll die Kommunikation zwischen Wissenschaftlern/-innen im deutschsprachigen Raum fördern, die sich grundlagenorientiert mit Datenbanken und Informationssystemen beschäftigen. Er ist insbesondere als Forum für Nachwuchswissenschafter/-innen gedacht, die ihre aktuellen Arbeiten in einem größeren Forum vorstellen wollen. Das Vattenfall-Tagungshotel Lübbenau bot für eine erfolgreiche Diskussion optimale Rahmenbedingungen in der landschaftlich sehr reizvollen Umgebung des Spreewaldes.

Insgesamt wurden für den Workshop 15 Arbeiten eingereicht und jeweils von drei Gutachtern bewertet. Die Einführung des Begutachtungsprozesses in den letzten Jahren sorgt schon im Vorfeld des Workshops für eine hohe Qualität der Arbeiten. Die Einreichungen spannten den Bogen von klassischen Themen wie Indexverwaltung über weitere Themen wie mobile Systeme, räumliche Datenbanken, Data Warehousing und XML-Verwaltung bis hin zu aktuellen Themen wie Self-Tuning, SaaS, Cloud Computing sowie probabilistische Datenbanken. Diese große Bandbreite der Gebiete reflektiert die Bandbreite von Problemen in der Datenbankgemeinde, wie sie auf großen Datenbankkonferenzen diskutiert werden. Dabei werden die Grenzen zwischen klassischen Datenbankthemen und benachbarten Bereichen immer durchlässiger. Gerade diese thematische Bandbreite und die damit einher gehende Spezialisierung auf einzelne Themen macht es für Doktoranden schwierig, die eigene Arbeit richtig einzuordnen und interessante Querverweise zu benachbarten Bereichen zu sehen. Die zwanglose Diskussion in einer offenen Atmosphäre des Workshop soll helfen, diese Lücke zu schließen.

Für die Keynote-Vorträge wurden Günther Specht, der Ausrichter des GvD-Workshops vom letzten Jahr und Bernhard Thalheim eingeladen. Ihnen sei an dieser Stelle für ihre Bereitschaft und ihr Kommen gedankt.

Des Weiteren danken wir dem Programmkomitee und allen Gutachtern für ihre Arbeit. Das Organisations-Komitee und dabei besonders Sandy Schneider, Marcel Zierenberg und Sascha Saretz haben den Großteil der Arbeit geleistet. Ohne ihren Einsatz und Engagement wäre der 24. Workshop nicht möglich gewesen. Herzlichen Dank. Besonderen Dank auch an Eike Schallehn und Hagen Höpfner sowie dem GI-Arbeitskreis „Grundlagen von Informationssystemen“, welche die Organisation sehr konstruktiv begleitet haben. Zum Schluss möchte ich allen Mitgliedern meines Lehrstuhl sowie den Autoren und Vortragenden für ihren Anteil am Gelingen am Workshop herzlich danken.

Mit den besten Grüßen,

Ingo Schmitt

Cottbus am 01.06.2012

Komitee

Programm-Komitee

- Wolf-Tilo Balke, TU Braunschweig
- Stefan Brass, Universität Halle
- Stefan Conrad, Universität Düsseldorf
- Erik Buchmann, Karlsruher Institut für Technologie
- Torsten Grust, Universität Tübingen
- Andreas Henrich, Universität Bamberg
- Hagen Höpfner, Bauhaus-Universität Weimar
- Harald Kosch, Universität Passau
- Klaus Meyer-Wegener, Universität Erlangen
- Daniela Nicklas, Universität Oldenburg
- Gunter Saake, Universität Magdeburg
- Eike Schallehn, Universität Magdeburg
- Ingo Schmitt, BTU Cottbus
- Holger Schwarz, Universität Stuttgart
- Günther Specht, Universität Innsbruck

Organisations-Komitee

- Ingo Schmitt, BTU Cottbus
- Sascha Saretz, BTU Cottbus
- Marcel Zierenberg, BTU Cottbus
- Sandy Schneider, BTU Cottbus

Weitere Gutachter

- Alexander Schulze, BTU Cottbus
- Sascha Saretz, BTU Cottbus
- Marcel Zierenberg, BTU Cottbus
- Daniel Blank, Universität Bamberg
- Martin Schäler, Universität Magdeburg
- Maik Mory, Universität Magdeburg
- Norbert Siegmund, Universität Magdeburg

Inhaltsverzeichnis

Out of Africa?

Günther Specht
Datenbanken und Informationssysteme
Institut für Informatik
Universität Innsbruck, Österreich
guenther.specht@uibk.ac.at

KURZFASSUNG

Woher? und Wohin? sind die bewegenden Fragen des Lebens, der Forschung und der Wissenschaft. Auch der Datenbanktechnologie. Dieser Vortrag gibt sich nicht mit der Antwort 42 zufrieden, sondern spannt einen Bogen bis hin zur interdisziplinären Vernetzung der Datenbanken und Informationssysteme mit der Genetik. Dort gelang es heuer in der Haplogruppeneinteilung der Frage „Out of Africa?“ mittels massivem IT- und DB-Einsatz eine neue Antwort zu geben...

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Privacy-Enhanced Information Systems

Bernhard Thalheim
Lehrstuhl für Technologie der Informationssysteme
Institut für Informatik
Christian-Albrechts-Universität zu Kiel
thalheim@is.informatik.uni-kiel.de

KURZFASSUNG

Nach einem kurzen Überblick über den State-of-the-Art der Nutzung des Internets werden Prinzipien, Trends und Technologie für die Wahrung der Privatsphäre dargestellt. Daraus lassen sich vielfältige Modellierungs-, Infrastruktur-, Technologie-, Sozial- und Rechtsprojekte zur Unterstützung von Privatsphäre im Internet ableiten. Ziel ist die Entwicklung einer erweiterten Infrastruktur, in der Privatheit auch im Internet möglich ist. Im Detail stellen wir Ansätze vor, mit denen die Privatheit an Daten gesichert werden kann. Dazu ist ein Sicherheitsmodell, ein Inhaltemodell und ein Sicherungsmodell notwendig. Darauf kann ein Koordinationsmodell aufgesetzt werden. Techniken zur Stützung von Privatheit können darauf aufgesetzt werden. Dies wird anhand von zwei Datenbank-Ansätzen gezeigt.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Towards Modeling Locations as Poly-Hierarchies

Hagen Höpfner and Maximilian Schirmer
Bauhaus-Universität Weimar
Media Department / Mobile Media Group
Bauhausstraße 11, 99423 Weimar, Germany
hoepfner@acm.org, maximilian.schirmer@uni-weimar.de

Keywords

Location, Location Modeling, Poly-Hierarchies, Enclave Problem, Multiple Belonging

ABSTRACT

Location models are formal descriptions of locations (i. e., named sets of geographical positions) as well as of semantic relationships among locations. There exist various location models that vary in the considered location information, their level of detail, the kind of modelled relations and the used formal representation. In fact, more complex location models cover more aspects required for implementing location-aware or location-dependent systems, but also require more complex algorithms. As each application domain requires only a limited set of features, limiting a model to those features helps to improve system performance. In this paper, we discuss a novel location model called location poly-hierarchies that models the belonging of one location to one ore more other locations. Furthermore, we present an approach for creating location poly-hierarchies from a given set of locations.

1. INTRODUCTION AND MOTIVATION

In February 2004, the authors of [5] stated: "The widespread deployment of sensing technologies will make location-aware applications part of every day life." Nowadays, location-awareness has become a key feature in a broad range of mobile information systems. Navigation systems, social network apps, tourist information systems, event information systems, shop finders and many more heavily rely on position data of their users. Consequently, almost all modern smartphones supply positioning techniques. However, a position determination, e.g, by the use of the Global Positioning System (GPS), enables a smartphone to calculate coordinates and accuracy, only. As, from the perspective of the user, a semantic location is more understandable than coordinate information, location-aware or location-based systems map position information to semantically meaningful locations. For example, the GPS coordinates 50.972 797 and 11.329 18 are precise but likely not meaningful for humans. They belong to the building which is placed in the Bauhausstraße 11 in Weimar, Germany and the assumption being that such information is usually more meaningful to the user. Hence, we should use a service that maps positions to locations. However, location-based applications differ in their required precision [10]. While the name of the city in which a user and/or a device is currently located might be appropriate for handling location-aware data processing in an event information system, a navigation system demands for exact coordinates, or street names at least.

The aforementioned example also illustrates another issue that is common for locations. In many cases, semantic locations form hierarchical structures. For example, in a hierarchical location model, earth is divided into continents, continents into countries, countries into states, states into cities, cities into streets and streets into street numbers, and so on. However, modelling locations as mono-hierarchies oversimplifies the reality and does not support multiple belongings. While, e. g., Weimar is part of Germany, Germany of Europe, etc., Istanbul is part of Turkey, but also of Europe and Asia. Please note, we focus on geographic belongs-to-relationships (subset and overlap) rather than on geopolitical ones. Location hierarchies benefit from the fact that determining low-level location information determines upper levels, too. Location poly-hierarchies cure the mentioned model incompleteness while keeping the benefit of a "fast" lookup of the correct path within the poly-hierarchy (cf., [11]). The research question addressed in this paper is:

> *How can one algorithmically create location poly-hierarchies from a given set of locations?*

The remainder of this paper is structured as follows: Section 2 surveys related work. Section 3 introduces the concept of location poly-hierarchies. Section 4 presents our approach for creating them. Section 5 discusses implementation aspects. Section 6 summarises the paper and gives an outlook on future research.

2. RELATED WORK

There has been a lot of active research on suitable models and representations for location data in the field of location models. Location models are a core component of location-based applications. They represent not only location information, but also spatial (or even spatio-temporal [7]) relationships in the data, help to express relative locations, proximity, and allow users to determine containment of locations or connectedness of relationships.

The authors of [13] present in great detail the broad variety of location models that has been developed in recent years of active research. A key factor for distinguishing and characterising location models is their way of representing spatial relationships. According to [1], they can be categorised into set-based, hierarchical, and graph-based models. Hybrid models that combine several aspects exist as well. Figure 1 presents an overview of the three main lo-

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

cation model concepts. In the illustrated examples, the set-based approach is the least expressive one, as it only models the fact that there are two distinct locations within a set of locations, and a set of coordinates is assigned to each location. The hierarchical model adds containment information, and the graph-based model adds connectedness and distance in the form of edge weights.

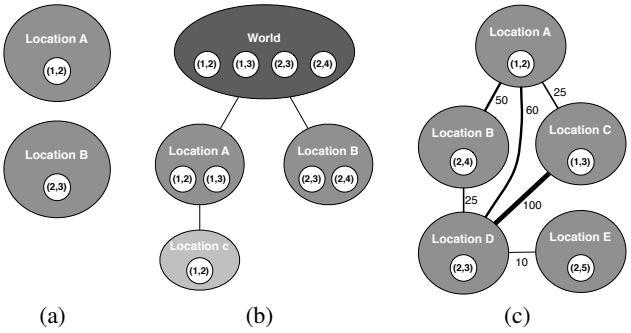


Figure 1: Examples for set-based (a), hierarchical (b), and graph-based (c) location models. The edge weights in the graph-based model represent distance information.

Hierarchical location models as a special case of set-based models represent containment relationships between different levels of the model and are widely used as basis for location-based applications [6, 2, 4]. Hierarchical models cannot represent distance information or directly encode proximity, but they have great advantages in traversal and for containment queries. They are also very close to the common human understanding of locations. As already stated in Section 1, the widely acknowledged segmentation of locations into administrative regions (country, state, city, street, and so on) is also a hierarchical model that heavily relies on containment information. On a city level, a lot of implicit information can be derived through the top-level relationship to a state or country (e.g., administrative language, local cuisine, prevalent religions).

3. LOCATION POLY-HIERARCHIES

The authors of [12] define the term "location" as follows: "Location of an object or a person is its geographical position on the earth with respect to a reference point." From our point of view, this definition is too restrictive, because geographic positions are points within a reference system. In contrast to a point, a location has a spatial extent. Another definition is given in [3]: "Geographic location is the text description of an area in a special confine on the earth's surface.". From a more theoretical point of view, an *area* is a set of geographical positions. So, we use a set-oriented definition: *A location is a named set of geographical positions on earth with respect to a reference point.*

For example, in a two-dimensional coordinate system[1], the location of a building, lets say a train station (ts), is given as set $L_{ts} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where each position (point) $(x_i, y_i), 1 \leq i \leq n$ belongs to the train station building's area. We discuss the calculation of the point set in Section 5.1. Consequently, we can characterise the location of an object or a person as a relationship between sets. A person, lets say Tanja, is located at the train station if $L_{Tanja} \subset L_{ts}$ holds. Recent positioning technologies like GPS return only single points. However, we can interpret Tanja's current GPS coordinate as a set of positions of cardinality 1. As discussed in Section 1 it is a common understanding that locations have a hierarchic nature. The train station is located within a city, the city is located in a state, the state within a country, and so on. Using this subset-based location interpretation, $L_{Tanja} \subset L_{ts} \subset L_{Weimar} \subset L_{Thuringia} \subset L_{Germany} \subset L_{Europe} \subset L_{Earth}$ represents the location of Tanja as path in a location mono-hierarchy.

However, there exist various real-world issues that require a poly-hierarchical representation of locations. For example, Istanbul as the capital of Turkey, belongs to the continents Europe and Asia. Hence, $L_{Istanbul} \not\subset L_{Europe}$ and $L_{Istanbul} \not\subset L_{Asia}$ hold. The same issue holds for Russia, which is located in Europe and in Asia, too. Another problem results from enclaves. Kaliningrad, e.g., is part of Russia, but this information is insufficient to decide whether Kaliningrad is part of Europe or part of Asia. The solution for these problems is the use of set overlaps in combination with containment relationships that form a location poly-hierarchy.

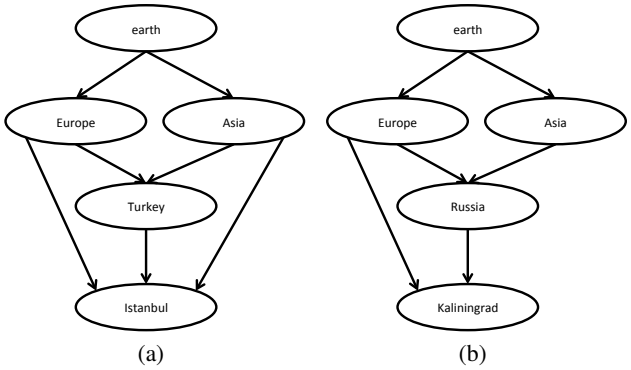


Figure 2: Examples for the Poly-hierarchical nature of locations: Istanbul (a), and Kaliningrad (b).

Figure 2 illustrates the simplified poly-hierarchies for Istanbul and Kaliningrad. From the definition of poly-hierarchies, we know that they can be represented by a directed acyclic graph $LPH = (V, E)$. Each node $v \in V$ is a location and each directed edge $e = (v_1, v_2)$ with $v_1, v_2 \in V$ represents that the child node v_2 belongs (semantically) to the parent node v_1. Each location poly-hierarchy has an unique root node $v_r \in V$ with $\neg\exists v_x \in V | (v_x, v_r)$, because the entire coordinate system is closed in case of locations (all considerable positions are elements of the set of all positions on earth). Finally, for each leaf node $v_l \in V$ that must not have any child node $\neg\exists v_x \in V | (v_l, v_x)$ holds.

From an implementation point of view, each node in the poly-hierarchy graph has the structure (n, P, C) where n is the name of the location L_n, P is a set of edges to the parent nodes and C is a set of edges to child nodes. The root node $r = (\text{earth}, P, C)$ is the only node in LPH for which $P = \emptyset \wedge C \neq \emptyset$ must hold. For leaf nodes $P \neq \emptyset \wedge C = \emptyset$, and for inner nodes $P \neq \emptyset \wedge C \neq \emptyset$ hold. For the concept described in the remainder of this paper, it is required to know the level of each node within this graph. The level $level(m)$ of a node m is defined as the number of nodes on the longest direct path from r to m, plus one. As illustrated in Figure 2(b), $level(Russia) = 2$ and $level(Kaliningrad) = 3$.

In a location mono-hierarchy each edge between a parent node and a child node represents the fact that all points of the location that corresponds to the child node are also elements of the location that corresponds to the parent node. However, as discussed before,

[1]For simplification purposes, we use two-dimensional coordinates in this paper. However, the formalism can easily be adapted to three dimensions.

it is not sufficient to use subset relationships only. The semantics of edges in the poly-hierarchical graph representation is as follows. Given a (child) node $c = (n_c, P_c, C_c)$ the following relations hold:

- For each parent node $p \in V$ referenced in P_c having *level*(p) with $\neg\exists p' \in P_c | p' \neq p \wedge$ *level*$(p) =$ *level*(p') the edge $(p, c) \in E$ represents the subset relationship $L_{n_c} \subset L_{n_p}$.
- For each (parent) node $p \in V$ referenced in P_c having *level*(p) with $\exists p' \in P_c | p' \neq p \wedge$ *level*$(p) =$ *level*(p') the edge $(p, c) \in E$ represents the overlapping relationship $L_{n_c} \cap L_{n_p} \neq \emptyset$.

In other words this means that per hierarchy level each edge between a single parent node and the child node means an subset relationship. If a child node has more than only one parent node in the same level, then those links represent an overlap relationship. Furthermore, we know that the in the latter case, due to the directed nature of the edges, the child node must be a subset of the union of the respective parent nodes (i. e., the conjunction of the overlap relationships per level holds).

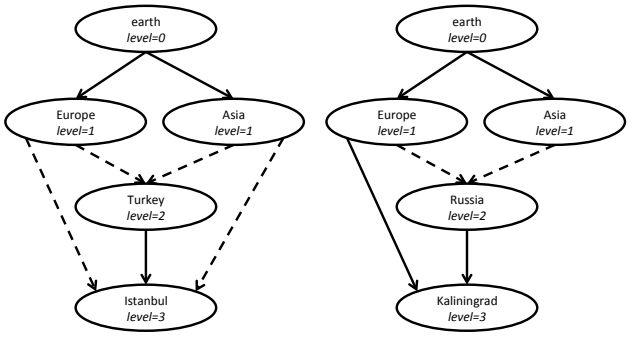


Figure 3: Examples for the semantics of edges in LPH, solid lines represent subset relationships and dashed lines overlaps.

Figure 3 illustrates the link semantics for the Istanbul and the Kaliningrad examples. As one can see in Subfigure 3(a), $L_{Istanbul} \subset L_{Turkey}$ as Turkey is the only parent node of Istanbul at level two. Since Istanbul has two parent nodes on level one (i. e., Europe and Asia), these links represent the overlaps $L_{Istanbul} \cap L_{Europe} \neq \emptyset$ and $L_{Istanbul} \cap L_{Asia} \neq \emptyset$. The facts that (in addition) $L_{Turkey} \cap L_{Europe} \neq \emptyset \wedge L_{Turkey} \cap L_{Asia} \neq \emptyset$ holds, also implies $L_{Istanbul} \subset L_{Europe} \cap L_{Asia}$. We could use this "transitive" conclusion in a similar way for the Kaliningrad example, too. However, as show in Subfigure 3(b) it would reduce the expressivity of this LPH. Kaliningrad is only part of Europe but not of Asia. Hence, the Europe node is the only parent node of the Kaliningrad node at level one.

4. LPH CREATION

As discussed in Section 3 one could create an LPH using overlap relationships only. We already pointed out that, in order to be as expressive as possible, an LPH must use as many subset relationships as possible. Algorithm 1 creates the LPH from a given set $\mathcal{L} = \{L_1, \ldots, L_k\}$ of locations. For illustration purposes, we assume that names of locations are unique. However, one could also use location IDs instead of the names to guarantee uniqueness.

Our algorithm uses four main steps. In the first step (lines 7-24), it creates two subgraphs, one $(V^{\subset}, E^{\subset})$ containing all subset re-

Algorithm 1 Creating a location poly-hierarchy from a location set

Input: $\mathcal{L} = \{L_1, \ldots, L_k\}$ // location set

Output: $LPH = (V, E)$ // location poly-hierarchy

```
01  def naive_lph_create(L):
02    V^⊂ = ∅ // init of the node set for subset relations
03    V^∩ = ∅ // init of the node set for overlap relations
04    E^⊂ = ∅ // init the edge set for subset relations
05    E^∩ = ∅ // init the edge set for overlap relations
06    L^V = ∅ // init set of already analysed locations
— STEP 1: FINDING OVERLAP AND SUBSET RELATIONSHIPS —
07    for each L_name1 ∈ L do
08      for each L_name2 ∈ L − {L_name1} − L^V do
09        if L_name2 ⊂ L_name1 then
10          V^⊂ = V^⊂ ∪ {name1, name2}
11          E^⊂ = E^⊂ ∪ {(name1, name2)}
12        fi
13        elif L_name1 ⊂ L_name2 then
14          V^⊂ = V^⊂ ∪ {name1, name2}
15          E^⊂ = E^⊂ ∪ {(name2, name1)}
16        fi
17        elif L_name2 ∩ L_name1 ≠ ∅ then
18          V^∩ = V^∩ ∪ {name1, name2}
19          E^∩ = E^∩ ∪ {(name2, name1)}
20          E^∩ = E^∩ ∪ {(name1, name2)}
21        fi
22      done
23      L^V = L^V ∪ {L_name1}
24    done
25    if E^⊂ == ∅ then return(FALSE) fi // no root node found
— STEP 2: CLEANING UP E^∩ (REMOVING LOOPS) —
26    for each name1 ∈ V^∩ do
27      L^t = ∅; E^t = ∅ // temporary location and edge sets
28      for each e ∈ V^∩ do
29        if e == (name1, x) then
30          L^t = L^t ∪ {L_x}; E^t = E^t ∪ {(name1, x)}
31        fi
32      done
33      if L_name1 ⊂ L^t then E^∩ = E^∩ − E^t fi
34    done
— STEP 3: CLEANING UP E^⊂ (TRANSITIVITY) —
35    for each name1 ∈ V^⊂ do
36      if ∃x, y ∈ V^⊂ ↔ {(x, name1), (y, x)} ∩ E^⊂ ≠ ∅ then
37        V^t = {z | z ≠ x ∧ (z, name1) ∈ V^⊂}
38        E^⊂ = E^⊂ − ⋃_{z∈V^t} {(z, name1)}
39      fi
40      V^t = {x | (x, name1) ∈ E^⊂} // all parents of name1
41      for each x ∈ V^t do
42        V^c = {y | (x, y) ∈ E^⊂ ∧ y ∈ V^t − {name1}}
43        if L_name1 ⊂ ⋃_{c∈V^c} L_c then
44          E^⊂ = E^⊂ − {(x, name1)}
45        fi
46      done
47    done
— STEP 4: MERGING E^∩ WITH E^⊂ AND V^∩ WITH V^⊂ —
48    V = V^∩ ∪ V^⊂; E = E^∩ ∪ E^⊂
49    return(V, E)
```

lationships and one ($V^{\cap}, E^{\cap}$) for all (additional) overlap relationships. Figure 4 illustrates these subgraphs for the Istanbul and the Kaliningrad examples. Obviously, step 1 might generate redundant edges (cf., Figure 4(c)) `earth`→`Russia`→`Kaliningrad` and `earth`→`Kaliningrad`) or (later on) unnecessary edges (cf., Figure 4(a) `earth`→`Turkey`). Furthermore, it generates loops in the overlap subgraph as overlap relations have a symmetric nature (cf., Figure 4(b) and Figure 4(d)). We illustrated those needless edges using dotted arrows.

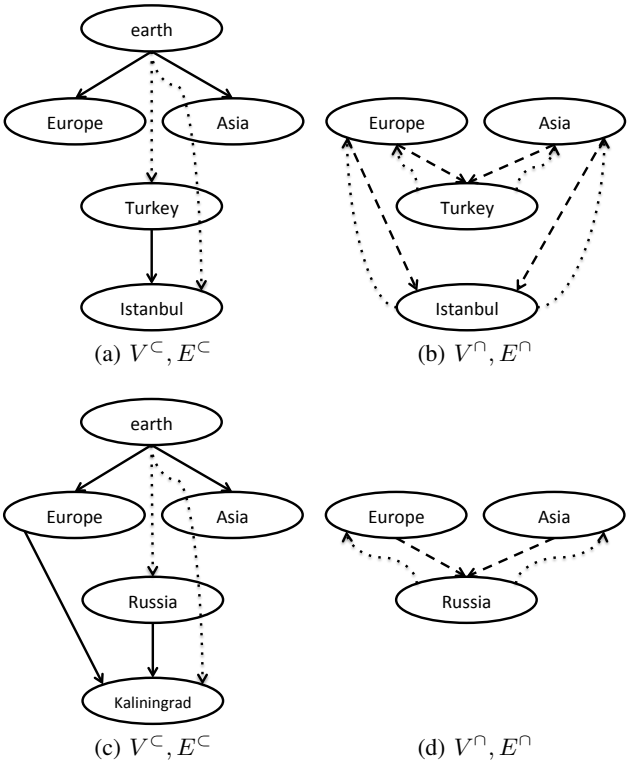


Figure 4: Intermediate graph(s) after step 1; (a,b) Istanbul, (c,d) Kaliningrad.

Step 2 (lines 26-34) of the algorithm cleans up the overlap subgraph, i. e., it breaks the loops. Therefore, each node of this subgraph is analysed (cf. Figure 5). If the location represented by a node is a subset of the union of all its direct child nodes, then we remove the outgoing edges.

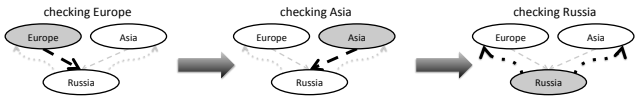


Figure 5: Step 2 for the Kaliningrad example; bold arrows highlight the edges analysed for currently handled node that is marked gray.

Figure 6 illustrates step 2 for the Istanbul example. After checking the nodes for Europe and Asia, the node for Turkey is checked. So far, it has the child nodes Europe and Asia. As all points of Turkey belong to Europe or Asia, both must not be represented by child nodes of the Turkey node. So, these edges are removed. The same procedure removes the edges `Istanbul`→`Europe` and `Istanbul`→`Asia`.

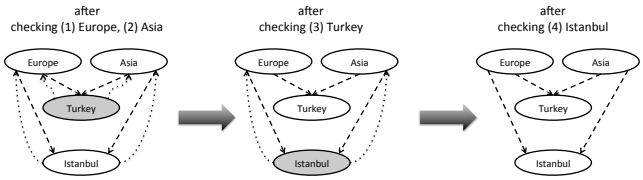


Figure 6: Step 2 for the Istanbul example.

Step 3 (lines 35-47) cleans up the subset subgraph using the transitivity property of subset relationships. In contrast to many graph optimisation approaches, we do not aim for reducing the number of edges in general, but for finding the minimal number of edges necessary for representing correct semantics (cf. Section 3). For the first optimisation (lines 36-39), we first calculate for each node the set of parent nodes having parent nodes themselves. Afterwards, we remove all direct edges from grandparents nodes as they can be reached through the parents. In our examples, this removes the edges `earth`→`Istanbul` and `earth`→`Kaliningrad`. A second optimisation (lines 40-46) checks whether a node is contained in the union of its sibling locations. If this is the case, we can remove the edge from the parent node (in our examples `earth`→`Turkey` and `earth`→`Russia`), even if this fragments the subgraph (cf. Figure 7).

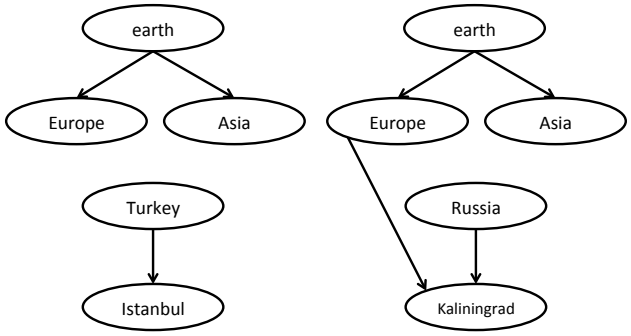


Figure 7: Intermediate graphs ($V^{\subset}, E^{\subset}$) after step 3.

Joining the subset subgraph(s) with the overlap subgraph, as it is done in the final step 4 (line 48), results in a connected LPH. The fragmentation, that might have happened in step 2, is cured as the removed edges resulted from an overlap relationship between the involved nodes. For our examples, Algorithm 1 therefore creates the location poly-hierarchies shown in Figure 3.

Limiting factors

Algorithm 1 requires a complete and closed set of locations. Without the location Asia, the Kaliningrad example graph would loose the LPH properties discussed in Section 3 as removing the Asia node also removes the edge `Asia`→`Russia`. Without this edge, the relationship `Europe`→`Russia` would be semantically wrongly interpreted as subset relationship. In fact, Algorithm 1 could be adapted in order to recognise this case through topologic sorting as the condition in line 33 would evaluate to false. Consequently,

$E^{\cap}$ would still contain loops after step 2. Furthermore, the algorithm does not support differently named locations with equal sets of position points (i. e., $L_a \subseteq L_b \wedge L_b \subseteq L_a$ holds). Step 1 would missinterpret this case as overlap relationship. Consequently, step 2 would nondeterministically remove one of the edges. A solution to this problem would be a cleanup phase that unifies those locations before starting Algorithm 1.

5. IMPLEMENTATION ASPECTS

As the "towards" in the title of this paper denotes, the presented results are subject to ongoing research. We were not able to finish the import of the evaluation database before the deadline. In fact, importing the OpenStreetMap database containing data about Europe (`http://download.geofabrik.de/osm`) using an slightly adapted version of the `osm2postgresql_05rc4.sh` script, which can be downloaded from `http://sourceforge.net/projects/osm2postgresql`, is still in progress. So far it took more than two weeks (PosgreSQL 9.1 [9] with PostGIS 1.5.1 [8] running on an iMac Intel Core 2 Duo 3.06 GHz, 4 GB 1067 MHz RAM, OS X 10.7.3). However, there are certain implementation issues that we find worthwhile to be discussed.

5.1 Point sets of locations

The formal definition of locations and location poly-hierarchies as discussed in Section 3 is based on set theory. From a theoretical point of view, those sets are infinite. A common approach to handle this infinity problem is to decrease the precision of set elements through quantisation. However, it would not be useful or even possible to physically store (materialise) all points of all locations. Hence, for the implementation of our approach we decided to use a polygon-based representation of locations, where a set of polygons describes the boundaries of the locations. In principal, all points that are inside of one of these polygons belong to the location. The OpenStreetMap data model provides an additional multipolygon relation (cf., `http://wiki.openstreetmap.org/wiki/Multipolygon_relation`) for representing more complex areas. It allows, e. g., for areas within a location that do not belong to this particular location. Hence, from an implementation point of view, a location is represented as a database view. The location name corresponds to the name of the view and the point set is defined by a database query resulting in the location outline (multi)polygon.

5.2 Set operations

Interpreting locations as (multi)polygons necessitates a different interpretation of the used set operations, too. As discussed in Section 4, Algorithm 1 has to check subset and set overlap relationships. Remember, a location L_a contains another location L_b if and only if $L_b \subset L_a$ holds. Hence, L_a must contain all points $(x_b, y_b) \in L_b$. For the polygon-based locations, the subset relation means that the (multi)polygon describing L_a must cover those of L_b completely. Similarly, an overlap relation of two locations implies that the boundary (multi)polygons overlap (and vice versa). Most geographical information system extensions, such as PostGIS, provide the corresponding operators (cf. [8]).

6. SUMMARY AND OUTLOOK

We presented a novel approach to model (geographical) relationships among locations. We extended the effective, but not complete location hierarchy approach in order to support enclaves and overlapping locations. We formally introduced the new location poly-hierarchy model and presented an algorithm for creating a location poly-hierarchy from a given (closed and complete) set of locations. We discussed limitations of the algorithm and also pointed out potential solutions to handle them. Finally, we discussed some issues that need to be considered for the implementation of our strategy. However, there are many open issues that lead to future research directions. First of all we plan to evaluate the algorithm with real world data. We expect that the algorithm works properly, but we are aware of the huge amount of data that needs to be processed. Hence, we will have to optimise the algorithm in order to avoid unnecessary (database) operations. Furthermore, we will research whether it would be better to explicitly keep the information about the type of the relationships. This extended graph model would, of course, ease the interpretability of the final location poly-hierarchy, but it would also increase the complexity of the model.

7. REFERENCES

[1] C. Becker and F. Dürr. On Location Models for Ubiquitous Computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[2] M. Beigl, T. Zimmer, and C. Decker. A Location Model for Communicating and Processing of Context. *Personal Ubiquitous Computing*, 6:341–357, Jan. 2002.

[3] Z. Dongqing, L. Zhiping, and Z. Xiguang. Location and its Semantics in Location-Based Services. *Geo-spatial Information Science*, 10(2):145–150, June 2007.

[4] F. Dürr and K. Rothermel. On a Location Model for Fine-Grained Geocast. In A. K. Dey, A. Schmidt, and J. F. McCarthy, editors, *UbiComp '03 Proceedings*, volume 2864 of *LNCS*, pages 18–35, Berlin / Heidelberg, 2003. Springer.

[5] M. Hazas, J. Scott, and J. Krumm. Location-Aware Computing Comes of Age. *IEEE Computer*, 37(2):95–97, Feb. 2004.

[6] C. Jiang and P. Steenkiste. A Hybrid Location Model with a Computable Location Identifier for Ubiquitous Computing. In G. Borriello and L. E. Holmquist, editors, *Ubicomp '02 Proceedings*, volume 2498 of *LNCS*, pages 246–263, London, UK, 2002. Springer.

[7] T. Kauppinen, R. Henriksson, R. Sinkkilä, R. Lindroos, J. Väätäinen, and E. Hyvönen. Ontology-based Disambiguation of Spatiotemporal Locations. In *IRSW '08 Proceedings*. CEUR-WS.org, 2008.

[8] OSGeo Project. *PostGIS 1.5.1 Manual*. `http://postgis.refractions.net/download/postgis-1.5.1.pdf`.

[9] The PostgreSQL Global Development Group. *PostgreSQL 9.1.3 Documentation*. `http://www.postgresql.org/docs/9.1/static/index.html`.

[10] B. N. Schilit, A. LaMarca, G. Borriello, W. G. Griswold, D. McDonald, E. Lazowska, A. Balachandran, J. Hong, and V. Iverson. Challenge: ubiquitous location-aware computing and the "place lab" initiative. In *WMASH '03 Proceedings*, pages 29–35, New York, NY, USA, 2003. ACM.

[11] M. Schirmer and H. Höpfner. Towards Using Location Poly-Hierarchies for Energy-Efficient Continuous Location Determination. In *GvD '12 Proceeding*. CEUR-WS.org, 2012. forthcoming.

[12] A. Y. Seydim, M. H. Dunham, and V. Kumar. Location dependent query processing. In *MobiDE '01 Proceedings*, pages 47–53, New York, NY, USA, 2001. ACM.

[13] J. Ye, L. Coyle, S. Dobson, and P. Nixon. A Unified Semantics Space Model. In *LoCA '07 Proceedings*, LNCS, pages 103–120, Berlin / Heidelberg, 2007. Springer.

Towards Using Location Poly-Hierarchies for Energy-Efficient Continuous Location Determination

Maximilian Schirmer and Hagen Höpfner
Bauhaus-Universität Weimar
Media Department / Mobile Media Group
Bauhausstraße 11, 99423 Weimar, Germany
maximilian.schirmer@uni-weimar.de, hoepfner@acm.org

Keywords

Location Determination, Location Poly-Hierarchies, Energy Efficiency

ABSTRACT

Location awareness is a key feature of mobile information systems. Typically, location is determined by interpreting a set of measured positions. Various approaches for position determination do exist. They vary greatly in their precision, applicability, and energy requirements. As mobile devices are battery-driven, energy is one of the most limiting factors for the system's uptime. Locations have a hierarchical nature and location-based applications differ in their required precision. In this paper, we present three approaches that utilise location poly-hierarchies in order to reduce the energy demand of continuous location determination: (1) We analyse the dependencies among the different hierarchy levels, (2) we incorporate an adaptive delay between measurements based on the hierarchy level and the calculated minimal required time to change, and (3) we select appropriate positioning techniques for each hierarchy level. We implemented and evaluated our approaches.

1. INTRODUCTION AND MOTIVATION

Navigation systems, social network apps, tourist information systems, event information systems, shop finders and many more heavily rely on position data of their users. Consequently, almost all modern smartphones supply positioning techniques. The most popular positioning technique is the Global Positioning System (GPS). However, even devices that do not provide GPS hardware are able to locate themselves using alternative techniques such as geotagged Wi-Fi hotspot and cell tower databases (db), wireless signal triangulation/lateration, or geotagging. Although all of them allow localisation of a mobile device, they vary dramatically in precision, applicability, hardware requirements, and energy demands. A GPS request requires a GPS receiver with a much higher energy demand compared to an on-device database lookup that is based on already localised cell towers or Wi-Fi hotspots [4]. However, in an outdoor scenario, GPS is far more precise than the analysis of location information of connected Wi-Fi hotspots or cell towers [22]. On the other hand, indoor GPS positioning is almost impossible because of the occlusion and shielding of satellite signals created by building structures.

Mobile devices are battery-driven. So, energy is one of the most limiting factors for their uptime. Additionally, the user acceptance of location-based or context-aware mobile applications is negatively influenced when the applications heavily strain the mobile devices' batteries. Furthermore, location-based applications differ in their required precision [16]. While the name of the city a user or a device is currently located in might be appropriate for an event information system, a navigation system might demand for exact coordinates or street names at least. A common representation of locations follows their hierarchical nature. In a hierarchical model, earth is divided into continents, continents into countries, countries into states, states into cities, cities into streets and streets into street numbers, and so on. Consequently, determining low-level location information in this hierarchy also determines upper levels and contains information that is connected to these levels. In addition to this, locations only change if the device is moving. While GPS coordinates might change with each movement, city information is stable until the device leaves the city. Hence, the system might wait with the next energy-demanding location determination on the city level until the device possibly left the city. In this paper, we present three approaches that utilise location poly-hierarchies for reducing the energy demand of continuous location determination:

1. We analyse dependencies among different hierarchy levels.
2. We postpone position measurements based on a calculated minimal time that is required for leaving the current location.
3. We select appropriate positioning techniques for each hierarchy level.

Our preliminary experimental results show that there is a strong potential in these techniques to reduce the energy demand compared to continuous GPS polling.

The remainder of the paper is structured as follows: Section 2 discusses related work. Section 3 introduces the concept of location hierarchies. Section 4 presents the three aforementioned approaches. Section 5 describes the evaluation approach and the results. Section 6 summarises the paper.

2. RELATED WORK

Our work mainly overlaps with the research fields *location models* and *energy-aware computing*. Location models form the basis for all high-level operations on location data. Consequently, the frequent and enduring use of location data in mobile computing immediately raises the issue of the mobile devices' limited energy

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

resources. The field of energy-aware computing presents a variety of concepts and methods to compensate for these constraints.

2.1 Location Models

Location models as core components of location-based applications represent location information and spatial (or even spatio-temporal [15]) relationships in data. They help to express relative locations, proximity, and allow users to determine containment of locations or connectedness of relationships.

The authors of [21] present in great detail the broad variety of location models that have been developed in recent years of active research. A key factor for distinguishing and characterising location models is their way of representing spatial relationships. According to [1], they can be categorised into set-based, hierarchical, and graph-based models. Hybrid models that combine several aspects exist as well. Figure 1 presents an overview of the three main concepts. In the illustrated examples, the set-based approach is the least expressive one, as it only models the fact that there are two distinct locations within a set of locations, and a set of coordinates is assigned to each location. The hierarchical model adds containment information, and the graph-based model adds connectedness as well as distance in the form of edge weights.

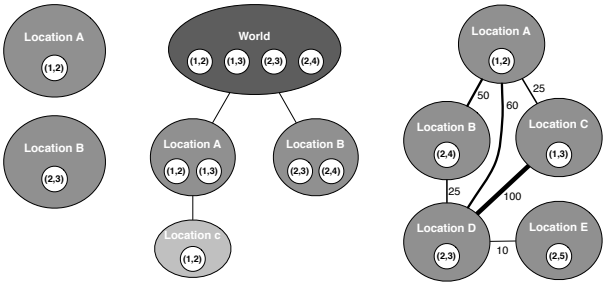


Figure 1: Examples for different popular location models: set-based (a), hierarchical (b), and graph-based (c). The edge weights in the graph-based model represent distance information.

Hierarchical location models as a special case of set-based models represent containment relationships between different levels of the model and are widely used as basis for location-based applications [14, 2, 6]. They cannot represent distance information or directly encode proximity, but they have great advantages in traversal and for containment queries. They are very close to the common human understanding of locations. Almost everyone understands the widely acknowledged segmentation of locations into administrative regions (country, state, city, street, etc.). At this, on a city level, a lot of implicit information can be derived through the top-level relationship to a state or country (e.g., administrative language, local cuisine, prevalent religions).

2.2 Energy-aware Computing

Energy-aware computing recognises the need for energy as a factor in modelling and implementing computing systems in order to manage and reduce their energy demand. This includes both hardware and software systems. While energy-aware hardware has been under active research for many years, energy-aware software is still a novel and underestimated field of research. Hardware solutions such as sleep modes or performance scaling cannot be directly transferred or adapted to software systems. Energy-aware software requires dedicated software engineering with new concepts and algorithms [9].

One concept of energy-aware computing is *resource substitution* [3]. It is based on the observation that in most cases, alternative resources exist for a given resource. These alternatives often vary greatly in their costs (e.g., computing power, storage capacity, or energy demands), but also in their accuracy, granularity, and frequency of data updates. This directly influences their appropriateness for substitution. In general, resource substitution favours resources with a lower cost (energy requirements) over expensive (high-energy) alternatives. In many cases, a high-energy resource is not necessarily required and can be substituted without measurable impact on system performance or user acceptance [19].

The authors of [17] utilise resource substitution in the form of *sensor substitution*. In a location-based context, data from a GPS device is often substituted with triangulation or lateration data from cell towers or Wi-Fi stations. A comparable concept is *sensor triggering*, where logical dependencies between different sensors are used. When low-energy sensors detect changes in the environment, a detailed update with high-energy sensors is triggered. An experiment described in [17] shows that low-energy accelerometer data can be used to trigger high-energy GPS sampling. This approach greatly reduces the energy demand of location determination for mobile applications that do not require a gap-less reconstruction of routes. This triggering approach has also been applied in the area of civil engineering, where it is critical that autonomous sensor nodes in buildings gather highly detailed data when vibrations occur. In the "Lucid Dreaming" system [13], a low-energy analogue circuit is sufficient to watch for these environment changes. It triggers a high-energy microcontroller-based sensor to gather the required fine-grained data.

3. TERMS AND DEFINITIONS

According to [18], "*location of an object or a person is its geographical position on the earth with respect to a reference point.*" From our viewpoint, this definition is too restrictive, as geographic positions are points. In contrast to a point, a location has a spatial extent. Due to [5], "*geographic location is the text description of an area in a special confine on the earth's surface.*". However, an *area* is a set of geographical positions. So, we use a set-oriented definition: *A location is a named set of geographical positions on earth with respect to a reference point.* In a two-dimensional coordinate system, e.g., the location of a building is given as a set, where each position (point) belongs to the building's area. We do not discuss the calculation of point sets here, but rather refer to techniques of geographical information systems (GIS) [7]. The location models presented in Section 2.1 describe relationships among locations. However, reality requires a more sophisticated location model. Istanbul, as the capital of Turkey, belongs to Europe and Asia. The same issue holds for Russia, which is located in Europe and in Asia, too. Another problem results from enclaves: Kaliningrad is part of Russia, but this information is not sufficient to decide whether Kaliningrad belongs to Europe or Asia. The solution for these problems is to use set overlaps instead of containment relationships in combination with a poly-hierarchical location model [11].

Figure 2 illustrates the simplified poly-hierarchies for Istanbul and Kaliningrad, represented as directed acyclic graphs. Each node is a location and each directed edge represents that the child node belongs (semantically) to the parent node(s). Moreover, the location poly-hierarchy LPH has a unique root node because the entire coordinate system is closed in case of locations (all considerable positions are elements of the set of all positions on earth). We do not discuss the construction of LPH in this paper, but assume that an

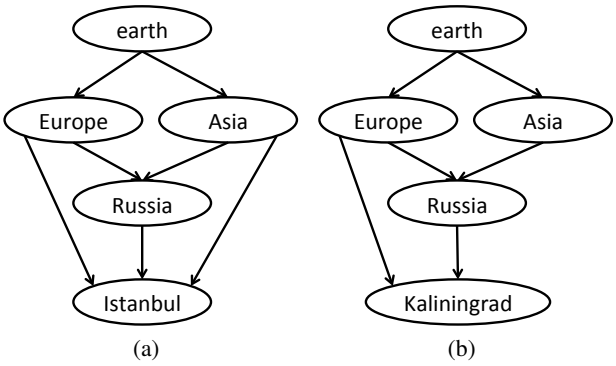


Figure 2: Poly-hierarchical example locations: Istanbul (a), Kaliningrad (b).

expert defined it in advance. Furthermore, we assume that the level of a node is defined as the number of nodes on the longest direct path from root to this node, plus one. As illustrated in Figure 2(b), $level(\textit{Russia}) = 2$ and $level(\textit{Kaliningrad}) = 3$.

4. LOCATION DETERMINATION STRATEGIES

The research question addressed in this paper is twofold: we want to continuously determine the location of an object while reducing the energy requirements for positioning, and we want to offer location information that is appropriate for different application scenarios. The two extrema are: GPS polling and not measuring at all. Polling is the most energy-intensive approach and not measuring is the most imprecise "solution". We developed three approaches for calculating appropriate location information with a minimal amount of energy. The first strategy reflects the fact that memory-intensive computations require much more energy than CPU-intensive ones [8] and reduces the number of required database lookups. The other two strategies aim for reducing the number of GPS request and lookup operations in order to find the correct path from a detected node to the root of LPH. This path correctly describes the different levels of detail for the current location.

4.1 Level Dependencies

In LPH the point sets' cardinalities of locations represented by higher-level nodes are smaller than those of locations represented by the linked lower-level nodes. The assignment of a location to a set of positions uses default GIS techniques. So, a db stores border polygons, and a db lookup fetches the proper location values from the db. Hence, one location lookup requires certain memory-intensive operations. Moreover, in case of continuous location determination, queries must be performed for each movement of the requesting object. However, if we know about the (semantic) dependencies among certain levels within the location poly-hierarchy, we can reduce the amount of energy-intensive db lookups by traversing LPH.

Figure 2(b) shows that if an object is located within Kaliningrad, it is in Russia and Europe and on earth, too. So, only one comparison of location sets is necessary. For the example in Figure 2(a), this is not as trivial. The requesting object is located in Istanbul and in Turkey. However, requesting the continent information requires an additional set comparison as Istanbul belongs to both Europe and Asia. So, the number of comparisons depends on the structure of the given poly-hierarchy. Our algorithm calculates the correct path for a given location while minimising the number of comparisons. We assume that the names of the leaf and inner nodes in LPH are unique and referenced within the GIS db.

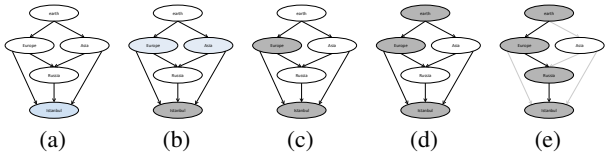


Figure 3: Traversing the location poly-hierarchy in case of Istanbul.

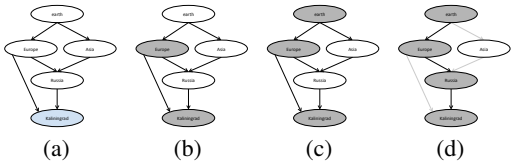


Figure 4: Traversing the location poly-hierarchy in case of Kaliningrad.

The algorithm works as follows (cf. Figures 3 and 4): At first, the proper leaf node is selected using a db query (F. 3(a); F. 4(a)). We then traverse the LPH bottom-up and mark nodes that belong to the correct path: The direct (grand)parent node(s) with the lowest level are analysed. If the current node has only one (grand)parent node in this level, this node is selected as path anchor (F. 4(b)). If more than one (grand)parent node exists on the minimal level, we check them with a db query (F. 3(b)), mark the correct one and select it as path anchor (F. 3(c)). We continue with the path anchor until we reach the root node (F. 3(d); F. 4(c)). The last step is to collect the nodes on the path from root to the leaf node including all marked inner nodes (F. 3(e); F. 4(d)).

4.2 Postponed Measurements

The postponed measurements strategy predicts the time required for a person or object to leave the current location. This time depends on the hierarchy level, the geographical model used for this level and on the movement speed [20].

For simplification purposes, the application specifies the velocity of the moving object as a movement profile (pedestrians: $\approx 6\,\mathrm{km\,h^{-1}}$, cyclists: $\approx 11\,\mathrm{km\,h^{-1}}$, car drivers: $\approx 60\,\mathrm{km\,h^{-1}}$). The most obvious approach is to take the object's current position $L_c = (x_c, y_c)$ and then calculate the minimal Euclidean distance d to all locations within the current path in LPH. For each location L on this path, we have to calculate: $\min(\sqrt{(x_c - x_l)^2 + (y_c - y_l)^2} \mid \forall (x_l, y_l) \in L)$. With a simple calculation, we then compute the time the object would need to leave this location. If, e.g., a pedestrian is 5 km away from the city limit, he or she would need at least 50 minutes ($\frac{5000\,\mathrm{m} * 3600\,\mathrm{s}}{6000\,\mathrm{m}} = 3000\,\mathrm{s}$) to leave the city. So, we postpone the next position measurement by 50 minutes if the application requires the location at a city level. The Euclidean distance guarantees the calculation of the shortest distance. Hence, if the velocity is correct, the calculation always returns a time window the moving object must be in the current location.

This approach is used for area-like locations where no route information exists. In case of a map-based location management, we utilise crossroad data to get more precise predictions (cf. [10]). Therefore, we maintain a db with all crossroads and calculate the Euclidian distance between the current position and the closest crossroad. Of course, the prediction is more exact if we use the entire map information but storing the complete maps would require much space (e.g., for the German state of Thuringia, we have to maintain 125,034 crossroads or to store and query 657 MB of OpenStreetMap data).

Besides the postponement calculation for the various levels, we have to consider the poly-hierarchy as well. Referring back to the Istanbul-Example: if the moving object is located in Istanbul, it may take one hour to leave the city, but only 10 minutes to leave the continent. The solution for this issue is to analyse the LPH in the following way. Given the LPH, the current location and the current velocity. First we calculate the correct location path using the algorithm discussed in Section 4.1. In the next step, we calculate the postponement value for each location in this path using the appropriate calculation (Euclidian distance, crossroad analysis). The minimal value determines the time until the next measurement.

4.3 Level Appropriateness

There exist various technologies to measure positions such as geotagged Wi-Fi hotspot and cell tower databases, wireless signal triangulation/lateration, or geotagging. They vary in their energy demand and their precision. While looking at the location poly-hierarchy one can recognise that locations represented closely to the root node mostly require less precise location techniques. Country information can directly be read from the country code provided by mobile telecommunications operators. The city information can be harvested from cell tower information using a simple db lookup. We have to use the most high-energy GPS only if precise location information are required. The approach discussed in the following combines the techniques illustrated in Section 4.1 and Section 4.2.

Hierarchy level	Level	Technique
Earth	0	known to be true
Continent	1	Country code + Cell tower ID lookup
Country	2	Country code
State	3	Cell tower ID lookup
City	4	Cell tower triangulation
Street	5	GPS

Table 1: Location determination lookup table.

For the postponed measurements, we adapted the cross-level postponement value in a way that we calculate different postponement values for each level while alternating the measurement strategy (cf. Table 1). Let's say that the object is located in Istanbul, and that the GPS-based measurement resulted in a postponement value of 10 minutes for the continent, and a postponement value of 1 hour for the city. After 10 minutes, we then check the appropriateness of the continent location using energy-efficient cell tower triangulation and calculate a new postponement value for this level on this basis. Hence, in best case (we do not leave the continent), we can wait with the next GPS positioning for 50 more minutes.

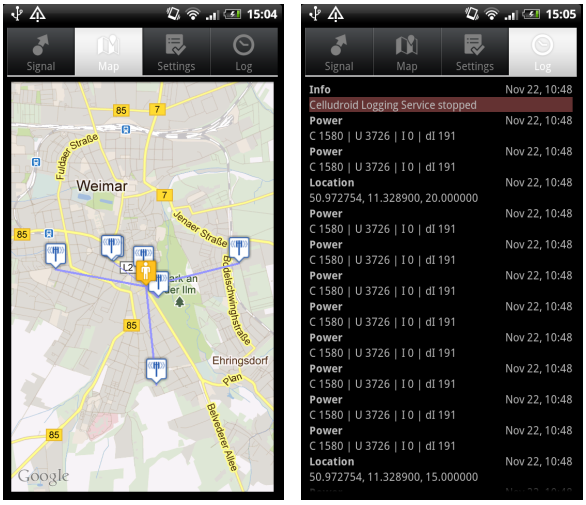
A more trivial approach supported by this idea are applications that do not need exact position information. As mentioned above, requesting the country information does not need any positioning as the information is provided by the service provider. Furthermore, less energy-intensive cell ID lookups for determining locations on the city level might be used in a polling mode. Anyway, position-based location determination such as cell tower triangulation in combination with a GIS requires too much calculation effort, if used in a polling manner. However, we will research a combination of polled and time-triggered updates of location information in a location poly-hierarchy in the near future.

5. EVALUATION

Our prototype is still work in progress. Therefore, we present a preliminary evaluation that enabled us to estimate the energy footprint of our proposed concept in an exemplary scenario.

5.1 Experimental Setup

The evaluation system was implemented as a mobile application on the Android 2.3.4 platform. We conducted our tests with the recently released HTC Sensation smartphone. As shown in Figure 5(b), the application is mainly a data logger for location and power management data. In order to gather location data, we implemented a cell tower lateration algorithm, and used the Android SDK's methods for obtaining GPS data. The application reads energy-related data (battery voltage and current) directly from the device's power management. It was implemented as an independent background logging service that gathers data even when the device is locked. The experimental setup follows our data-based energy measurement approach, as described in [12].



(a) (b)

Figure 5: Celludroid evaluation app running on an HTC Sensation smartphone.

All data was stored in an sqlite database that could later on easily be used for analysis. For convenience, the application also shows the estimated location on a map (cf. Figure 5(a)) and allows to explore the properties of nearby cell towers.

The cell tower lateration uses a Google service[1] to look the location of nearby cell towers up. Because all retrieved cell tower locations are cached in an sqlite database, subsequent location requests for previously discovered cell towers do not require additional (high-energy) network communication.

[1]Unfortunately, access to this service is not publicly documented, our implementation is based on the general process documented in this forum article: `http://stackoverflow.com/a/3356956`

Our test run consisted of a sequence of 30-minute city walks, one for each test condition:

a) baseline,

b) cell tower lateration, and

c) GPS.

In the baseline condition, the display was turned off, Wi-Fi was on, Bluetooth was off, and no background tasks except our logging service were running. During each run, power management data and cell tower locations were logged every second. GPS was requested every 5 s. However, the Android SDK cannot guarantee an exact time between GPS location updates. In fact, our measured time is slightly higher (7.2 s on average).

From the power management data, we derived electrical power and electrical energy data. In order to assess the accuracy of the gathered location data, additional processing was necessary. While the GPS data already included accuracy information, we computed accuracy information for the cell tower lateration by comparing the gathered coordinates to their counterparts from the GPS run. This enabled us to give an estimate for the upper bound of the error.

5.2 Results and Discussion

Energy

Our results indicate that cell tower triangulation has no significant impact on the device's energy demand at all. The baseline condition shows a total of 181.07 J after 10 minutes and 565.5 J after 30 minutes, while the test run with cell tower lateration resulted in a slightly higher 183.76 J after 10 minutes and 583.28 J after 30 minutes. This difference of 1.5 % is within the measuring tolerance of our method. Far more interesting is the difference between baseline

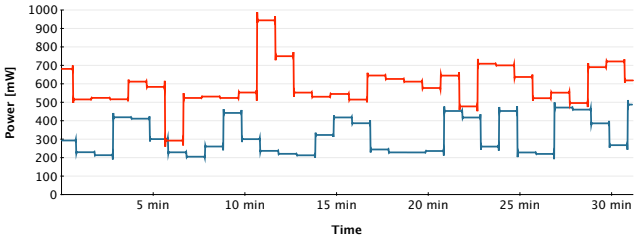


Figure 6: Comparison of power consumption for GPS (red line, top) and cell tower lateration (blue line, bottom) during continuous position determination.

and GPS condition. Figure 6 documents the power consumption progress during cell tower and GPS run. In the GPS run, the device required 316.07 J after 10 minutes and 1057.8 J after 30 minutes. Compared to baseline, this is an increase of 74.56 % after 10 minutes, or 87.06 % after 30 minutes (cf. Figure 7). Remember, GPS data was gathered every 7.2 s on average. In these 7.2 s, our setup required 4896 mJ of energy on average ($7.2\,\text{s} \cdot 680\,\text{mJ}\,\text{s}^{-1}$). In the baseline condition, 7.2 s of idling required 1296 mJ of energy on average ($7.2\,\text{s} \cdot 180\,\text{mJ}\,\text{s}^{-1}$), which is 26.5 % of the amount required for GPS.

Accuracy

The results regarding accuracy of the position determination techniques are very clear. While the GPS condition results show an average of 9.2 m, cell tower lateration performs drastically worse at 308.26 m (a difference of 3242.55 %). Some outliers in the data

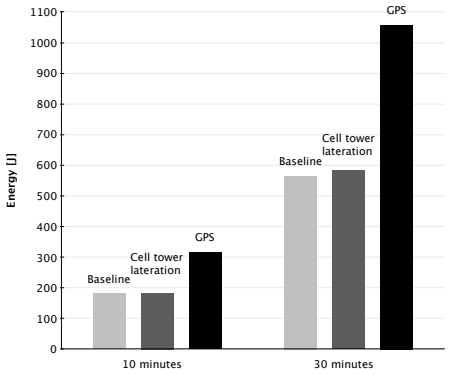


Figure 7: Comparison of accumulated energy demand.

even showed an error of more than 800 m. These extreme differences highlight the fact that the reduced energy requirements of cell tower lateration condition have an at least equally drastic influence on accuracy.

5.3 Scenario

The gathered results provide insight into the areas of application where sufficient potential exists to reduce the energy requirements for continuous location determination. In this subsection, we sketch a scenario that relies on our three conceptual pillars:

a) using level dependencies,

b) postponed measurements,

c) using appropriate positioning techniques.

This scenario-based evaluation surely cannot serve as a proof for our proposed concept, but it provides a glimpse on its possible impact. Our scenario application is a mobile tourist information system for smartphones. All data is stored on the mobile device and can be accessed using db queries. The application can access the following device's location sensors:

a) SIM card operator's country code (country information),

b) Cell tower association (state information),

c) Cell tower lateration (city part/street information),

d) GPS (street number information).

In our scenario, a tourist from Japan is on a bus tour through Germany and is currently visiting Thuringia. In Weimar, she decides to start using the tourist information system.

Using level dependencies

Upon first use, the system requires a single positioning with cell tower lateration. With the gathered coordinates, the country, state, and city nodes of the location poly-hierarchy can be determined: earth→Europe→Germany→Thuringia→Weimar. The system uses this data to switch to the tour mode for Thuringia.

Using appropriate techniques according to level

In this mode, a continuous perimeter search delivers points of interest, such as restaurant, museums, public places, or theaters. Because this feature at this point only requires the general information about the presence of such points of interest (and not a detailed routing information on how to get there), it is appropriate to rely on cell tower lateration.

Postponing unnecessary measurements

When the tourist finished exploring the city, she wants to find a nice place to have lunch. After selecting one of the restaurants from the perimeter search result list, the tourist information system enters the navigation mode. In this mode, a database of intersections is used in combination with a movement profile to estimate important turning points along a route where it is necessary to activate the GPS technique.

Estimation of reduced energy requirements

Upon first use, at least one GPS request can be saved. As we have shown in the previous subsection, each request requires about 4896 mJ. The continuous perimeter search for points of interest took 2 hours in our scenario. During these 2 hours, a continuous GPS polling would have required about 1000 GPS requests (4896 J). In contrast, the cell tower lateration used by the system only required 1296 J. Giving an estimation of the reduction achieved by the postponed measurements in the navigation mode (using the intersection database) is very difficult, as it relies on a multitude of factors (e.g., velocity, distance between intersections). A conservative lower bound would be to assume that this method reduces the amount of GPS requests to 50 %.

6. SUMMARY, CONCLUSIONS, AND OUTLOOK

We addressed two research questions in the area of continuous location determination: We analysed precision appropriateness and discussed energy requirements. We utilised the poly-hierarchical nature of locations to provide different levels of location information. We discussed three approaches: We reduced the amount of location calculations within the location poly-hierarchy, we introduced the concept of postponed measurements, and we discussed appropriateness issues in location poly-hierarchy levels.

The overall goal behind our research is to realise a location framework that application developers can use to implement location-aware applications without the need to take care of the high energy requirements of GPS. We know that various frameworks do exist and address the same issue. However, our results show the potential to reduce the required energy further. The paper opens various research directions that we will follow on: The concept of location poly-hierarchies requires a detailed and formal definition. While location hierarchies and ontology-based models are well researched, poly-hierarchies are rather novel. Furthermore, more research needs to be done in the area of calculating the postponement values, and the combination of polling and postponed updates must be addressed.

7. REFERENCES

[1] C. Becker and F. Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[2] M. Beigl, T. Zimmer, and C. Decker. A location model for communicating and processing of context. *Personal Ubiquitous Computing*, 6:341–357, Jan. 2002.

[3] C. Bunse and H. Höpfner. Resource substitution with components — optimizing energy consumption. In *ICSOFT '08 Proc.*, volume SE/GSDCA/MUSE, pages 28–35. INSTICC press, July 2008.

[4] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. P. Cox. Energy-Efficient Localization via Personal Mobility Profiling. In *MobiCASE '09 Proc.*, pages 203–222. Springer, 209.

[5] Z. Dongqing, L. Zhiping, and Z. Xiguang. Location and its Semantics in Location-Based Services. *Geo-spatial Information Science*, 10(2):145–150, June 2007.

[6] F. Dürr and K. Rothermel. On a location model for fine-grained geocast. In *UbiComp '03 Proc.*, pages 18–35. Springer, 2003.

[7] I. Heywood, S. Cornelius, and S. Carver. *An Introduction to Geographical Information Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, Nov. 2002.

[8] H. Höpfner and C. Bunse. Towards an energy-consumption based complexity classification for resource substitution strategies. In *GVDB '10 Proc.*, volume 581. CEUR-WS.org, May 2010.

[9] H. Höpfner and C. Bunse. Energy Awareness Needs a Rethinking in Software Development. In *ICSOFT '11 Proc.*, volume 2, pages 294–297. SciTePress, 2011.

[10] H. Höpfner and M. Schirmer. Energy efficient continuous location determination for pedestrian information systems. In *MobiDE '12 Proceedings*, 2012. accepted for publication.

[11] H. Höpfner and M. Schirmer. Towards modeling locations as poly-hierarchies. In *GVBD '12 Proc.*, May 2012. accepted for publication, forthcoming.

[12] H. Höpfner, M. Schirmer, and C. Bunse. On measuring smartphones' software energy requirements. In *ICSOFT '12 Proc.*, 2012. accepted for publication, forthcoming.

[13] S. Jevtic, M. Kotowsky, R. P. Dick, P. A. Dinda, and C. Dowding. Lucid dreaming: Reliable analog event detection for energy-constrained applications. In *IPSN 'ß07 Proc.*, pages 350–359. ACM, 2007.

[14] C. Jiang and P. Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing. In *Ubicomp '02 Proc.*, pages 246–263. Springer, 2002.

[15] T. Kauppinen, R. Henriksson, R. Sinkkilä, R. Lindroos, J. Väätäinen, and E. Hyvönen. Ontology-based Disambiguation of Spatiotemporal Locations. In *IRSW '08 Proc.*, volume 422. CEUR-WS.org, 2008.

[16] B. N. Schilit, A. LaMarca, G. Borriello, W. G. Griswold, D. McDonald, E. Lazowska, A. Balachandran, J. Hong, and V. Iverson. Challenge: ubiquitous location-aware computing and the "place lab" initiative. In *WMASH '03 Proc.*, pages 29–35. ACM, 2003.

[17] M. Schirmer and H. Höpfner. SenST: Approaches for Reducing the Energy Consumption of Smartphone-Based Context Recognition. In *CONTEXT '11 Proc.*, pages 250–263. Springer, 2011.

[18] A. Y. Seydim, M. H. Dunham, and V. Kumar. Location dependent query processing. In *MobiDE '01 Proc.*, pages 47–53. ACM, 2001.

[19] J. P. Sousa, R. K. Balan, V. Poladian, D. Garlan, and M. Satyanarayanan. User Guidance of Resource-Adaptive Systems. In *ICSOFT '08 Proc.*, volume SE/GSDCA/MUSE, pages 36–44. INSTICC press, 2008.

[20] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *SSDBM '98 Proc.*, pages 111–122. IEEE CS, 1998.

[21] J. Ye, L. Coyle, S. Dobson, and P. Nixon. A unified semantics space model. In *LoCA '07 Proc.*, LNCS, pages 103–120. Springer, 2007.

[22] P. A. Zandbergen. Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning. *Transactions in GIS*, 13(s1):5–26, July 2009.

Datenbank-Performance-Experimente in der Lehre

Max Flügel, Alexander Stein, Michael Höding
FH Brandenburg
Magdeburger Straße 50
14770 Brandenburg/Havel
(+49)3381 355 243
[fluegel|steina|hoeding]@fh-brandenburg.de

ABSTRACT
In diesem Beitrag stellen wir unsere Idee und deren Ansätze vor, die die Ausbildung im Bereich Datenbank-Performance mittels eines Praktikums unterstützen sollen. Wichtiges Element des Erkenntnisgewinns ist das eigene Erleben im Rahmen von Experimenten. Im Bereich Datenbank-Performance müssen Experimente und Experimentierumgebung so gestaltet und aufgebaut sein, dass den Erwartungen an Vergleichbarkeit und Wissenszuwachs entsprochen wird.

Keywords
Database Performance, Praktikum, Experiment

1. Einführung und Motivation
Datenbank-Performance ist in Forschung, Lehre und Praxis ein zentraler Betrachtungsgegenstand. Betriebliche Anwendungssysteme müssen durch akzeptable Antwortzeiten einen reibungslosen Betrieb ermöglichen. Datenbankforschung und Datenbankhersteller haben hierzu eine Vielzahl von Methoden und Mechanismen entwickelt und eingeführt. Eine Aufgabe der Lehre ist es, diese Methoden zu vermitteln, um ihre Anwendung zu unterstützen. Zwar gibt es zunehmend Self-Tuning-Mechanismen in Datenbankmanagementsystemen (DBMS), jedoch ist der gut ausgebildete Datenbankadministrator ein wichtiges Erfolgselement des soziotechnischen Gesamtsystems.

Um Studierende für ein Thema zu motivieren und auch zu begeistern, muss ihnen die Möglichkeit gegeben werden, Erkenntnisse in der praktischen Arbeit zu erlangen. Hierzu können Experimente eingesetzt werden, welche sich unter anderem durch selbstständiges Arbeiten auszeichnen. Der Student wird durch eine Aufgabenstellung mit einem in der Theorie beschriebenen Problem bekannt gemacht. Das Experiment hat in seiner Struktur einen festen Rahmen. Dieser unterstützt den Studenten ein Thema methodisch zu betrachten (vgl. [2] S.239). Zusätzlich erlernt der Student neben neuem Wissen auch soziale Kompetenzen (Teamfähigkeit, Konfliktlösung, usw.).

24th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012-01.06.2012, Lübbenau, Germany.

1.1 Problem
In Unternehmen, in denen leistungsstarke Datenbanken und ein gutes Antwortzeitverhalten eine Rolle spielen, kommt es vor, dass bezüglich der Datenbank-Performance voreilige und einseitige Entscheidungen getroffen werden und somit die sorgfältige Betrachtung der Datenbank-Performance zu kurz kommt.

In der Lehre wird oftmals zu wenig Aufschluss über dieses Thema gegeben bzw. in Veranstaltungen lediglich darauf hingewiesen, dass Datenbank-Performance unter anderem bei Unternehmen eine große Rolle spielt oder spielen kann. Welche Beiträge, sprich welche Methoden und/oder Werkzeuge, für eine optimale Performance-Auslastung eingesetzt werden können und wie mit potentiellen Leistungsdefiziten umgegangen werden kann, wird im Rahmen der Lehre viel zu selten betrachtet. Eine Vertiefung durch eigene Übungen findet zudem i.d.R. nicht statt.

Ein weiteres Problem, welches mit der zuvor genannten Sachlage einher geht, ist, dass das Thema Datenbank-Performance als sehr abstrakt und wenig kalkulierbar bzw. greifbar von vielen Studenten eingeschätzt wird. Es ist nicht genau klar, wie die Performance dargestellt und beeinflusst werden kann.

1.2 Zielstellung
Für die genannten Problemfelder sollen in der Lehre ein höherer Stellenwert und die notwendige Akzeptanz erreicht werden. Dabei soll nicht nur die Datenbank-Performance als solche beleuchtet werden. Auch Wirtschaftlichkeitsbetrachtungen der Performance sollen mit einfließen. Kernziel ist es, mögliche Umgangsweisen mit der Performance von Datenbanken deutlich zu machen.

Erreicht werden kann dieser Ansatz durch das praktische Arbeiten mit *Datenbank-Performance-Experimenten in der Lehre*.

In mehreren Übungsveranstaltungen sollen die Studenten eigenständig Praxiserfahrungen sammeln können, um Datenbank-Performance selbst zu erleben. Hierfür soll aufgezeigt werden, wie die Performance gemessen und interpretiert werden kann. Im Nachgang werden Methoden und Werkzeuge vorgestellt, mit denen die Performance optimiert wird. Denkbar wäre auch, dass die Studenten selbst nach geeigneten Möglichkeiten suchen und diese dann anwenden müssen.

Ziele sind demnach:

- Performance erleben
- Performance-Messmethoden kennenlernen und einsetzen

- Gesamtarchitektur betrachten (beispielsweise zugrundeliegendes System) und analysieren, um „Flaschenhälse“ zu erkennen
- Optimieren der Performance
- Wirtschaftlichkeit von Performance-Lösungen betrachten

Dabei werden fachliche als auch überfachliche Fähigkeiten ausgeprägt.

2. Theoretische Grundlagen

Um Performance zu erleben, sollen Performance-Tests und die didaktische Methode Experiment miteinander verbunden werden. Im folgenden Abschnitt wird auf Performance-Tests und ihre Rolle in den Systemtests eingegangen. Im Anschluss daran werden das Experiment in seiner Methode und die wesentlichen Phasen beschrieben. Abschließend sollen die Zusammenhänge zwischen beiden Themenfeldern aufgezeigt werden.

2.1 Performance-Test

„Die Performance oder Leistung in der Informatik ist die Fähigkeit eines Systems, eine Aufgabe in einer bestimmten Zeit zu erledigen“ ([4] S. 439). Dabei wird Performance durch den Anwender wahrgenommen und muss entlang der Zielgruppen-Anforderungen betrachtet werden (vgl. [4] S. 439). So werden Wartezeiten bei Datenanalysen durch den Nutzer eher akzeptiert, als die Wartezeit bei Internetsuchmaschinen. Die Performance eines Systems lässt sich hingegen durch Tests systematisch nachweisen und im Anschluss gegebenenfalls an die Anforderungen anpassen. Dabei gehören die Performance-Tests zu den Systemtests. Systemtests werden nach H. Balzert in Funktionstest, Leistungstest, Benutzbarkeitstest, Sicherheitstest und Interoperabilitätstest unterteilt. Diese Tests dienen zur Bestimmung der Produktqualität eines Softwaresystems (vgl. [5] S. 503 ff.). Durch Leistungstests können zwei grundsätzliche Fragestellungen beantwortet werden:

- Wie viele Daten kann das System verarbeiten?
- Wie lange braucht das System für die Datenverarbeitung?

Zusätzlich können hier auch die Dimensionen der Belastung eingegliedert werden. Hierzu zählen die Tests unter normalen Bedingungen des Regelbetriebs, das Testen im Grenzbereich durch Lasttests und das bewusste Überschreiten des Grenzbereichs durch Stresstests.

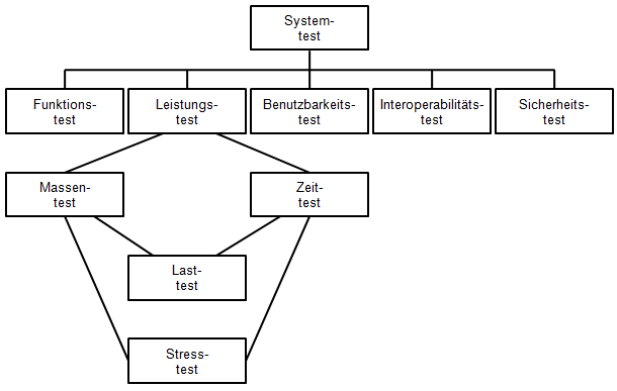


Abbildung 1: Systemtest

Die jeweiligen Tests lassen sich dabei in die Phasen Vorbereitung, Durchführung und Nachbereitung aufteilen (vgl. [6] S. 253). Die Tests werden entsprechend dokumentiert, um daraus Maßnahmen abzuleiten. In der Vergangenheit haben sich für System- bzw. Softwaretests verschiedene Normen und Standards entwickelt. Hier ist unter anderem die DIN 66273 (Vorgehensmodell für die Lastmessung) zu nennen.

2.2 Aufbau des Praktikums

Um Performance für die Studenten erlebbar zu machen, wird begleitend zur Vorlesung ein Praktikum durchgeführt. Dieses Praktikum teilt sich in einzelne Experimente auf. Die Experimente bauen in der Folge aufeinander auf, so erhalten die einzelnen Experimente einen roten Faden. Es werden die wichtigsten Eckpunkte der Performance-Messung berücksichtigt. Innerhalb der einzelnen Experimente sollen bewusst die Stellschrauben an den Systemen geändert werden, um die Wirkung der einzelnen Variablen auf das Verhalten des Systems zu erkennen und anschließend zu bewerten. Der Student soll hierbei selbstständig die Werte der Variablen ändern können.

Das Experiment lässt sich gemäß K. Reich in drei Phasen aufteilen (vgl. [3] S. 7):

- Am Beginn steht die Planungsphase. Hier wird die Vorbetrachtung umgesetzt. Zudem wird die Fragestellung formuliert, auf deren Basis eine Hypothese aufgestellt werden kann. Weiterhin werden die Randbedingungen des Experiments beschrieben und der Versuchsaufbau geplant.
- Dann erfolgt die Durchführungsphase. In dieser Phase wird die Planung umgesetzt, indem der Praktikumsversuch durchgeführt wird.
- Nach der Durchführung des Experiments erfolgt die Nachbetrachtung bzw. Auswertungsphase. Hier werden die Ergebnisse überprüft und hinsichtlich der potentiellen Fehlerquellen analysiert.

Daraus wird für die praktische Umsetzung folgender Ablauf abgeleitet:

1. Aufgabe stellen; durch den Lehrenden
2. Vorbetrachtung; durch den Studierenden
3. Überprüfung; durch den Lehrenden
4. Praktikumsversuch; durch den Studierenden
5. Nachbetrachtung; durch den Studierenden
6. Evaluation des Experiments; durch den Lehrenden

Bei der Formulierung der Aufgabenstellung sind zwei Extreme denkbar. Zum einen besteht die Möglichkeit, die Aufgabe völlig offen und ohne Restriktionen zu stellen, z.B. „Untersuchen Sie das System in Bezug auf CPU, Arbeitsspeicher und Storage!“. Zum anderen kann durch eine geschlossene Aufgabenstellung die Aufgabe so klar formuliert sein, dass der Student einen fest vorgeschriebenen Weg beim Abarbeiten der Aufgabe einschlägt und garantiert zum Ziel kommt. Beide Varianten haben nach Meinung der Autoren ihre Vor- und Nachteile (siehe Tabelle „Vor- und Nachteile bei der Aufgabenstellung“). Die zweckmäßigere Variante soll im Rahmen des Projekts ermittelt werden.

Tabelle 1: Vor- und Nachteile bei der Aufgabenstellung

Offene Aufgabenstellung	Aufgabenstellung mit Restriktionen
+ Der Student kann sich frei entfalten	+ Ergebnis ist schnell überprüfbar
+ Neue Erkenntnisgewinne	+ Ergebnis ist nachvollziehbar
+ Vielfältige Ergebnisse sind möglich	+ Ergebnisse sind vergleichbar
- Ergebnis ist ungewiss	- Erkenntnisgewinn ist vorgeschrieben
- Ergebnis und Experiment evtl. nicht nachvollziehbar	- Experiment lässt wenig Platz für Eigeninitiative
- Ergebnisse eventuell schwer vergleichbar	- Gänzlich neue Erkenntnisse/Lösungswege sind kaum möglich

Folgende Themengebiete sind beispielsweise für das Praktikum denkbar:

- Erkundung der Systemeigenschaften
- Einfluss der SQL-Statements
- Nutzen und Wirkung von Indexen
- Arbeiten mit Puffern

Die Experimente werden durch den Studenten vorbereitet. In dieser ersten Phase müssen sich die Studenten die Aufgabenstellung genau versinnbildlichen und eine Zielstellung definieren. Im Anschluss daran gilt es den Lösungsweg zu planen. Dabei sollen theoretische Grundkenntnisse und das spezifische Fachwissen aus den Lehr- und Übungsveranstaltungen ebenso dienen, wie Kenntnisse und Fähigkeiten, welche sich die Studenten für die Aufgabenerfüllung selbstständig aneignen müssen. Desweiteren sollen die Prämissen der Planungsphase laut K. Reich zum Tragen kommen (siehe oben), d.h. die Studenten sollen mittels des erlangten Wissens eine Hypothese parallel zur Aufgabenstellung formulieren und diese Annahme dann durch ihren Praktikumsversuch entweder bestätigen oder widerlegen. Wenn eine Aufgabe beispielsweise verlangt eine bestimmte Datenmenge aus einer Datenbank einerseits mit einem SQL-Statement und andererseits mit Programm-Code (z.B. PHP, Java) abzufragen, dann könnten die Studenten hier die Hypothese aufstellen, dass das Abfragen mittels SQL performanter als das Abfragen mit Programm-Code ist. Als Ergebnis der Vorbetrachtung entsteht der Protokollentwurf.

Erst bei ausreichender Vorbetrachtung des Experiments, wird die Durchführung durch den Lehrenden möglich gemacht, hierzu erfolgt eine Überprüfung. Während des Praktikumsversuchs wird der Vorplanung entsprochen und die Aufgabenstellung praktisch gelöst. Die Ergebnisse werden in einem (Mess-)Protokoll festgehalten.

Nach erfolgreicher Durchführung des Praktikumsversuchs, werden die Ergebnisse in der Auswertungsphase (Nachbetrachtung) zusammengefasst und beurteilt. Die Beurteilung erfolgt hinsichtlich der in der Vorbetrachtung getroffenen Hypothese. Hier soll der Student durch eine ehrliche und realistische Einschätzung seiner ursprünglichen Annahme das Resultat seines Versuchs bewerten. An dieser Stelle sollte auch eine Fehleranalyse mit einfließen. Neben der technischen Beurteilung zählen auch die Wirtschaftlichkeitsbetrachtungen (Verhältnis zwischen Nutzen und Aufwand/Kosten) zu den Erfordernissen dieser Phase.

Abschließend wird das Experiment vom Lehrenden evaluiert. Zum einen wird im Rahmen der Evaluation die Einhaltung der formalen Vorgehensweise geprüft, hier die Phasen des Experiments und die Dokumentation. Zum anderen wird die Plausibilität begutachtet. An dieser Stelle müssen die Vor- und Nachbetrachtung in einem logischen Zusammenhang stehen, d.h. wurde in der Nachbetrachtung auch auf genau das eingegangen, was in der Vorbetrachtung geplant wurde. Das setzt natürlich voraus, dass in der Durchführungsphase auch das umgesetzt wurde, was in der Vorbetrachtungsphase angesetzt wurde.

Der Evaluationsaufwand steigt natürlich auch mit steigendem Grad einer offenen Aufgabenstellung. Hier ist ganz besonders darauf zu achten, wie die Studierenden vorgegangen sind und wie plausibel der eingeschlagene Lösungsweg ist. Näher betrachtet werden muss an dieser Stelle der wissenschaftliche Anspruch des Lösungswegs.

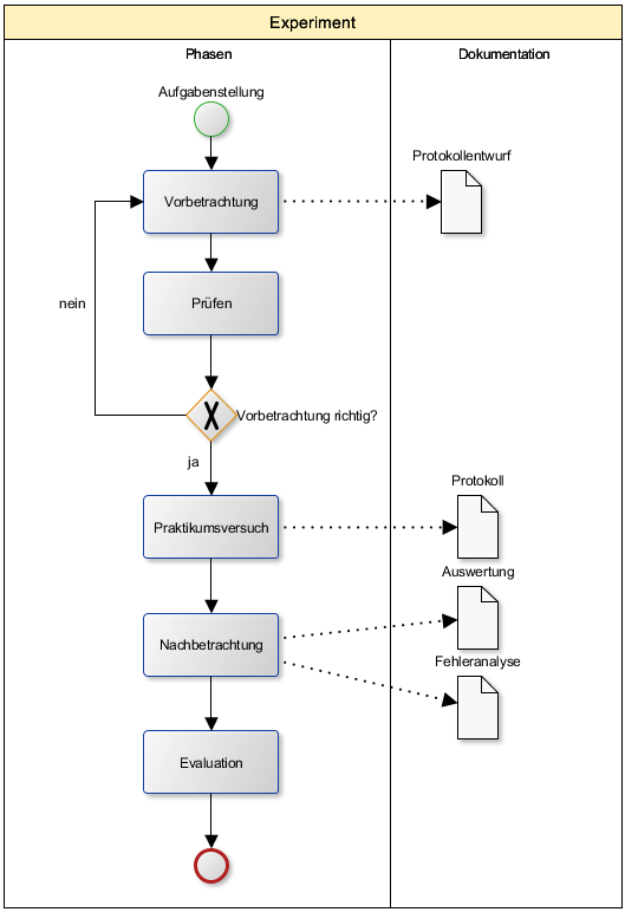


Abbildung 2: Ablauf eines Experiments

Für die Durchführung der Praktikumsversuche soll ein im Vorfeld festgelegter Wochentag, der Versuchstag, dienen. Die Festlegung auf einen solchen Versuchstag und den übrigen „normalen“ Tagen erfolgt einerseits aus lehrbedingten und andererseits aus technischen Gründen. Die Studenten sollen Erfahrungen mit einem technisch „großen“ Umfeld sammeln können. Das bedeutet, dass sie die Performance von umfangreichen Datenbanken in leistungsstarken Experimentierumgebungen (Servern) testen sollen. An den Versuchstagen stehen diese Experimentierumgebungen zur Verfügung. An den übrigen Tagen haben die Studenten Zugriff auf wesentlich „kleinere“ Umgebungen, um ihre Praktikumsversuche vorbereiten bzw. nach den Versuchen Nacharbeiten durchführen zu können. Genaueres zum Zusammenspiel der beiden Umgebungsarten ist im vierten Kapitel nachzulesen.

Die Zeit zwischen den Versuchstagen soll durch den Studenten für die Vorbereitung und Nachbereitung seines Versuchs dienen. Die jeweiligen Schritte der Experimente werden durch den Studenten dokumentiert und lassen sich zum Abschluss des Praktikums als Basis für die Prüfungsnote nutzen.

2.3 Zusammenhang zwischen Performance-Test und Experimenten

In der Informatik kann man Performance-Tests zu den Experimenten zählen (vgl. [7] S. 233 ff.). Wie in jedem Test folgen auch Performance-Tests einer bestimmten Abfolge. Tests werden geplant und im Anschluss durchgeführt. Die Ergebnisse des Tests werden dokumentiert und ausgewertet. Genauso wie Experimente müssen Tests auch vergleichbar und reproduzierbar sein. Wenn man diesem allgemeinen Vorgehen folgt, lassen sich hier Parallelen zum Experiment ziehen. Die Studenten sollen diese Parallelen erkennen und sie für den Aufbau der Experimente nutzen.

3. Organisation der Lehre

Nachdem die Begriffe Experiment und Performance-Test beschrieben sind, soll nun gezeigt werden, wie das Experiment die Lehre beeinflusst. Es wird dabei auf die Aufteilung der in der Lehrveranstaltung teilnehmenden Studenten und die daraus resultierenden Gruppenstärken ebenso eingegangen, wie auf die Rolle des Lehrenden und des Studenten bei dieser Form der praktischen Übung

3.1 Aufteilung der Lehrveranstaltung

Für die Experimente sollen sich die Studierenden in Gruppen zusammenfinden. Die Arbeit im Team wird erfahrungsgemäß bessere Ergebnisse hervorbringen als Einzelarbeit, da sich die Studenten mit ihren unterschiedlichen Wissensständen austauschen und somit ergänzen können. Zudem ist die Bildung von Gruppen der Ressourcenplanung dienlich, wie in Kapitel vier ersichtlich wird.

In der Fachhochschule Brandenburg im Studiengang Wirtschaftsinformatik wird pro Semester mit einer Studierendenzahl von bis zu 80 Studenten ausgegangen. Dabei ist eine Aufteilung in maximal 20 Gruppen noch handhabbar. So können diese 20 Gruppen ihre Praktikumsversuche (am Versuchstag) in maximal 4 Blöcken á 90 Minuten durchführen. Es ergibt sich folgende Matrix der Gruppenstärke und Aufteilung:

Tabelle 2: Aufteilung der Gruppen

		Studierendenzahl		
		40	**60**	**80**
Gruppenstärke	**2**	20	30	40
	3	12*3+1*4	20	24*3+2*4
	4	10	15	20
	5	8	12	16

Es wird für die weitere Planung von einer maximalen Gruppenzahl von 20 Gruppen ausgegangen, d.h. bei einer Studierendenzahl von 40-80 Studierenden eine Gruppenstärke von 2-4 Studierende je Gruppe. Innerhalb der einzelnen Gruppen müssen dann die aus dem Einzelexperiment abgeleiteten Aufgaben entsprechend aufgeteilt werden. Ist die Studierendenzahl über 80, muss die Gruppengröße auf 5 erhöht werden, um die Gesamtgruppenzahl von 20 nicht zu überschreiten.

3.2 Rolle des Studierenden

Der Studierende wird während der Experimente selbst aktiv. Durch entsprechende theoretische Vor- und Nachbetrachtungen wird er ein optimales Verhältnis zwischen Theorie und Praxis erhalten. Er schult neben den rein fachlichen Fähigkeiten auch andere Kompetenzen. So muss er gerade in der Zusammenarbeit mit anderen Studierenden seine Kommunikationsfähigkeit schulen und auch sich selbst und die Gruppe in der er arbeitet organisieren. Bei der jeweiligen Gruppenorganisation wird vorausgesetzt, dass sich die Studierenden eigenständig für die Aufgaben einteilen, bei denen ihre fachlichen Stärken liegen (z.B. Dokumentation, Programmierung, System-Administration). Als primäre Anforderung an den Studierenden muss dessen Interesse, neue Sachverhalte zu entdecken, geweckt werden (vgl. [3] S. 12). Die Fähigkeit, eventuell auftretende Konflikte frühzeitig zu erkennen und diese abzustellen, wird als sekundäre Anforderung angesehen.

3.3 Rolle des Lehrenden

Dem Lehrenden wird während des Experiments eher die passiv unterstützende Rolle zu teil. Er führt in den Vorlesungen in die Thematik ein und vermittelt dabei die theoretischen Grundlagen (vgl. [3] S. 11). Während der Experimente wird er beratend zur Seite stehen und dem Studenten bei Bedarf Denkanstöße geben. Er muss nach der Vorbereitungsphase eine erste Überprüfung des Aufbaus und des Experiments durchführen, bevor er den Praktikumsversuch durchführen lässt. Es wäre hier vorstellbar Lernplattformen wie Moodle [11] einzusetzen, um in einem kurzen Wissenstest die Vorbereitung zu überprüfen. Auch die Ergebnisse des Praktikumsversuchs (z.B. das Messprotokoll) können via Moodle verwaltet und überprüft werden.

4. Ressourcenplanung

Um den Studentengruppen die Möglichkeit der praktischen Arbeit zu geben, sind Experimentierumgebungen auf und mit speziellen Systemen (Hardware und Software) notwendig.

4.1 Einsatz von Virtualisierung

Für die Lehre der Messung von Datenbank-Performance ist es wichtig, dass in einem Umfeld gearbeitet wird, in dem keine oder nur sehr wenige Leistungsschwankungen bzgl. der Hardware zu erwarten sind. Grundsätzlich kann es eine mögliche Lösung sein, in die bestehende Infrastruktur der Hochschule einen zusätzlichen Server oder ein Server-Blade zu integrieren, welcher von anderen Servern zumindest Software-seitig abgekapselt wird.

Nun stellt das gleichzeitige Arbeiten der Studentengruppen auf einem und demselben Server keine gute Lösung dar, da kaum feststellbar ist, welche Server-Ressource oder Software gerade von einer bestimmten Gruppe beansprucht wird. Hier sind ein eindeutiges Messen der Performance und somit vergleichbare Experimente nicht möglich.

Die Lösung stellt hier die Server-Virtualisierung dar. „Die Virtualisierung abstrahiert die physische Hardware vom Betriebssystem. Die klassische 1:1-Verbindung von Hardware und Betriebssystem, die man vor allem in der PC-Welt kennt, wird aufgehoben; dazwischen wird eine neue Schicht gelegt, die diese Abstraktion vornimmt, die Virtualisierungsschicht.“ ([10] S. 33) Die Aufgabe des Virtualisierens für die Experimentierumgebung übernimmt das VMware-Produkt vSphere mit seinem ESX Server (siehe nachfolgende Abbildung).

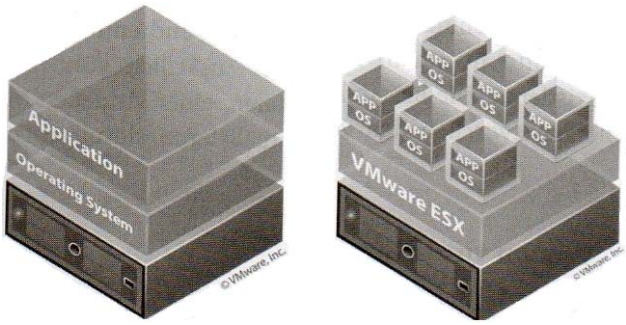


Abbildung 3: Konventioneller Server-Aufbau (links) versus Server-Virtualisierung (rechts) ([10] S. 33)

Durch die Virtualisierung können auf dem Server mehrere virtuelle Maschinen (VMs) eingerichtet werden, innerhalb derer die Studenten dann ihre Experimente unter gleichen Rahmenbedingungen (Vereinheitlichung der Hardware bzw. virtuellen Hardware) durchführen können. Jede VM erhält dann ihr eigene Softwareumgebung, wie Betriebssystem (engl.: Operating System oder kurz: OS), Datenbank und weitere Applikationen (App), wie beispielsweise Performance-Tools. Dies ist in der obigen Abbildung durch die kleinen Würfel dargestellt.

Die zum Einsatz kommenden VMs lassen sich in Entwicklungssysteme und Versuchssysteme unterteilen. Die Systeme unterscheiden sich im Wesentlichen durch ihr Leistungsspektrum und ihre Benutzung.

Für die Planung des Versuchsaufbaus kommen Entwicklungssysteme zum Einsatz. Das sind kleine VMs, sprich VMs mit einer geringeren Leistung und Konfiguration. Jede Gruppe erhält eine solche Entwicklungs-VM an der sie Änderungen vornehmen kann bzw. bestimmte Teilaufgaben bereits im Vorfeld testen kann. Diese Systeme stehen den Gruppen zwischen den einzelnen Versuchstagen zur Verfügung. Die Entwicklungssysteme erhalten eine kleine Datenbank. Während die Entwicklungs-VMs aktiv sind, bleiben die Versuchs-VMs inaktiv (siehe nachfolgende Abbildung).

Die eigentlichen Praktikumsversuche werden auf den Versuchssystemen mit großen VMs realisiert. Diese VMs besitzen eine große Datenbank und deutlich mehr Leistung und ermöglichen Performance-Tests in großem Umfang. Zeitlich werden die Versuchs-VMs so geschaltet, dass an den Versuchstagen die Entwicklungs-VMs der Gruppen abgeschaltet und die Versuchs-VMs aktiviert werden.

Nach den Praktikumsversuchen, werden die Systeme entsprechend umgedreht wieder aktiviert bzw. deaktiviert.

In der nachfolgenden Abbildung sind die schematische Aufteilung der unterschiedlichen VM-Gruppen und deren Aktivität ersichtlich.

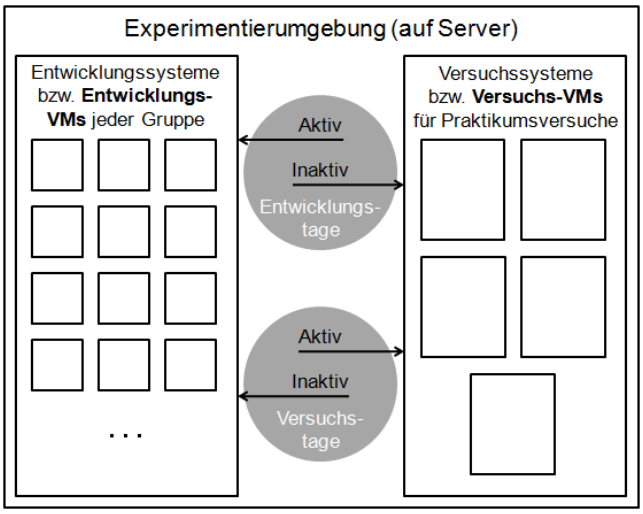


Abbildung 4: Aufbau der Experimentierumgebung

Durch Anfertigen von Clones (Klone) der ersten angelegten VM pro VM-Gruppe kann schnell die gewünschte VM-Anzahl erreicht werden, ohne jede VM einzeln auf herkömmlichem Wege anlegen zu müssen.

Die zugrundeliegenden Ressourcen, sprich die physische Hardware des Servers, wird unter den jeweils aktiven VMs fest aufgeteilt.

Mit der Virtualisierungs-Lösung können folgende positive Nebeneffekte für das Praktikum erreicht werden:

- Performance-Steigerung der Systeme
- Berücksichtigung von Aspekten der Green-IT (Einsparung von Hardware, Verringerung des Stromverbrauchs und Kühlaufwands)
- Gute Administrierbarkeit der Experimentierumgebung (Änderungen an den VMs, Zurücksetzen der VMs durch Snapshots) ([10] S. 34 ff.)

Die genannten Nebeneffekte können den Studenten in der Lehre kommuniziert bzw. selbst von ihnen erfahren werden.

Zudem kann als positiv angesehen werden, dass bei eventuellen Schäden am System lediglich eine VM betroffen sein wird und nicht der Server bzw. die Hardware direkt. Die beschädigte VM kann dann wieder in den Urzustand zurückgesetzt werden.

4.2 Konfiguration der virtuellen Maschinen

Ausgehend von der Gruppenplanung aus dem Abschnitt 3.1 kann die nachfolgende Konfiguration für die VMs abgeleitet werden. Hierbei wird lediglich auf die Zusicherungen von Arbeitsspeicher und Festplattenkapazität eingegangen.

Den Entwicklungs-VMs wird während der Entwicklungszeit 4 GB Arbeitsspeicher zugesichert. Daraus resultiert ein Arbeitsspeicherbedarf von 80 GB. Wird an dieser Stelle mit einer Reserve kalkuliert, kommt man, in Anbetracht der gängigen Konfiguration von RAM-Modulen, auf einen erforderlichen Arbeitsspeicher von 96 GB. Mit den 16 GB Reservespeicher kann unter anderem die Gruppenzahl bei Bedarf erhöht werden (4 Gruppen mehr mit wiederum jeweils 4 GB RAM). Die Entwicklungs-VMs werden mit einer kleinen Datenbank mit max. 4 GB Datenvolumen ausgestattet (für vorbereitende Tests). Weiterhin wird zusätzlicher Storage von 6 GB für das Betriebssystem, das DBMS und weitere Tools (Performance-Tools) beansprucht. Eine Reserve von 2 GB Speicher wird ebenfalls eingeplant. Daraus folgt für jede Entwicklungs-VM ein Storage-Bedarf von 12 GB. Bei 20 Gruppen bedeutet das einen Storage von 240 GB. Rechnet man die 4 potentiellen Entwicklungssysteme aus der Reserve hinzu, führt dies zu einem Speicherbedarf von 288 GB für die Entwicklungs-VMs.

Für die Versuchs-VMs sollen, wie in Abbildung 4 erkenntlich ist, 5 Systeme zur Verfügung stehen. Da zu den Versuchstagen sämtliche Ressourcen den Versuchs-VMs zur Verfügung stehen sollen, kann der gesamte Arbeitsspeicher (96 GB) entsprechend aufgeteilt werden. Das führt zu einem Arbeitsspeicher pro Versuchs-VM von rund 19 GB. In Sachen Storage-Kapazität sollen je Versuchssystem 80 GB vorgehalten werden (für Betriebssystem, DB, DBMS und Tools). Das führt zu einem Storage-Bedarf von 400 GB für die Versuchssysteme.

4.3 Server-Hardware

Nachfolgend werden Eckpunkte der Server-Hardware genannt.

Für das Praktikum kommt ein Server-Blade zum Einsatz. Dieses Blade besteht aus zwei Prozessoren (CPUs) mit jeweils 6 Kernen und 3.46 GHz. Desweiteren besitzt der Server, wie aus der Kalkulation hervorgeht, 96 GB RAM. Der Gesamt-Storage, welcher in den Berechnungen bestimmt wurde, beträgt ca. 700 GB. Hierfür stehen Hardware-seitig drei 450 GB Fibre Channel Festplatten zur Verfügung. Die großzügige Reserve soll eventuellen späteren Anforderungen gerecht werden.

4.4 Werkzeuge der Performance-Messung

Die Performance-Messung hat verschiedene Aspekte. Es müssen hier Methoden und Werkzeuge für die Umsetzung voneinander abgegrenzt werden. So werden durch die Methoden einzelne Verfahrensansätze für die Tests beschrieben. Mit „Werkzeuge“ sind hier die Tools zur Messung der entsprechenden Werte zusammengefasst. Es gibt ein weites Spektrum an Tools, wobei immer die Aktualität und Verwendbarkeit der Werkzeuge geprüft werden sollte. Auf der einen Seite können Tools genutzt werden, welche in den jeweiligen Systemen, wie beispielsweise im Betriebssystem oder im DBMS, vorhanden sind oder es können zusätzliche Werkzeuge zum Einsatz kommen, wie z.B. Nagios [12]. Für die Wahl der Methoden und Werkzeuge bestehen mehrere Möglichkeiten, zum einen können feste Vorgaben durch den Lehrenden gemacht werden und zum anderen kann der Einsatz freigestellt werden, so dass in der Vorlesung ein Pool an Methoden und Werkzeugen vorgestellt wird und der Student entscheidet im Rahmen des Experiments, wie er vorgehen möchte. In der weiteren Arbeit muss geprüft werden, welche der beiden Methoden den Studenten am besten beim Lernprozess unterstützt.

5. Fazit und Ausblick

Performance ist wichtig. Das ist jedem, der mit Datenbanksystemen zu tun hat, klar. Den Studierenden ein Gefühl für diese Thematik zu geben, dass ist eine der Herausforderungen der Lehre. Um diese Herausforderung zu meistern, bieten sich Experimente an. Dadurch kann der Student praktische Erfahrungen im Bereich Datenbank-Performance erlangen. Damit den Studenten einheitliche Umgebungen zum Experimentieren zur Verfügung gestellt werden, bieten sich virtualisierte Systeme an. Dieser Rahmen ist bekannt und darüber sind sich die Autoren einig. In Zukunft muss gezeigt werden, wie die Aufgabenstellungen für die Studenten zu formulieren sind und wie die Performance konkret zu messen ist. Im Studiengang Wirtschaftsinformatik muss neben den technischen Details der Systeme auch immer die Wirtschaftlichkeit betrachtetet werden. Auch dieser Bereich muss in Zukunft genauer betrachtet werden.

6. REFERENCES

[1] Euler D.; Hahn A., Wirtschaftsdidaktik, 2004, 1. Auflage, Haupt Verlag, Bern

[2] Schubert S.; Schwill 1., Didaktik der Informatik, 2004, 1. Auflage, Spektrum Akademischer Verlag, München

[3] Reich K., Experiment, 2008, http://methodenpool.uni-koeln.de/download/experiment.pdf Abruf: 28.03.2012

[4] Faustmann; Höding; Klein; Zimmermann, Oracle- Datenbankadministration für SAP, 2007, 1. Auflage, Galileo Press, Bonn

[5] Balzert H.; Lehrbuch der Software- Technik, 1998 Spektrum Akademischer Verlag GmbH, Heidelberg; Berlin

[6] Schlimm N.; Novakovic M.; Spielmann R.; Knierim T., Performance-Analyse und -Optimierung in der Softwareentwicklung, 2007, Informatik Spektrum, Springer-Verlag

[7] Claus V.; Schwill A., Duden der Informatik, 2001, 3. Auflage, Duden Verlag, Mannheim

[8] Müller T., Leistungsmessung und -bewertung, 2004, Seminararbeit, Westfälische Wilhelms-Universität Münster, http://www.wi.uni-muenster.de/pi/lehre/ss04/SeminarTesten/Leistungsmessung.pdf Abruf: 28.03.2012

[9] Barrett D., Linux kurz und gut, 2004, 1. Auflage, O’Reilly Verlag, Köln

[10] Zimmer D.; Wöhrmann B.; Schäfer C.; Baumgart G.; Wischer S.; Kügow O., VMware vSphere 4 - Das umfassende Handbuch, 2010, 1. Auflage, Galileo Press, Bonn

[11] http://moodle.de/ Abruf: 18.05.2012

[12] Barth; Schneemann; Oestreicher, Nagios: System- und Netzwerk-Monitoring, 2012, 3. Auflage, Open Source Press

Migration nicht-nativer XML-Strukturen in den nativen XML-Datentyp am Beispiel von DB2 for z/OS V9.1

Christoph Koch

Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena
Christoph.Koch@uni-jena.de

DATEV eG
Datenbanken
Paumgartnerstr. 6-14
90429 Nürnberg
Christoph.Koch@datev.de

KURZFASSUNG

Mit der Einführung der pureXML-Technologie [4] bietet das von IBM entwickelte Datenbankmanagementsystem (DBMS) DB2 for z/OS die Möglichkeit, XML-Daten nativ zu speichern und XML-spezifisch zu verarbeiten. Die native Art der XML-Ablage verspricht gegenüber der nicht-nativen XML-Speicherung zahlreiche Vorzüge, die es sinnvoll werden lassen, im Unternehmen vorhandene nicht-native XML-Daten in die native Struktur zu überführen. Der vorliegende Beitrag setzt genau an dieser Migrationsthematik auf und zielt darauf ab, sowohl die einzelnen für eine solche Migration notwendigen Schritte aufzuzeigen, als auch deren Performance in Relation zum Gesamtaufwand der Überführung nicht-nativer XML-Dokumente in die native XML-Speicherform zu evaluieren.

Die Ergebnisse dieses Beitrags basieren auf der Grundlage der Arbeiten [1] und [2], die im Rahmen einer umfassenden Untersuchung der XML-Unterstützung von DB2 for z/OS V9.1 seit 2010 in der DATEV eG angefertigt wurden und richten sich daher insbesondere an den dortigen Anforderungen aus.

Kategorien und Themenbeschreibung

Database Performance and Benchmarks

Allgemeine Bestimmungen

Management, Measurement, Performance, Design, Languages

Schlüsselwörter

XML, nicht-nativ, nativ, Migration, Methodik, Performance

1. EINLEITUNG

Seit die Extensible Markup Language (XML) im Jahr 1998 vom World Wide Web Consortium (W3C) verabschiedet wurde, entwickelte sie sich fortlaufend weiter zum De-facto-Standard für den Austausch von (semi-)strukturierten Daten. Damit einher ging auch der verstärkte Einzug von XML in den Datenbanksektor [6], sodass sich sowohl dedizierte XML-DBMS herausbildeten, als auch herkömmliche relationale Systeme um XML-Strukturen erweitert wurden.

Die einfachste und älteste Form, XML-Daten in relationalen DBMSen zu speichern, ist die nicht-native XML-Speicherung, die die datenbankseitige Ablage von XML-Daten als Binär- oder Character-Strings vorsieht [7]. Demgegenüber existieren seit jüngerer Zeit auch native XML-Speichermöglichkeiten, wie beispielsweise die hier betrachtete in DB2 implementierte pureXML-Technologie und der damit verbundene native XML-Datentyp.

Da die native XML-Speicherung verglichen mit den verschiedenen nicht-nativen Ablageformen diverse Vorteile hinsichtlich Performance, Flexibilität und Kosteneffizienz verspricht [4], besteht auf langfristige Sicht das Ziel darin, „alte“ nicht-native Datenbestände in die „neue“ native Form der XML-Speicherung zu überführen. Die dazu notwendige Migration charakterisiert sich im Kontext von DB2 for z/OS durch den in den folgenden Abschnitten beschriebenen Ausgangs- und Zielzustand.

1.1 Ausgangszustand

Den Ausgangszustand der hier betrachteten Migrationsthematik bilden nicht-native XML-Dokumente, die im Fall der DATEV eG vorrangig in Form des VARCHAR- oder CLOB-Datentyps als Character-String und noch spezieller in bestimmten Kontexten über mehrere Tupel verteilt gespeichert sind. Solche Dokumente sind in unserem Kontext zum Beispiel von einer OCR-Software ins XML-Format digitalisierte Belege. Auf die genannten Besonderheiten der nicht-nativen Speicherformen wird nun konkreter eingegangen.

24th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012-01.06.2012, Lübbenau, Germany.

„Nicht-nativ“ bedeutet der Wortherkunft nach „nicht natürlich“. Bezogen auf die nicht-native oder auch XML-enabled genannte XML-Speicherung resultiert daher ein Unterschied zwischen Speicher- und Verarbeitungsformat, sodass auf diese Weise gespeicherte XML-Dokumente ein Mapping von dem vorliegenden (relationalen) Speichermodell hin zu dem XML-Datenmodell benötigen [5]. Beispiele für nicht-native XML-Speicherformen sind die Ablage von XML-Dokumenten als Binär- oder Character-String (siehe *Abbildung 1*) oder auch das Konzept der XML-Dekomposition. Letzteres basiert auf der Zerlegung von XML-Dokumenten in relationale Strukturen. DB2 stellt hierzu die XMLTABLE-Funktion [11] bereit.

Die weiteren Ausführungen dieses Beitrags konzentrieren sich aber wegen ihrer höheren Relevanz für die DATEV eG ausschließlich auf die nicht-native XML-Speicherform als Character-String und betrachten dabei speziell die Verwendung des CLOB-Datentyps. In diesem Fall wird für die betrachtete Migration als Mapping die zentrale Funktionalität des XML-Parsens benötigt. Dabei werden nicht-native XML-Daten in die native Speicherform transformiert. Nähere Details dazu folgen in Abschnitt 2.4.

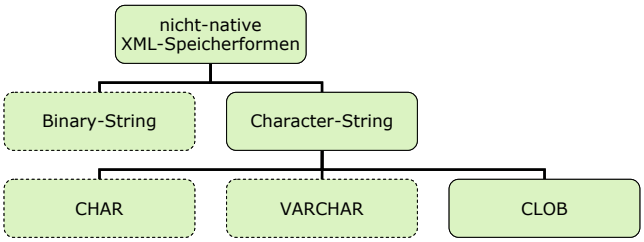


Abbildung 1: nicht-native XML-Speicherformen (Auswahl)

In der DATEV eG werden XML-Daten zum aktuellen Zeitpunkt nicht-nativ in der EBCDIC-Codepage gespeichert. Dabei handelt es sich um einen seit langem etablierten Single-Byte-Zeichensatz, der zur Repräsentation von Character-Daten genutzt wird und nur einen verglichen mit heutigen Standards wie Unicode geringen Zeichenvorrat abdeckt.

Für die hier betrachtete nicht-native XML-Speicherform des CLOB-Datentyps gelten diverse Einschränkungen hinsichtlich seiner Länge. Zwar kann ein CLOB-Dokument DB2-seitig prinzipiell eine maximale Länge von 2 GB besitzen. Diese lässt sich jedoch durch die DB2-Subsystem-Parameter LOBVALA [8] und LOBVALS [9] weiter einschränken, sodass per Default zur Ressourceneinsparung bei der (Haupt-)Speicheradressierung maximal 10 MB große LOB-Dokumente in DB2 verarbeitet werden können. In der DATEV eG wird von diesem Standardwert geringfügig nach oben abgewichen (aktuell 30 MB). Somit fällt die tatsächlich mögliche Maximallänge eines CLOB-Dokuments in unserem Kontext wesentlich kleiner als die genannten 2 GB aus.

Nicht zuletzt infolge dieser praktizierten Einschränkungen hat sich zur DB2-seitigen Speicherung großer XML-Dokumente in der DATEV eG das nachfolgend beschriebene Verfahren etabliert. Derartige Dokumente werden anwendungsseitig, wie in *Abbildung 2* beispielhaft dargestellt, nötigenfalls in kleine, in der Regel nicht wohlgeformte Teile „zerschnitten“ und diese über mehrere (CLOB-)Tupel hinweg datenbankseitig abgelegt. Eine ähnliche Diskussion wird in [1] und [2] auch für die nicht-native XML-Speicherung als VARCHAR geführt.

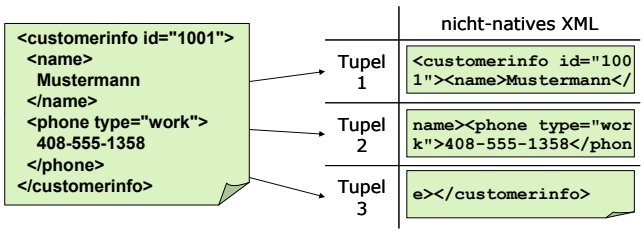


Abbildung 2: Verteilung über mehrere Tupel

1.2 Zielzustand

Entgegen dem zuvor beschriebenen Ausgangszustand erfordert der native Zielzustand der in diesem Beitrag betrachteten XML-Migration keine Variationen. Dies begründet sich durch die eine in DB2 implementierte, optimierte und sich implizit selbst verwaltende native XML-Speicherform, die auf speziell angepassten, aus der relationalen Welt bereits bekannten Strukturen basiert (siehe *Abbildung 3*). Hierzu zählen, ähnlich zur LOB-Speicherung, in XML-Tablespaces befindliche XML-Auxiliary-Tabellen, die über DocIDs und NodeIDs mit den Ausgangstabellen verknüpft sind. Die gesamten XML-Strukturen sind aber so gestaltet, dass sie aufgrund ihrer impliziten Verwaltung durch DB2 dem Nutzer gegenüber transparent bleiben und ihm somit nur als „normale“ Spalte vom Datentyp XML sichtbar werden. Dadurch hat er allerdings auf der anderen Seite unter anderem keinen auch Einfluss auf den verwendeten Zeichensatz, sodass der native XML-Datentyp in DB2 stets die Unicode-Codepage UTF-8 nutzt [10].

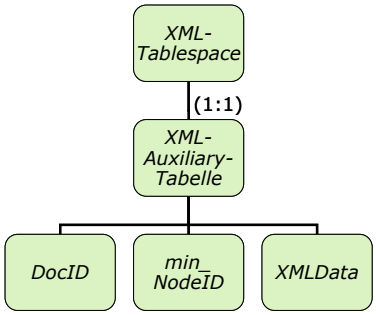


Abbildung 3: implizite XML-Verwaltungsobjekte (nach [10])

2. MIGRATIONSSCHRITTE

Ziel einer Datenmigration ist es, Informationen aus bestehenden Ausgangsstrukturen in spezielle Zielstrukturen zu transformieren. Die dazu notwendigen Teilschritte sind daher die direkte Konsequenz der Differenzen beider Strukturen. Übertragen auf die hier betrachtete Migration leitet sich auf diese Weise die in *Abbildung 4* dargestellte Folge von Migrationsschritten ab, auf die in den kommenden Abschnitten näher eingegangen wird.

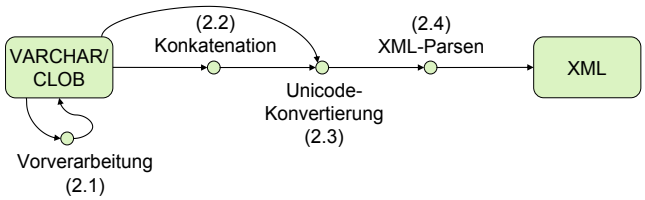


Abbildung 4: Ablauf der Migration

2.1 Vorverarbeitung

Der Schritt der Vorverarbeitung nimmt eine besondere Rolle in der Abfolge der Migration ein, da seine Bedeutung zum einen nicht direkt aus der geschilderten Zustandsdifferenz deutlich wird. Zum anderen lässt sich eine Vorverarbeitung aber auch von der eigentlichen Migrationsfolge (siehe *Abbildung 4*) losgelöst durchführen. Dies erklärt sich durch den funktionalen Zweck des dabei stattfindenden Vorgangs, der im Wesentlichen darin besteht, die Ausgangsdaten zu bereinigen.

Ein derartiger Bereinigungsvorgang ist in unserem Szenario nötig, da aufgrund des beschränkten Zeichenvorrats der EBCDIC-Codepage bestimmte Zeichen nicht dargestellt und infolgedessen als Substitution-Character (HEX: 3F) substituiert werden. Dieser Substitution-Character ist jedoch ein ungültiges Zeichen und wird vom XML-Parser in DB2 nicht akzeptiert. Damit nicht-native XML-Dokumente mit enthaltenen ungültigen Zeichen aber dennoch migriert werden können, ist es notwendig, durch eine Vorverarbeitung diese ungültigen Zeichen durch gültige zu ersetzen. Ein solcher Ersetzungsvorgang kann bereits auf dem Ausgangsdatenbestand durchgeführt werden und ist somit für den direkten Migrationsablauf unkritisch. Eine genauere Untersuchung der Performance dieses Vorverarbeitungsschritts erfolgt daher in Abschnitt 3.2 nicht.

2.2 Konkatenation

Während beliebig große XML-Dokumente nicht-nativ durch das in *Abbildung 2* dargestellte Aufteilen DB2-seitig abgelegt werden, gilt diese Einschränkung für die XML-Speicherung unter Verwendung des nativen XML-Datentyps nicht. Zwar werden native XML-Dokumente intern auch auf verschiedene Tupel verteilt gespeichert, dies aber erfolgt implizit knotenabhängig und gestaltet sich dem Nutzer gegenüber somit vollständig transparent [1]. Für die betrachtete Migration resultiert daher folgende Konsequenz: „zerteilte“ nicht-native XML-Dokumente müssen zur Überführung in den nativen XML-Datentyp wieder konkateniert werden.

Um diesen Vorgang der Konkatenation möglichst performant zu gestalten, sind in [1] und [2] die datenbankseitigen Vorgehensweisen Konkatenation per Stored Procedure und Konkatenation per Recursive SQL untersucht worden. Dabei hat sich erstere klar als die effizientere Variante herausgestellt, wobei sich bereits mit Blick auf Abschnitt 3.2 vorwegnehmen lässt, dass auch diese Form der Konkatenation einen erheblichen Anteil am Gesamtaufwand der Migration einnimmt. Wesentlich effizienter als die hier betrachteten Verfahren der expliziten Konkatenation wäre die direkte Unterstützung einer iterativen Eingabe von nicht-nativen Dokumentteilen durch den XML-Parser. Hierzu stellt DB2 stellt allerdings keine Mechanismen bereit. Insgesamt sei aber nochmals darauf hingewiesen, dass die in *Abbildung 2* gezeigte Zerteilung und damit auch die Konkatenation eher eine Art Sonderform darstellt, die nur in verhältnismäßig wenigen Fällen (sehr große XML-Dokumente) anzutreffen ist.

2.3 Unicode-Konvertierung

Zum aktuellen Zeitpunkt überwiegt in unserer Arbeitsumgebung in DB2 deutlich der Anteil von EBCDIC- gegenüber Unicode-Daten. Demzufolge sind auch nicht-native XML-Dokumente wie in Abschnitt 1.1 beschrieben in der EBCDIC-Codepage codiert abgelegt. Da aber für den nativen XML-Datentyp in DB2 strikt die Unicode-Codepage UTF-8 vorgegeben ist, müssen die ursprünglichen EBCDIC-Inhalte im Rahmen einer Migration in den Unicode-Zeichensatz überführt werden. Hierzu dient der Migrationsschritt der Unicode-Konvertierung.

2.4 XML-Parsen

Der wesentlichste Teil bei der Überführung von nicht-nativen XML-Dokumenten in den nativen XML-Datentyp besteht in dem Schritt des XML-Parsens. Dieser dient dazu, die textual (oder binär) repräsentierten Ausgangsdokumente auf ihre Gültigkeit zu überprüfen und in die hierarchische native XML-Struktur zu überführen. Nähere Details zu dieser speziellen (nativen) Speicherform sind bereits in Abschnitt 2.2 angedeutet worden, sollen hier aber nicht weiter vertieft werden. Eine ausführliche Darstellung dazu liefert [1].

Aus theoretischer Sicht lässt sich hinsichtlich der erwähnten Gültigkeitsprüfung nicht ausschließen, dass es unter bestimmten Bedingungen – beispielsweise bei dem Vorhandensein ungültiger Zeichen im Ausgangsdokument (siehe Abschnitt 2.1) – beim XML-Parsen zu Fehlersituationen kommt. Für den speziellen, hier betrachteten Kontext der Migration gilt dies allerdings nur bedingt. Zum einen werden aktuell sämtliche XML-Dokumente in der DATEV eG bereits vor ihrem Einfügen in DB2 anwendungsseitig geparst und zum anderen dient bereits der in Abschnitt 2.1 angesprochene Migrationsschritt dazu, verbleibende Fehler in den Ausgangsdaten zu bereinigen. Sollten dennoch Probleme während der Migration auftreten, sieht das in [1] diskutierte Konzept zur Migration auch für diesen Fall gezielte Fehlerbehandlungsmaßnahmen vor. Nähere Details dazu sollen hier nicht angefügt werden, sodass für die weitere Abhandlung die Korrektheit der Ausgangsdaten angenommen wird.

DB2 nutzt zum XML-Parsen die gleichnamige XMLPARSE-Funktion [11], die neben einer spezifizierbaren Option zur unterschiedlichen WHITESPACE-Handhabung (siehe *Abbildung 5*) auch zur XML-Schemavalidierung herangezogen werden kann. In der Nachfolgeversion 10 des hier betrachteten DB2 for z/OS V9.1 existieren zusätzlich auch andere Möglichkeiten, XML-Dokumente gegen ein XML-Schema zu validieren. Weitere Informationen dazu und zur XML-Schemavalidierung allgemein finden sich in [2]. Die hier diskutierte XMLPARSE-Funktion nutzt intern die z/OS-XML-System-Services [12] und ermöglicht auf diese Weise das effiziente XML-Parsen nicht-nativer XML-Dokumente.

Abbildung 5: WHITESPACE-Handhabung

3. PERFORMANCE

Die Ermittlung von Performance-Kennzahlen zu der in diesem Beitrag betrachteten Migration basiert auf einer Differenzbildung der Kosten verschiedener Folgen einzelner INSERT-Statements von CLOB-Daten bezogen auf das in Abschnitt 3.1 beschriebene Vergleichskriterium. Zur Interpretation der dabei gewonnenen Resultate ist es wichtig, einen groben Überblick über die aktuell in der DATEV eG eingesetzte DB2-Infrastruktur geben. Hierzu seien die folgenden Daten angeführt.

- zSeries = z196 (neueste Generation)
- Betriebssystemversion = z/OS 1.12 [IBM10b]
- Storage-System = Hitachi Data Systems (HDS) – Virtual Storage Platform (VSP)

Die anschließenden Ausführungen beschäftigen sich mit der Darstellung und Analyse der unterschiedlichen, teils unerwarteten Performance-Resultate. Diese finden sich in ausgeweitetem Umfang in [2] wieder und werden dort durch zusätzliche, für diesen Beitrag aber irrelevante Informationen zur Ergebnisdeutung, zum Hintergrund der Messdatenerhebung sowie zur Betrachtung weiterer Kriterien ergänzt.

Generell liegt bei der fortan beschriebenen Erhebung von Performance-Messdaten die Zielstellung darin begründet, den gesamten Migrationsablauf innerhalb von DB2 durchzuführen. Dies schränkt zwar die Möglichkeiten zur Überführung der XML-Daten auf die Verwendung von DB2-Bordmitteln ein – demgegenüber „mächtigere“ Anwendungsbibliotheken werden somit nicht verwendet –, bietet aber die größtmögliche Effizienz, da nur auf diese Weise zusätzlicher Kommunikationsaufwand der zu migrierenden XML-Daten zwischen Anwendung und DB2 eingespart werden kann.

3.1 Vergleichskriterium – Größe und Anzahl

Während [2] die Performance der Migrationsschritte hinsichtlich drei verschiedener Vergleichskriterien analysiert, soll für die Ausführungen dieses Beitrags die Betrachtung des Kriteriums der Größe und Anzahl genügen. Dieses zielt neben einer Gegenüberstellung der Kosten der einzelnen Migrationsschritte auch darauf ab, den Einfluss der Größe der zu migrierenden XML-Dokumente zu untersuchen. Dazu wird ein konstantes Gesamtdatenvolumen vorausgesetzt, das sich durch eine entsprechende Dokumentanzahl reguliert und für die in Abschnitt 3.2 angefügten Messresultate konstant 100 MB beträgt. Dieses Gesamtdatenvolumen repräsentiert in seinem Umfang natürlich nicht die realen Mengenverhältnisse der zu migrierenden Datenbestände der DATEV eG. Da sich aber bei den Analysen zu [2] eine relativ gute Skalierbarkeit bezogen auf die ermittelten Performance-Ergebnisse zeigte, genügt bereits eine derartige Datenmenge zur Grundlage für die Exploration der real zu erwartenden Migrationskosten. Die angesprochene Regulierung der Gesamtdatenmenge erfolgt für die abgebildeten Messresultate ausgehend von der Merkmalsausprägung „1 KB x 100.000“ – also der Migration von 100.000 je 1 KB großen XML-Dokumenten – bis hin zur Ausprägung „10 MB x 10“

3.2 Analyse der Migrationskosten

Gemäß der eingangs angedeuteten Vorgehensweise erfolgt die Ermittlung der Performance der einzelnen Migrationsschritte durch die sukzessive Aufschlüsselung der Gesamtkosten der Migration in einzelne (Teil-)Prozessketten. Die anschließenden Ausführungen betrachten daher die folgenden Messreihen.

- CLOB-INSERT (Basiskosten)
- CLOB-INSERT (inklusive Unicode-Konvertierung)
- XML-INSERT (inklusive Unicode-Konvertierung und XML-Parsen)
- XML-INSERT (inklusive Konkatenation, Unicode-Konvertierung und XML-Parsen)

Die Performance des Vorverarbeitungsschritts wird aus den in Abschnitt 2.1 genannten Gründen nicht weiter untersucht. Die anschließend präsentierten Resultate zeigen die bei den Messreihen ermittelten CPU-Zeiten. Neben diesen sind bei der Performance-Analyse in DB2 auch diverse Elapsed-Zeiten (inklusive I/O-Zeiten, Lock/Latch-Waits, etc.) von Bedeutung, die aber aufgrund ihrer starken Schwankungen in Abhängigkeit zur aktuellen Last auf dem DB2-Entwicklungssystem der DATEV eG in diesem Beitrag vernachlässigt werden. Eine Gesamtaufgliederung der relevanten Zeiten findet sich in [2]. In den weiteren Ausführungen werden für die genannten Messreihen die Resultate aufgezeigt und teilweise andiskutiert. Umfangreichere Informationen dazu finden sich ebenfalls in [2].

Zuerst sei der Fokus auf die Zusatzkosten gerichtet, den die Schritte **Unicode-Konvertierung** und **XML-Parsen** verursachen. Hier zeigt sich in *Abbildung 6* eine mit zunehmender Größe der zu migrierenden XML-Dokumente generell fallende CPU-Zeit. Während die Zusatzkosten für die Unicode-Konvertierung verhältnismäßig gering ausfallen, resultiert für den zusätzlichen Anteil, den das XML-Parsen verursacht, ein wesentlich höherer Aufwand. Dieser ist jedoch der Komplexität des XML-Parse-Vorgangs geschuldet, bei dem neben der Überführung der Ausgangsdaten in die native Speicherform unter anderem auch deren Struktur geprüft wird.

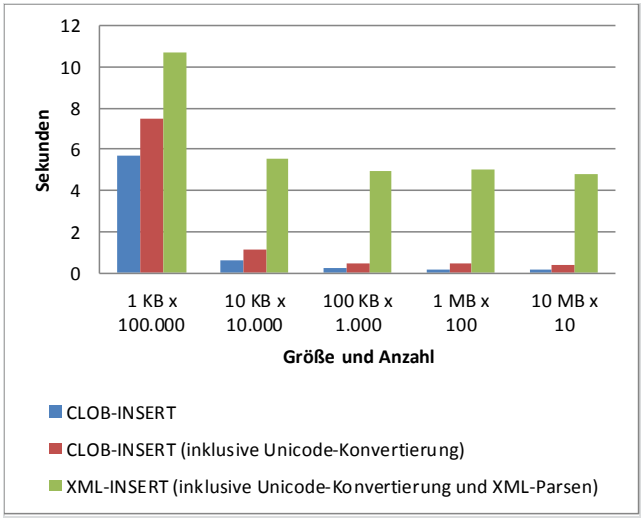


Abbildung 6: Performance der Migrationsschritte

Überraschenderweise sind die Grundkosten für einen einzelnen (komplexen) XML-Parse-Vorgang (unter anderem für das Laden und Initialisieren des XML-Parsers) jedoch kaum merklich, sodass sich die Messresultate ab der Merkmalsausprägung 10 KB x 10.000 trotz der kontinuierlich abnehmenden Anzahl von Dokumenten und somit auch XML-Parse-Vorgängen nur unwesentlich verändern. Mit anderen Worten formuliert bedeutet dies: Die relativ konstanten Gesamtkosten in *Abbildung 6* (jeweils circa 5 Sekunden CPU-Zeit für die "vier rechten hohen Balken") zeigen, dass der Kostenaufwand primär in den insgesamt zu parsenden Megabytes liegt, nicht aber in deren Aufteilung auf viele oder auf wenige XML-Dokumente.

Noch kostenintensiver als das XML-Parsen ist – eine gewisse Anzahl von Dokumentteilen vorausgesetzt – unerwarteterweise nur der Migrationsschritt der **Konkatenation**. Um dessen Einfluss geeignet zu visualisieren, sind die durch ihn bei der Datenerhebung zur Messreihe „XML-INSERT (inklusive Konkatenation, Unicode-Konvertierung und XML-Parsen)“ verursachten Kosten anteilig am Gesamtaufwand einer Migration in einer separaten *Abbildung 7* dargestellt.

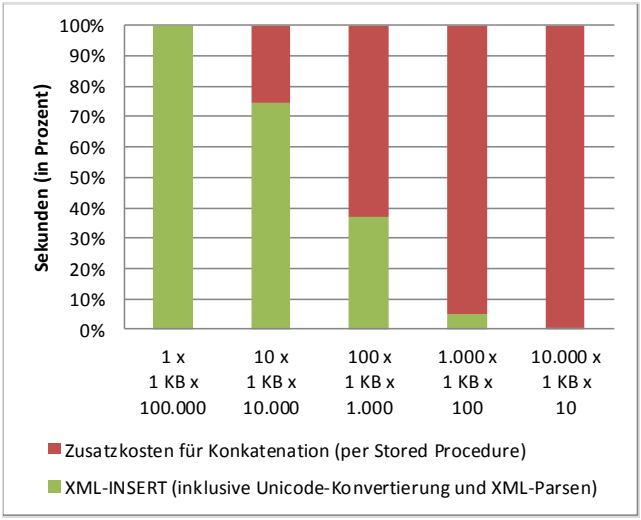


Abbildung 7: Kostenanteil der Konkatenation

Hier sei angenommen, dass stets 1 KB große Dokumentteile zu einem Gesamtdokument konkateniert und anschließend weitermigriert werden. Daher variiert bei den zu den abgebildeten Resultaten gehörenden Messreihen nicht mehr die Dokumentgröße in Abhängigkeit eines konstanten Gesamtdatenvolumens, sondern die Anzahl an, zu einem Dokument zu konkatenierenden Teile. Beispielweise werden bei der Merkmalsausprägung „10 x 1 KB x 10.000“ genau 10 je 1 KB große Dokumentteile zu einem 10 KB großen XML-Dokument zusammengefügt und anschließend wie ein „herkömmliches“ 10 KB großes nicht-natives XML-Dokument weitermigriert.

Auffällig ist hier, dass trotz der simplen Logik der Konkatenation mittels der CONCAT-Funktion dennoch ein erheblicher Aufwand entsteht, der ab einer Anzahl von 100 zu konkatenierenden Dokumentteilen den Hauptanteil der Gesamtkosten einer Migration einnimmt. An dieser Stelle lässt sich aber zumindest für die betrachteten Datenbestände der DATEV eG Entwarnung geben, da Konkatenationen mit derartig vielen Dokumentteilen im Rahmen der untersuchten Migration nicht vorzunehmen sind.

4. ZUSAMMENFASSUNG

Der vorliegende Beitrag hat einen Einblick in den Themenschwerpunkt der Migration von nicht-nativen XML-Dokumenten in den nativen XML-Datentyp in DB2 gegeben. Dabei erfolgte die Herleitung und Erarbeitung von zwingend erforderlichen und bedingt notwendigen Migrationsschritten sowie die Analyse deren Performance bezogen auf eine beispielhafte explorative Migration von CLOB-Daten mittels INSERT-Statements. Im Ergebnis dieser exemplarischen Auswertung zeigte sich, wie unterschiedlich die einzelnen Schritte die Gesamtkosten einer Migration beeinflussen und wo dabei die Hauptkostenfaktoren zu vermuten sind.

Neben dem fachlichen Aspekt soll dieser Beitrag aber ebenso dazu dienen, den wissenschaftlichen Charme eines praxisorientierten Anwendungsszenarios aufzuzeigen. Insbesondere wenn man wie in dem vorliegenden Beitrag angedeutet ins Detail einer konkreten Migrationslösung einsteigt, ergeben sich zahlreiche Parallelen zu anderen DBMS-Themengebieten wie Archivierung oder Aktive Datenbanken. Für den interessierten Leser seien daher zur vertieften Auseinandersetzung mit der hier diskutierten Thematik nochmals die Arbeiten [1] und [2] empfohlen. Insbesondere in [2] werden zudem auch weiterführende Untersuchungen zu den Themenschwerpunkten XML-Schema-Management (siehe auch [3]) und Query Performance angestellt.

5. LITERATUR

[1] C. Koch. *XML-Speicherstrukturen in DB2 for z/OS Version 9.1 und Überführungskonzepte für nicht-native XML-Speicherformen*, Studienarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 08.2011.

[2] C. Koch. *Der Datentyp XML in DB2 for z/OS aus Performance-Perspektive*, Diplomarbeit, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 01.2012.

[3] C. Koch. *Möglichkeiten und Konzepte zum XML-Schema-Management am Beispiel von DB2 for z/OS V9.1*, Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena und DATEV eG, 06.2012.

[4] IBM Corporation. *DB2 pureXML – Intelligent XML database management*, Produkt-Homepage, 2012. http://www-01.ibm.com/software/data/db2/xml

[5] M. Päßler. *Datenbanken und XML – Passt das?*, 04.07.2007. http://inka.htw-berlin.de/datenverwaltung/docs/Paessler.pdf

[6] G. Lausen. *Datenbanken: Grundlagen und XML-Technologien*, Spektrum Akademischer Verlag, 1. Auflage, 2005.

[7] H. Schöning. *XML und Datenbanken: Konzepte und Systeme*, Carl Hanser Verlag GmbH & Co. KG, 1. Auflage, 2002.

[8] IBM Corporation. *DB2 Version 9.1 for z/OS – USER LOB VALUE STG field (LOBVALA subsystem parameter)*, 03.2011. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z9.doc.inst/src/tpc/db2z_ipf_lobvala.htm

[9] IBM Corporation. *DB2 Version 9.1 for z/OS – SYSTEM LOB VAL STG field (LOBVALS subsystem parameter)*, 03.2011. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z9.doc.inst/src/tpc/db2z_ipf_lobvals.htm

[10] IBM Corporation. *DB2 Version 9.1 for z/OS – XML Guide*, 12.2010. http://publibfp.dhe.ibm.com/epubs/pdf/dsnxgk16.pdf

[11] IBM Corporation. *DB2 Version 9.1 for z/OS – SQL Reference*, 03.2011. http://publib.boulder.ibm.com/epubs/pdf/dsnsqk1a.pdf

[12] IBM Corporation. z/OS – XML System Services User's Guide and Reference, 01.2012. http://www-03.ibm.com/systems/resources/gxlza151.pdf

Evolution von XML-Schemata auf konzeptioneller Ebene

Übersicht: Der CodeX-Ansatz zur Lösung des Gültigkeitsproblems

Thomas Nösinger, Meike Klettke, Andreas Heuer
Lehrstuhl Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
(tn, meike, ah)@informatik.uni-rostock.de

ABSTRACT

If new requirements arise models have to be adapted to them. Additionally, the instances based on these models have to be adapted as well otherwise they might not be valid anymore. The same applies to the *XML schema* as a model and widely accepted description for XML documents. One solution for solving the occurring **validity problem** is the here described **XML schema evolution**. On the basis of a conceptual model the user interaction is analyzed and translated into transformation steps for the adaption of XML instances. The user interaction contains the use of predefined change operations on the conceptual model, a representation of the corresponding *XML schema*

Categories and Subject Descriptors

I.7.1 [**Document and Text Editing**]; H.3.2 [**Information Storage**]; H.3.3 [**Information Search and Retrieval**]; D.2.8 [**Metrics**]

Keywords

XML-Schemaevolution, konzeptionelle Modellierung, Gültigkeitsproblem, Instanzanpassung

1. EINLEITUNG

Die dynamische Anpassung von Modellen an aktuelle Gegebenheiten birgt die Notwendigkeit, vorhandene Instanzen und Anwendungen an die neuen, eventuell veränderten Umstände anzupassen. Ändert sich somit die strukturelle Beschreibung einer Datenbasis, muss zwangsläufig auch die Datenbasis angepasst werden, damit die Gültigkeit bezüglich des neuen Modells gewährleistet wird. Diese Problematik ist Kernpunkt des **Gültigkeitsproblems**, d.h. sind Instanzen eines Modells nach dessen Änderung noch gültig?

Die Veränderung der Struktur eines XML-Schemas bspw. durch das Umsortieren von Inhaltsmodellen, das Erhöhen des minimalen Vorkommens eines Elementes, das Einfügen oder auch Umbenennen nicht-optionaler Elemente etc. kann die Gültigkeit bereits vorhandener XML-Dokumente verletzen und macht eine Adaption dieser notwendig (siehe Abbildung 1).

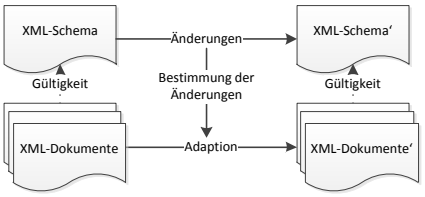


Figure 1: Das Gültigkeitsproblem nach [14]

Der Vorgang der XML-Schemaänderung und der darauf folgenden Anpassung der XML-Dokumente wird als **XML-Schemaevolution** bezeichnet. Es existieren unterschiedliche Ansätze zur Durchführung der XML-Schemaevolution [13]:

1. Die Anwendung einer expliziten Evolutionssprache.
2. Der Vergleich zweier Versionen eines Modells und die Ableitung von Transformationsschritten.
3. Die Beobachtung und Auswertung der Nutzerinteraktion, aus der dann Transformationsschritte abgeleitet werden. Dieser Ansatz wird hier thematisiert.

Der erste Ansatz ist die Anwendung einer Evolutionssprache zur XML-Schemaevolution. Dies entspricht im Datenbankbereich dem Alter-Befehl der Data Definition Language SQL. Zur Zeit ist dieser Ansatz aufgrund der fehlenden, standardisierten Evolutionssprache für XML-Schemata nicht anwendbar. Des Weiteren wird vom Anwender an dieser Stelle ein tiefes Verständnis bzw. Expertenwissen der möglichen Sprachkonstrukte und deren Anwendung verlangt.

Der zweite Ansatz ist die Versionierung von Modellen. Sind verschiedene Modelle vorhanden, dann können Transformationsschritte aus dem Vergleich beider Modelle erzeugt werden. Es müssen an dieser Stelle allerdings unterschiedliche, eventuell nicht vergleichbare Versionen vorgehalten werden. Des Weiteren wird normalerweise bei syntaktischen und /oder semantischen Konflikten eine automatische Lösung basierend auf Heuristiken vorgeschlagen oder aber der Anwender zur Lösung benötigt. Dieses Vorgehen erfordert wiederum ein tieferes Verständnis bzw. Expertenwissen der beteiligten XML-Schemaversionen.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Der dritte Ansatz nutzt die Interaktion eines Anwenders aus, indem von diesem getätigte Änderungen an einer konzeptionellen Repräsentation eines XML-Schemas analysiert werden. Die Nutzeränderungen sind dabei die Anwendung vordefinierter Operationen auf einem konzeptionellen Modell. Das somit erlangte Wissen wird für eine automatische Erzeugung von Transformationsschritten zur Anpassung der XML-Dokumente und Anwendungen verwendet. Es wird im Vergleich zu den ersten beiden Ansätzen weder vom Nutzer Expertenwissen bezüglich einer Evolutionssprache benötigt, noch müssen unterschiedliche Versionen eines XML-Schemas vorgehalten, ausgewählt und verwendet werden.

2. STAND DER FORSCHUNG

Die XML-Schemaevolution wird sowohl von den großen Datenbank- und Softwareherstellern (u.a. Microsoft [4], IBM [3], Oracle [5], Altova [2]) als auch in Forschungsprototypen (u.a. XCase [15], PRISM++ [8], X-Evolution [11], EXup [7] und GEA [9]) thematisch behandelt und auch teilweise umgesetzt.

Microsoft unterstützt den XML-Datentyp, lässt den Nutzer in Collections XML-Schemata sammeln und danach mittels einer Typisierung die Spalten einer Relation einem Schema zuordnen. XML-Instanzen typisierter Spalten können bezüglich ihres XML-Schemas auf Gültigkeit geprüft werden, ändert sich allerdings ein XML-Schema, muss dieses unter einer neuen Version erneut eingefügt werden. Microsoft unterstützt die hier angestrebte XML-Schemaevolution nicht, sondern nutzt die Versionierung von XML-Schemata. IBM DB2 ermöglicht die Registrierung von XML-Schemata in einem XML-Schema-Repository (XSR) und eine anschließende Erweiterung dieser unter Beachtung von zehn strengen Kompatibilitätsanforderungen. In Oracle können Schemata ebenfalls registriert und anschließend mittels der Methoden copyEvolve und inPlaceEvolve evolutioniert werden. Sowohl bei IBM als auch Oracle sind restriktive Anforderungen gestellt, die nur eine Generalisierung des alten Schemas ermöglichen und somit nur teilweise die angestrebte XML-Schemaevolution unterstützen.

Altova bietet mit Diffdog eine Erweiterung ihres Editors an, mit dem zwei Schemata miteinander verglichen werden können, um nachfolgend XSLT-Skripte zur Transformation der Instanzen zu erzeugen. Treten bei dem Vergleich Konflikte auf, d.h. es kann keine eindeutige Zuordnung zwischen den unterschiedlichen Strukturen der XML-Schemata hergeleitet werden, dann wird vom Anwender eine manuelle Zuordnung verlangt. Eine Automatisierung ist nicht möglich, eine Nutzerinteraktion und somit Expertenwissen bezüglich der XML-Schemata ist erforderlich.

Die Prototypen X-Evolution [11] und EXup [7] nutzen eine grafische Oberfläche zur Spezifikation, Ausführung und Verifikation von Schemaänderungen (beschrieben durch Primitive), wobei die Updatesprache XSchemaUpdate für die Beschreibung der Änderungen und die Dokumentadaption verwendet wird. Eine Grundlage dieser Systeme ist ein DBMS, welches XMLTYPE unterstützt. EXup und X-Evolution verwenden darüber hinaus XSUpdate als Evolutionssprache, welches XSPath nutzt (eine Teilmenge von XPath).

PRISM++ [8] bietet einen fein-granularen Evolutionsmechanismus zur Evolution von Datenbankschemata, welcher Datenbankadministratoren unter Verwendung von SMO-ICMO (Evolutionssprache: Schema Modification Operators - Integrity Constraints Modification Operators) die Evolution erleichtern soll. PRISM++ ermöglicht keine XML-Schemaevolution, thematisiert allerdings die Notwendigkeit, auch Integritätsänderungen vollziehen zu können.

In [9] wird das GEA Framework (Generic Evolution Architecture) vorgestellt, in dem XML-Schemata als UML-Klassendiagramme mit Stereotypen beschrieben werden. Unter Verwendung von elementaren Transformationsregeln werden Änderungen in UML auf XML propagiert.

XCase [15] ist eine Implementierung von XSEM (konzeptionelles Modell: Xml SEmantics Modeling), welches unter Verwendung einer MDA (Model-Driven Architecture) die XML-Schemaevolution durchführt. Es existieren unterschiedliche Abstraktionslevel (u.a. PIM als Platform-Independent Model und PSM als Platform-Specific Model) die Änderungen auf abstrakterem Niveau ermöglichen und diese dann zwischen den verschiedenen Ebenen propagieren (ein XML-Schema ist ein PSM). Dieser Ansatz ist dem hier vorgestellten am ähnlichsten, unterscheidet sich aber grundlegend in der Herangehensweise der Erfassung von Änderungen, deren Auswertung, Kategorisierung und dem Umfang möglicher Änderungen bezüglich eines XML-Schemas.

Eine automatische Erzeugung von Transformationsschritten und die damit verbundene Anpassung von XML-Dokumenten sind in den hier vorgestellten Forschungsprototypen nicht möglich bzw. die umsetzbaren Änderungen sind nicht umfangreich genug. Des Weiteren wird bei keinem der Prototypen der hier vorgestellte Ansatz der XML-Schemaevolution verfolgt, es besteht somit weiterhin Forschungsbedarf in dieser Thematik.

3. FORMALE GRUNDLAGEN

Eine Grundlage der XML-Schemaevolution ist das an der Universität Rostock entwickelte, konzeptionelle EMX Modell (Entity Model for XML-Schema), das zur grafischen Modellierung von XML-Schemata im Forschungsprototypen CodeX (Conceptual Design and Evolution for XML-Schema [12]) implementiert wurde. Die folgende Formalisierung des konzeptionellen Modells ist notwendig, um die bereits erwähnten Änderungsoperationen auf dem EMX Modell definieren zu können.

3.1 Konzeptionelles Modell

Das konzeptionelle Modell (M_M) ist ein Tripel, das aus Knoten (N_M), gerichteten Kanten zwischen Knoten (E_M) und zusätzlichen Eigenschaften (F_M) besteht.

$$M_M = (N_M, E_M, F_M) \tag{1}$$

Die Knoten sind Elemente ($elements_M$), Attributgruppen ($attributegroups_M$), Inhaltsmodelle ($groups_M$), Typen (einfache ($simple\text{-}types_M$), komplexe ($complex\text{-}types_M$)), Module ($modules_M$, u.a. externe XML-Schemata), Integritätsbedingungen ($integrity\text{-}constraints_M$) oder Annotationen ($annotations_M$). Die Knotenarten werden näher charakterisiert (z.B. $elements_M = \{\text{element}_M\}$, $\text{element}_M = (\text{ID, name, namespace, geometry, minoccur, maxoccur})$), was auf Grund von Platzbeschränkungen an dieser Stelle und bei den folgenden Modellen nicht vertieft werden soll. Die gerichteten Kanten werden jeweils zwischen Knoten definiert, wobei die Richtung die Zugehörigkeit (Enthalten-sein Beziehung) umsetzt und gemäß der Möglichkeiten von XML-Schema festgelegt wurden.

Die zusätzlich zu definierenden Eigenschaften ermöglichen die Nutzer-spezifische Anpassung eines konzeptionellen Mo-

dells, insofern dies erwünscht ist. Dazu zählen u.a. CodeX-Standardwerte zur Konfiguration des Editors, Angaben zur Pragmatik, Kosten, Namen und zur Version eines EMX Modells. Die Pragmatik ($Pragmatik \in$ {kapazitätserhaltend, kapazitätserhöhend, kapazitätsreduzierend}) ist im Zusammenhang mit den Änderungsoperationen notwendig, um den Informationsverlust durch nicht beabsichtige Modellanpassungen zu verhindern. Die Kosten sollen die Steuerung von Änderungsoperationen ermöglichen, damit zu komplexe und somit eventuell zu teure Transformationen verhindert werden können. Kosten werden verwendet, um den Aufwand der Änderungen zu bestimmen und gegebenenfalls eine Versionierung vorzuschlagen.

3.2 XML-Schema und XML-Instanzen

XML-Schema (XSD) als strukturelle Beschreibung von XML-Instanzen ist formal durch das W3C definiert [6]. Ein XML-Schema (M_S) ist in unserem Kontext ein Tupel aus Knoten (N_S) und zusätzlichen Eigenschaften (F_S). Beziehungen zwischen den Knoten (z.B. gerichtete Kanten) sind implizit in den Knoten enthalten, die zusätzlichen Eigenschaften sind Informationen zum Namen und der Version.

$$M_S = (N_S, F_S) \tag{2}$$

Knoten sind laut abstraktem Datenmodell Definitionen ($type\text{-}definitions_S$), Deklarationen ($declarations_S$), Modellgruppen ($model\text{-}group\text{-}components_S$), Gruppendefinitionen ($group\text{-}definitions_S$), Annotationen ($annotations_S$) oder Bedingungen ($constraints_S$). Neben der abstrakten Definition von Knoten existiert die *Element-Informationseinheit*, die für jeden Knoten die Realisierung innerhalb eines XML-Schemas definiert, d.h. welche Inhalte und Attribute können verwendet werden oder nicht.

XML-Instanzen (M_D) sind analog zum XML-Schema Tupel aus Knoten (N_D) und zusätzlichen Eigenschaften (F_D), wobei die Beziehungen zwischen den Knoten wieder implizit in diesen enthalten sind.

$$M_D = (N_D, F_D) \tag{3}$$

Die Knoten einer XML-Instanz sind Dokumente ($documents_D$), Attribute ($attributes_D$), Prozessanweisungen ($processing\text{-}instructions_D$), Namensräume ($namespaces_D$), Texte ($texts_D$) oder Kommentare ($comments_D$) [1].

Zusätzlichen Eigenschaften sind der Name, die Version, Änderungseigenschaften und Signifikanz eines Dokumentes. Die Änderungseigenschaften ($changeable \in$ {true, false}) sollen bei XML-Instanzen den ungewollten Informationsverlust verhindern (z.B. Entfernen von Elementen). Die Signifikanz ($significance$) ist ein normierter Wert, um bei Auswertungen von Dokumentkollektionen zum Finden von Standardwerten Dokumente priorisieren bzw. gewichten zu können.

3.3 Operationen

Auf dem konzeptionellen Modell (EMX) werden vordefinierte Änderungsoperationen vom Nutzer durchgeführt. Die anwendbaren Operationen lassen sich gemäß der Kategorisierung von Änderungsoperationen herleiten und charakterisieren (siehe [10]). Charakteristika sind u.a. die *Kapazitätsänderung* (kapazitätserhaltend, kapazitätserhöhend oder kapazitätsreduzierend), der *XML-Instanzeinfluss* (Instanzanpassung bei EMX-Änderung notwendig, nicht notwendig oder abhängig vom Parameter), die herleitbaren bzw. abschätzbaren *Kosten* einer Operation und die *Abhängigkeiten* von anderen Operationen. Eine Operation wird auf den oben definierten Modellen ausgeführt.

$$M_x \xrightarrow{Operation} M'_x,\ x \in \{M, S, D\} \tag{4}$$

Operationen sind zum Beispiel: *changeElementType* (Umsortieren vom Inhaltsmodell), *changeElementMinOccur* (Erhöhen des minimalen Vorkommens eines Elementes), *addElementToComplexElement* (Einfügen nicht-optionaler Elemente) oder *renameElement* (Umbenennen eines Elementes).

Eine Übersicht über die Zusammenhänge zwischen den Modellen und den Operationen ist in Abbildung 2 dargestellt. Die *renameElement*-Operation soll exemplarisch auf den oben definierten Modellen betrachtet werden. Dies ist eine einfache Operation, kompliziertere Evolutionsoperationen werden analog ausgeführt.

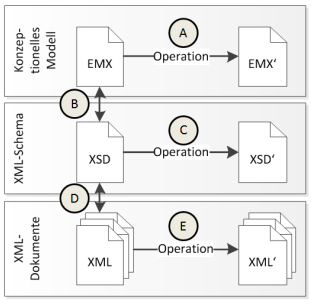


Figure 2: EMX, XML-Schema und XML-Instanzen

A: $renameElement_M$ (IDValue, oldValue, newValue)

$\forall\ n \in N_M \wedge n.ID = IDValue$:

if $n \in elements_M$

then $n.name := newValue$

Wenn ein Nutzer die grafische Repräsentation eines Elementes selektiert und das Attribut Name ändert, dann wird dieses in CodeX erkannt und als $renameElement_M$ Operation geloggt. Regeln garantieren dabei, dass nur sinnvolle Operationen geloggt werden (z.B. Voraussetzung für *renameElement*-Operation: $oldValue \neq newValue$). Die Identifikation des selektierten Elements wird im konzeptionellen Modell durch eine eindeutige ID gewährleistet. Neben der ausgeführten Operation, den beteiligten Werten und der ID müssen Informationen zum Kontext gespeichert werden. Dazu zählen u.a. Angaben zum *Gültigkeitsbereich*, zu vorhandenen *Referenzen* und zum direkten *Knotenumfeld* (Position in einem Inhaltsmodell etc.). Diese *Kontextinformationen* sollen bei den nachfolgenden Operationen ($renameElements_S$ und $renameElement_D$) die Identifizierung des betroffenen Elementes ermöglichen und die Verarbeitung erleichtern.

C: $renameElements_S$(oldValue, newValue, context)

$\forall\ n, m, k \in N_S \wedge n \neq m \wedge n \neq k \wedge m \neq k \wedge context(n)$:

// Globales Element mit neuem Namen existiert?

if $n, m \in elements_S \wedge n.scope.variety = 'global' \wedge m.scope.variety = 'global' \wedge n.name = oldValue \wedge m.name = newValue$

then $newValue := uniqueName(newValue) \wedge$
$n.name := newValue$
if $\exists\, k \in elements_S \wedge k.scope.variety = 'local' \wedge$
$k.ref = oldValue$
then $\forall\, k$: $k.ref := newValue$

// Lokales Element mit neuem Namen, aber anderem Typ existiert?

elseif $n,\, m \in elements_S \wedge n.scope.variety = 'local' \wedge$
$m.scope.variety = 'local' \wedge n.name = oldValue \wedge$
$m.name = newValue \wedge n.type \neq m.type \wedge$
$n.parent.name = m.parent.name$
then $newValue := uniqueName(newValue) \wedge$
$n.name := newValue$

// Lokales Element, kein Namen-Typ-Konflikt?

elseif $n \in elements_S \wedge n.scope.variety = 'local' \wedge$
$n.name = oldValue$
then $n.name := newValue$

// Globales Element, kein Namen-Konflikt?

elseif $n \in elements_S \wedge n.scope.variety = 'global' \wedge$
$n.name = oldValue$
then $n.name := newValue$
if $\exists\, k \in elements_S \wedge k.scope.variety = 'local' \wedge$
$k.ref = oldValue$
then $\forall\, k$: $k.ref := newValue$

Nachdem der Nutzer das konzeptionelle Modell verändert hat, muss das entsprechende XML-Schema ebenfalls angepasst werden. Dafür wird das Log von CodeX geladen und analysiert. Durch dieses Vorgehen sind die auszuführende Operation, die veränderten Werte und der entsprechende Knoten bekannt (*Kontextinformationen*). Durch die darauffolgende Umbenennung eines Elementes im XML-Schema, müssen je nach *Gültigkeitsbereich*, umgebenen *Knotenumfeld* und vorhandenen *Referenzen* weitere Anpassungen am XML-Schema vorgenommen werden. Zum Beispiel muss geprüft werden, ob eventuell gleich benannte, globale Elemente schon vorhanden sind oder ob Referenzen angepasst werden müssen. Die im konzeptionellen Modell gesammelten und im Log gespeicherten *Kontextinformationen* erleichtern dies.

E: $renameElement_D$(oldValue, newValue, context)

$\forall\, n \in N_D \wedge F_D.changeable$:

if $n \in elements_D \wedge n.node\text{-}name = oldValue \wedge$
$context(n)$
then $n.node\text{-}name := newValue$

Der letzte Schritt in der XML-Schemaevolution ist die automatische Erzeugung von Transformationsschritten aus den Änderungen des Nutzers am konzeptionellen Modell. Nachdem das XML-Schema angepasst wurde, sind vorhandene XML-Instanzen eventuell nicht mehr gültig (Gültigkeitsproblem). Dies kann aus den Charakteristika der Operationen, den gespeicherten Informationen des Logs und den Anpassungen des XML-Schemas hergeleitet werden. Sind Instanzen betroffen (der XML-Instanzeinfluss der Operation) und XML-Instanzen sollen verändert werden können ($F_D.changeable$), dann muss ein entsprechendes Transformationsskript erzeugt werden, das die Instanzen anpasst. In CodeX wird aktuell ein XSLT Skript (XML Stylesheet Language Transformation) erzeugt, welches auf XML-Dokumente angewendet werden kann.

3.4 Zusammenhänge

Die Beschreibung der Operationen macht deutlich, dass es Abhängigkeiten sowohl zwischen den Modellen als auch zwischen den Operationen gibt. Zum Beispiel sind die Kontextinformationen des konzeptionellen Modells für die Identifikation von Knoten im XML-Schema und/oder XML-Instanzen notwendig, zeitgleich lässt die Element-Informationseinheit des XML-Schemas u.a. Aussagen über notwendige oder nicht notwendige Anpassungen von XML-Instanzen zu (minOccurs, maxOccurs). Diese Zusammenhänge wurden in der Abbildung 2 dargestellt und führen zu folgenden Theoremen:

THEOREM 1. *Ein konzeptionelles Modell M_M wird durch die Operation $\boldsymbol{A}$ eines Nutzers verändert zu M'_M. Sei $\boldsymbol{B}$ eine Beschreibungsvorschrift zur Abbildung (Korrespondenz) des konzeptionellen Modells M_M auf ein XML-Schema M_S. $\boldsymbol{C}$ sei die Änderungsoperation zur Überführung von M_S zu M'_S. Dann gilt:*

$$\boldsymbol{A} \;+\; \boldsymbol{B} \Rightarrow \boldsymbol{C}$$

Das Theorem 1 besagt: Sind die Operation des Nutzers auf dem EMX Modell und die Korrespondenzen zwischen dem konzeptionellen Modell und dem XML-Schema bekannt, so kann die Operation zum Anpassen des XML-Schemas hergeleitet werden.

THEOREM 2. *Ein konzeptionelles Modell M_M wird durch die Operation $\boldsymbol{A}$ eines Nutzers verändert zu M'_M. Sei $\boldsymbol{B}$ eine Korrespondenz zwischen dem konzeptionellen Modell M_M und einem XML-Schema M_S und sei $\boldsymbol{D}$ eine Korrespondenz zwischen dem XML-Schema M_S und M_D. $\boldsymbol{E}$ sei die Änderungsoperation zur Überführung von XML-Instanzen M_D zu XML-Instanzen M'_D, die bezüglich M'_S gültig sind. Dann gilt:*

$$\boldsymbol{A} \;+\; \boldsymbol{B} \;+\; \boldsymbol{D} \Rightarrow \boldsymbol{E}$$

Das Theorem 2 besagt: Sind die Operation des Nutzers auf dem EMX Modell und die Korrespondenzen zwischen dem konzeptionellen Modell, dem XML-Schema und den XML-Instanzen bekannt, so kann die Operation zum Anpassen der XML-Instanzen hergeleitet werden.

Es können somit aus der Auswertung der Nutzerinteraktion mit dem konzeptionellen Modell automatisch entsprechende Transformationsschritte hergeleitet werden.

4. BEISPIEL

Die vorgestellte *renameElement*-Operation wird nachfolgend an einem durchgängigen Beispiel auf den unterschiedlichen Modellen angewendet.

Der Knoten mit dem Namen 'nummer' wird vom Nutzer ausgewählt und im konzeptionellen Modell verändert (siehe Abbildung 3). Es sind keine Einschränkungen bezüglich der Modellpragmatik oder der nicht zu überschreitenden Kosten gemacht worden (siehe F_M). CodeX erkennt, dass der ausgewählte Knoten eine spezifische ID (hier z.B. '1') hat und entsprechend ein Element ist. Wird nun das Attribut Name verändert, d.h. dieses Attribut bekommt den neuen Wert 'ID' ('nummer' $\neq$ 'ID'), dann wird dies in das Log geschrieben. Im Log steht, dass die Operation *renameElement*('1', 'nummer', 'ID') ausgeführt wurde. Des Weiteren werden anhand der eindeutigen Knoten-ID Kontextinformationen gespeichert, zum Beispiel die Gültigkeit ('lokal'), eventuelle Referenzen (ist in dem Fall aufgrund der Gültigkeit nicht

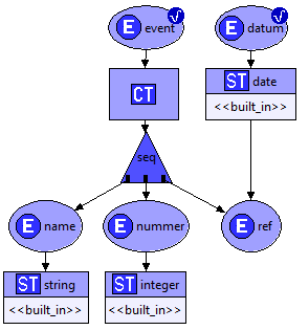


Figure 3: Konzeptionelles EMX Modell

möglich) und das Knotenumfeld (Vaterknoten ist gemäß der Kanteninformationen E_M 'event', Inhaltsmodell ist eine Sequenz, zweite Position in Sequenz).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:element name="event">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="nummer" type="xs:integer"/>
            <xs:element ref="datum"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="datum" type="xs:date"/>
</xs:schema>
```

Figure 4: XML-Schema zum EMX Modell

Abbildung 4 ist das zum obigen EMX Modell gehörende XML-Schema. Die Korrespondenzen zwischen dem konzeptionellem Modell und dem XML-Schema sind definiert, d.h. es ist aufgrund der Abbildungsvorschriften möglich, das eine Modell in das andere zu überführen. Durch das Log ist bekannt, dass die *renameElement*-Operation ausgeführt wurde, diese Operation muss demnach auch auf dem XML-Schema nachvollzogen werden. Es gibt dem Log folgend ein lokales Element ohne Referenzen im XML-Schema, welches den Namen 'nummer' hat, in einer Sequenz an zweiter Stelle steht und den Vater 'event' besitzt. Dieses Element soll umbenannt werden, das Attribut 'name' soll den neuen Wert 'ID' erhalten. In dem überschaubaren Beispiel ist dies nur ein Knoten, der den entsprechenden Kontext hat und mittels der *renameElement*-Operation umbenannt wird.

Aufgrund der Anwendung einer Operation auf dem konzeptionellen Modell (Operation und Kontextinformationen werden geloggt) und der bekannten Korrespondenzen, kann demnach die Operation zum Anpassen des XML-Schemas hergeleitet werden (siehe Theorem 1).

Durch die Umbenennung eines Elementes kann die Gültigkeit von XML-Instanzen betroffen sein. Es wurde die *renameElement*-Operation ausgeführt, welche laut Operationsspezifikation eine kapazitätserhaltende Operation ist. Des Weiteren kann die Operation je nach Parametern (d.h. die Kontextinformationen) einen Einfluss auf die Gültigkeit von XML-Instanzen haben. Es muss zunächst der entsprechende Elementknoten im XML-Schema durch die Kontextinformationen ermittelt werden. Es ist bekannt, dass das Element 'nummer' laut Korrespondenz zwischen XML-Schema und XML-Instanz ein zwingendes Element in einer Sequenz war (minOccurs = '1'), falls das globale Vaterelement 'event' als Dokumentenwurzel ausgewählt wurde. Das bedeutet, dass alle XML-Instanzen die gültig bezüglich des alten XML-Schemas waren und die verändert werden sollen ($F_D.changeable$), entsprechend an das neue XML-Schema angepasst werden müssen.

Durch die Anwendung einer Operation auf dem konzeptionellen Modell und der bekannten Korrespondenzen zwischen EMX Modell, XML-Schema und XML-Instanz, kann die Operation zum Anpassen der XML-Instanzen hergeleitet werden (siehe Theorem 2).

Diese Operation bzw. die notwendigen Transformationsschritte sind in einem XSLT-Skript realisiert, welches für das Beispiel in Abbildung 5 dargestellt ist.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/event">
      <event>
         <xsl:for-each select="name">
            <name><xsl:value-of select="."/></name>
         </xsl:for-each>
         <xsl:for-each select="nummer">
            <ID> <xsl:value-of select="."/></ID>
         </xsl:for-each>
         <xsl:for-each select="datum">
            <datum> <xsl:value-of select="."/></datum>
         </xsl:for-each>
      </event>
   </xsl:template>
</xsl:stylesheet>
```

Figure 5: XSLT-Skript zur XML-Instanzanpassung

5. UMSETZUNG

Der CodeX-Editor bietet unterschiedliche Funktionalitäten zur Durchführung und Unterstützung der XML-Schemaevolution (siehe Abbildung 6). Unter anderem kann das

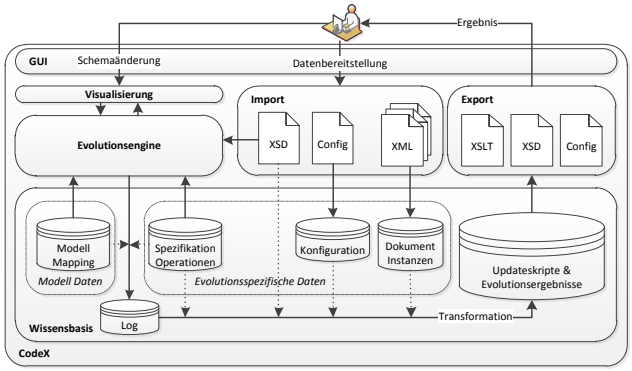


Figure 6: Komponentenmodell von CodeX

konzeptionelle Modell bzw. dessen grafische Repräsentation von einem Nutzer verändert werden, was entsprechend in einem Log gespeichert wird. Welche Änderungen bzw. vordefinierten Operationen anwendbar sind, wird durch Operationsspezifikationen beschrieben (siehe Kategorisierung in [10]). Das konzeptionelle Modell wird aus einem gegebenen XML-Schema unter Verwendung vom Modell-Mapping Informationen (Korrespondenzen) extrahiert oder neu angelegt. Das Log kann nach einer entsprechenden Analyse zur Erzeugung

von XSLT-Skripten verwendet werden, welche die aus den Nutzeraktionen hergeleiteten Transformationsschritte umsetzen. Diese Transformation nutzt die evolutionsspezifischen Daten, u.a. erneut die Operationsspezifikation, XML-Schemainformationen, gegebene Konfigurationen oder auch Dokumentkollektionen. Konfigurationen sind die zusätzlichen Eigenschaften (F_M) des konzeptionellen Modells, während die Dokumentinstanzen zur Kostenabschätzung und gegebenenfalls zur Wertfindung verwendet werden können. Die Datenbereitstellung und Extraktion von Ergebnissen werden entsprechend durch Import und Export-Komponenten realisiert.

6. ZUSAMMENFASSUNG

Die XML-Schemaevolution behandelt das Gültigkeitsproblem. Wenn sich XML-Schemata ändern und somit an neue Gegebenheiten bzw. Anforderungen angepasst werden, müssen deren Instanzen zur Wahrung der Gültigkeit ebenfalls adaptiert werden. Dies sollte weitestgehend automatisch erfolgen. Im präsentierten Ansatz wird ein konzeptionelles Modell verwendet, welches eine vereinfachte, grafische Repräsentation eines XML-Schemas ist. Wird das konzeptionelle Modell durch vordefinierte Operationen von einem Nutzer verändert, dann wird das damit gewonnene Wissen entsprechend geloggt und danach für die Anpassung des XML-Schemas und der XML-Instanzen angewendet. In diesem Zusammenhang sind Kontextinformationen und Korrespondenzen zwischen den Modellen entscheidend, denn nur durch diese Informationen können automatisiert Transformationsschritte aus der Nutzerinteraktion hergeleitet werden.

Die einzelnen Modelle wurden kurz vorgestellt, bevor die renameElement-Operation formal beschrieben wurden. Die Zusammenhänge der einzelnen Modelle und die Anwendung der renameElement-Operation wurde ebenfalls thematisiert. An einem durchgängigen Beispiel wurde darüber hinaus dargestellt, wie diese Operation auf den einzelnen Modellen angewendet wird. Des Weiteren wurde CodeX als Forschungsprototyp präsentiert, der zur Realisierung und Durchführung der XML-Schemaevolution unerlässlich ist.

Ein hier nicht ausführlich, aber dennoch in CodeX bereits vorhandener Aspekt ist die Berücksichtigung von Kostenfunktionen bei der XML-Schemaevolution. Aktuell werden die möglichen Kosten einer Evolution anhand der durchzuführenden Änderungsoperation abgeschätzt und mit einem Nutzer-spezifischen Grenzwert verglichen. CodeX kann so konfiguriert werden, dass bei Überschreitung dieses Wertes Änderungsoperationen abgelehnt werden und stattdessen eine Versionierung vorgeschlagen wird.

7. AUSBLICK

CodeX bietet derzeit Basisfunktionalitäten zur Modellierung von XML-Schema und zur XML-Schemaevolution an. Momentan fehlen allerdings die Behandlung von Integritätsbedingungen, Namensräume, die Beachtung von Dokumentkollektionen und die Auswertung und Speicherung der Kontextinformationen. In Zuge der Anpassung und Erweiterung soll CodeX als Webapplikation freigeschaltet werden, damit der Prototyp von anderen Anwendern zur XML-Schemaevolution verwendet werden kann.

Des Weiteren werden die bisher vorhandenen Beschreibungen der Änderungsoperationen gemäß der Kategorisierung von [10] erweitert, formal beschrieben und integriert.

8. REFERENCES

[1] 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 01-March-2012.

[2] Altova Produkt diffdog: XML-Schemavergleich. `http://www.altova.com/de/diffdog/xml-schema-diff-tool.html`, 2011. Online; accessed 15-December-2011.

[3] IBM DB2 LUW V9R7 Online Documentation. `http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp`, 2011. Online; accessed 15-December-2011.

[4] Microsoft technet: Verwenden von xml in sql server. `http://technet.microsoft.com/de-de/library/ms190936(SQL.90).aspx`, 2011. Online; accessed 15-December-2011.

[5] Oracle XML DB Developer's Guide 11g Release 2 (11.2). `http://docs.oracle.com/cd/E11882_01/appdev.112/e23094/xdb07evo.htm`, 2011. Online; accessed 12-December-2011.

[6] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. `http://www.w3.org/TR/2012/PR-xmlschema11-1-20120119/`, January 2012. Online; accessed 01-March-2012.

[7] F. Cavalieri. EXup: an engine for the evolution of XML schemas and associated documents. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 21:1–21:10, New York, NY, USA, 2010. ACM.

[8] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo. Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++. *PVLDB*, 4(2):117–128, 2010.

[9] E. Domínguez, J. Lloret, B. Pérez, Á. Rodríguez, A. L. Rubio, and M. A. Zapata. Evolution of XML schemas and documents from stereotyped UML class models: A traceable approach. *Information & Software Technology*, 53(1):34–50, 2011.

[10] H. Grunert. XML-Schema Evolution: Kategorisierung und Bewertung. Bachelor Thesis, Universität Rostock, 2011.

[11] G. Guerrini and M. Mesiti. X-Evolution: A Comprehensive Approach for XML Schema Evolution. In *DEXA Workshops*, pages 251–255, 2008.

[12] M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63, 2007.

[13] M. Klettke. *Modellierung, Bewertung und Evolution von XML-Dokumentkollektionen.* Habilitation, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2007.

[14] M. Klettke, H. Meyer, and B. Hänsel. Evolution — The Other Side of the XML Update Coin. In *2nd International Workshop on XML Schema and Data Management (XSDM), Tokyo*, April 2005.

[15] J. Klímek, L. Kopenec, P. Loupal, and J. Malý. XCase - A Tool for Conceptual XML Data Modeling. In *ADBIS (Workshops)*, pages 96–103, 2009.

Ein Partitionierungsdienst für Geographische Daten in Räumlichen Datenbanken

Hendrik Warneke und Udo W. Lipeck
Institut für Praktische Informatik, FG Datenbanken und Informationssysteme
Leibniz Universität Hannover, Welfengarten 1, 30159 Hannover
{hwa,ul}@dbs.uni-hannover.de

ZUSAMMENFASSUNG

Da in der computergestützten Geographie mit zum Teil sehr großen Mengen von räumlichen Daten gearbeitet werden muss, hat sich mittlerweile die Überzeugung durchgesetzt, solche Datensätze in räumlichen Datenbanken zu speichern. Allerdings wird bei der Entwicklung von geographischen Programmen dem Aspekt der Skalierbarkeit oftmals wenig Beachtung geschenkt. So enthalten verbreitete Programme für Geoinformationssysteme wenig bis keine sogenannten externen Algorithmen, die den Geschwindigkeitsunterschied zwischen internem (RAM) und externem Speicher (Festplatte) berücksichtigen und versuchen, die Anzahl der I/O-Zugriffe auf letzteren zu minimieren. Diese Programme arbeiten daher nur dann effizient, wenn genug interner Speicher für die zu bearbeitenden Geodaten zur Verfügung steht. Wir stellen in diesem Beitrag einen auf Partitionierung basierenden Ansatz vor, der den Aufwand für die Entwicklung von auf großen Geodatenmengen skalierenden Programmen stark reduziert. Dazu werden Zugriffe auf externe Speicher in eine vorgelagerte Partitionierungs- und eine nachgelagerte Rekompositionsphase verschoben und für diese Phasen flexible Operationen, die ein breites Spektrum an geographischen Problemstellungen unterstützen, als Partitionierungsdienst für räumliche Datenbanksysteme angeboten.

1. EINLEITUNG

Lange Zeit verwendete man für die Arbeit mit geographischen Informationen größtenteils auf Papier gedruckte Landkarten. Diese wurden in handliche Kartenblätter unterteilt, um die zur Beschreibung der Erdoberfläche in hoher Auflösung benötigten Datenmengen besser handhaben zu können. Der Übergang zu computergestützten Geoinformationssystemen erlaubte es, Geodaten mit Hilfe unabhängig vom Maßstab in Form von Geometrien zu modellieren. Dabei behielt man häufig die Aufteilung in Kartenblätter auf Dateiebene bei, da die verarbeitbare Datenmenge für viele Systeme und Datenformate beschränkt war. Heute baut man sogenannte Geodateninfrastrukturen auf, in denen Datenproduzenten, Dienstleister und Nutzer über Netzwerke, in der Regel das Internet, miteinander verknüpft sind und geographische Informationen mit standardisierten Verfahren austauschen und verarbeiten [4]. Diese Strukturen sehen vor, dass geographische Daten blattschnittfrei in *räumlichen Datenbanken* gespeichert werden, wozu man häufig objektrelationale Systeme mit Erweiterungen um räumliche Datentypen, Operatoren und Indexe einsetzt.

Aus den grundlegenden, mit den Techniken der Landesvermessung hergestellten Daten, den sogenannten *Geobasisdaten*, können durch vielfältige Prozesse neue Datensätze abgeleitet werden, die man für kartographische Darstellungen von Informationen aus Fachbereichen wie z.B. Verkehr, Umwelt und Ökonomie verwendet. Die folgende Auflistung enthält einige Beispiele für Klassen solcher Prozesse, die heute meistens automatisiert durch Computerprogramme, im folgenden *Geoprogramme* genannt, durchgeführt werden.

Generalisierung: Um eine lesbare Kartendarstellung aus den Geobasisdaten zu erzeugen, müssen diese durch Operationen wie z.B. das Weglassen von Objekten sowie Vereinfachen, Zusammenfassen oder Verdrängen von Geometrien verändert werden. Dabei besitzen die Basisdaten meist eine höhere Auflösung als für die beabsichtigte Darstellung benötigt wird.

Transformation: Erfordert eine Anwendung ein bestimmtes Datenmodell, das sich von dem der Geobasisdaten unterscheidet, müssen diese zunächst in das beabsichtigte Modell transformiert werden.

Integration: Um Informationen aus verschiedenen Datensätzen gemeinsam nutzen zu können, werden diese zu einem Datensatz integriert. Dies erfordert z.B. Verfahren wie die Identifikation korrespondierender geographischer Objekte (Matching) sowie die Anpassung von Geometrien für die gemeinsame Darstellung.

Geoprogramme, die diese Prozesse implementieren, müssen aufgrund des beträchtlichen Umfangs von Geobasisdatensätzen in der Lage sein, den begrenzten Hauptspeicher eines Rechners effizient zu verwalten, um Performanceprobleme aufgrund von Swapping oder Programmabstürze wegen Speichermangels zu vermeiden. Dazu würde es sich anbieten, diese Programme als Datenbankanwendungen zu implementieren, die sämtliche Datenzugriffe über SQL-Befehle abwickeln. Dieses Vorgehen ist jedoch mit einigen Nachteilen verbunden. Zunächst ist man auf die von dem räumlichen Datenbanksystem angebotene Funktionalität für geometrische und geographische Berechnungen beschränkt. Die

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

freie Auswahl aus spezialisierten und optimierten Software-Bibliotheken entfällt. Weiterhin hängt die Performance von SQL-basierten Anwendungen entscheidend von der Qualität der Anfrageoptimierung ab. Aufgrund der inhärenten Schwierigkeit der Kostenschätzung für geometrische Operationen stellt dies insbesondere bei räumlichen Datenbanken ein Problem dar. Schließlich müssten die Algorithmen für Geoprogramme in eine relationale Formulierung umgewandelt werden, was aufgrund von deren Komplexität i.A. aufwändig und fehleranfällig ist. Hinzu kommt noch, dass die Entwickler geographischer Anwendungen oftmals keine Datenbankspezialisten sind.

Wir stellen in diesem Beitrag einen Ansatz vor, der diese Probleme durch *Partitionierung* der Geobasisdaten umgeht. Statt ein Geoprogramm auf den gesamten Datensatz anzuwenden, wird jede Datenpartition für sich allein bearbeitet, wobei diese so klein gewählt wird, dass der verfügbare Hauptspeicher nicht überfüllt wird. Die für die einzelnen Partitionen berechneten Ergebnisdatensätze müssen anschließend wieder zu einem Gesamtdatensatz zusammengesetzt werden, was wir als *Rekomposition* bezeichnen. Da unterschiedliche Prozesse auch unterschiedliche Strategien für die Partitionierung und Rekomposition erfordern, bieten wir eine Sammlung solcher Operationen als Dienst auf einem räumlichen Datenbanksystem an. Die Entwickler von Geoprogrammen können diesen über eine einfache Schnittstelle ansprechen, wodurch die Notwendigkeit entfällt, innerhalb der Programme selbst den Austausch der Daten zwischen Haupt- und sekundärem Speicher zu berücksichtigen.

Für geographische Daten bietet es sich an, zur Partitionierung die räumliche Lage zu verwenden und Datenobjekte aus einem zusammenhängenden Gebiet in eine Partition einzuteilen. Dieser Ansatz ist allgemein für Geoprogramme anwendbar, die sich *lokal* verhalten. Dies bedeutet, dass für die Berechnung von Ergebnissen an einer Position nur Daten aus einer begrenzten Umgebung benötigt werden und diese Umgebung klein gegenüber einer Partition ist. In diesem Fall lassen sich Fehler, die insbesondere am Rand einer Partition entstehen, weil dem Geoprogramm nicht alle Daten zur Verfügung stehen, reduzieren oder ganz vermeiden, indem man sich überlappende Partitionen erzeugt [6].

Der Rest des Beitrags ist wie folgt strukturiert: In Abschnitt 2 nennen wir Arbeiten, die mit unserem Ansatz verwandt sind. Abschnitt 3 beschreibt die Architektur und das Funktionsprinzip des Partitionierungsdienstes. In Abschnitt 4 verwenden wir die Liniensegmentierung als konkretes Anwendungsbeispiel und geben für diesen Prozess alternative Partitionierungs- und Rekompositionsoperationen an. Abschnitt 5 beschreibt die Ergebnisse von Experimenten mit diesen Operationen auf realen geographischen Daten. Schließlich liefert Abschnitt 6 ein Fazit über den in diesem Beitrag vorgestellten Ansatz und einen Ausblick auf Folgearbeiten.

2. VERWANDTE ARBEITEN

Das Prinzip, Datensätze zuerst aufzuteilen, Teilergebnisse zu berechnen und später aus den Teilergebnissen ein Gesamtergebnis zu generieren, ist in der Informatik unter dem Namen Divide-and-Conquer bekannt. Auch für geometrische Probleme gibt es eine Reihe von Vorschlägen, beispielsweise von Güting und Schilling [3], die externe Algorithmen nach diesem Prinzip entwickeln. Um eine optimale asymptotische Laufzeit zu erreichen, werden die drei Teilschritte stark aufeinander abgestimmt, indem z.B. Sortierungen oder spezielle Repräsentationen der Daten weitergegeben werden. Für unseren Partitionierungsdienst sind wir hingegen an flexiblen Partitionierungs- bzw. Rekompositionsoperationen interessiert, die für die Entwicklung der Geoprogramme keine einschränkenden Vorgaben machen.

Aufgrund seiner Wichtigkeit bei der Berechnung räumlicher Verbunde ist das Problem der Bestimmung von sich überschneidenden achsenparallelen Rechtecken besonders intensiv untersucht worden. Während für interne Algorithmen das Plane-Sweep-Paradigma [1] favorisiert wird, verwenden externe oder parallele Algorithmen häufig Partitionierung, wie z.B. Patel und DeWitt [5]. Der Einfluss von Redundanz auf die Laufzeit von partitionierungsbasierten Algorithmen für dieses Problem wird beispielsweise von Zhou et.al. [8] untersucht. In einer ähnlichen Untersuchung [2] kommen Dittrich und Seeger u.a. zu dem auf den ersten Blick überraschenden Ergebnis, dass man durch mehr Redundanz in den Berechnungen sogar die Laufzeit verbessern kann.

3. ENTWURF DES PARTITIONIERUNGSDIENSTES

3.1 Architektur

Wir implementieren Partitionierungs- und Rekompositionsoperationen als Stored Procedures auf einem räumlichen Datenbanksystem. Diese können über eine Datenbankschnittstelle von einem *Steuerprogramm* außerhalb der Datenbank aufgerufen werden, um in der Datenbank gespeicherte Daten für ein Geoprogramm aufzuteilen und wieder zusammenzusetzen (Abbildung 1). Das Steuerprogramm liest jeweils

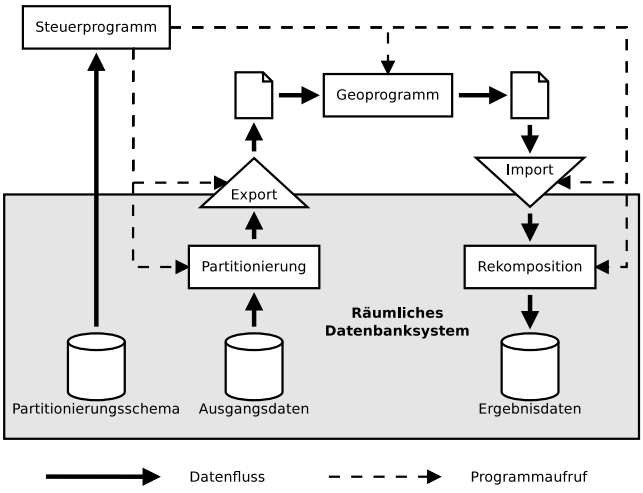


Abb. 1: Geoprogramm (Ablauf) mit Partitionierung

einen Eintrag aus dem Partitionierungsschema (Abschnitt 3.2), das die Geometrien der Partitionen enthält. Damit wird eine geeignete Partitionierungsoperation aufgerufen und eine Partition der Ausgangsdaten berechnet. Diese muss in ein von dem Geoprogramm verwendetes Dateiformat exportiert werden, bevor dieses vom Steuerprogramm aufgerufen wird, um für die Datenpartition ein Ergebnis zu berechnen, das zunächst ebenfalls im Dateisystem abgelegt wird. Nachdem dieses Ergebnis wieder in die Datenbank importiert worden ist, ruft das Steuerprogramm eine geeignete Rekompositionsoperation auf, die die Daten mit den bereits rekompo-

nierten Daten aus anderen Partitionen zusammensetzt und Konflikte auflöst. Anschließend kann das Steuerprogramm mit der nächsten Partition aus dem Partitionierungsschema fortfahren bis der komplette Datensatz bearbeitet ist.

Für Zugriffe auf die möglicherweise großen Datensätze innerhalb der Partitionierungs- und Rekompositionsoperationen verwenden wir konsequent SQL-Anweisungen. Dadurch machen wir implizit von den im räumlichen Datenbanksystem implementierten externen Algorithmen Gebrauch, so dass innerhalb dieser Operationen eine effiziente Abfolge von I/O-Zugriffen erfolgt. Da innerhalb des Geoprogramms nur noch auf die Daten aus einer Partition zugegriffen wird, sind dort keine externen Algorithmen mehr nötig.

3.2 Redundanz

Um vorzugeben, wie geographische Daten bei der Partitionierung aufgeteilt werden, verwenden wir ein sogenanntes *Partitionierungsschema*. Dieses besteht aus einer schnittfreien und lückenlosen Menge von Polygonen (Partitionspolygone), denen jeweils eine eindeutige Partitionsnummer zugeordnet ist. Es ist möglich, ein vordefiniertes Partitionierungsschema (z.B. eine Unterteilung in Verwaltungsbezirke), das in der Datenbank gespeichert ist, zu verwenden, oder den Partitionierungsdienst ein geeignetes Schema (z.B. ein Gitter aus gleich großen Rechtecken) erzeugen zu lassen.

Berechnungen in Geoprogrammen weisen oft eine mehr oder weniger starke Abhängigkeit vom räumlichen Kontext der Datenobjekte auf. Dies bedeutet, dass nur dann korrekte Ergebnisse in einem bestimmten Gebiet erzeugt werden können, wenn auch die Objekte aus einer lokalen Umgebung dieses Gebiets in der Partition enthalten sind. Bei einer streng disjunkten Aufteilung der geographischen Daten, wie durch das Partitionierungsschema vorgegeben, steht insbesondere für Objekte am Rand der Partition möglicherweise nicht genug Kontext zur Verfügung, da Objekte von außerhalb des Partitionspolygons beim Aufruf des Geoprogramms nicht in den Daten enthalten sind. Um diesen Nachteil zu kompensieren, erlauben es die von uns entwickelten Operationen, dass dieselben Datenobjekte in mehreren Partitionen enthalten sind, was wir als *Redundanz* bezeichnen. Das Ziel dabei ist es, dass beim Aufruf des Geoprogramms für eine Partition genug Daten in dieser enthalten sind, um korrekte Ergebnisse zumindest für alle Positionen innerhalb des Partitionspolygons zu berechnen.

Die Erzeugung von Redundanz in den partitionierten Daten lässt sich auf verschiedene Arten erreichen. Eine Möglichkeit ist, die Partitionspolygone um einen festen Abstand zu vergrößern, so dass sich benachbarte Polygone am Rand überlappen. Diese Operation, die auch in Abbildung 2 dargestellt ist, bezeichnet man als Pufferbildung. In Abschnitt 4 geben wir weitere Möglichkeiten an, bei denen redundant zu repräsentierende Daten über die Beziehungen zwischen den Objekten bestimmt werden. Zu beachten ist dabei, dass sich in Abhängigkeit davon, wieviel Redundanz man für eine bestimmte Anwendung benötigt, das Datenvolumen für einzelne Partitionen vergrößert. Im schlimmsten Fall kann eine Partition dann mit dem zur Verfügung stehenden Hauptspeicher nicht mehr bearbeitet werden, wodurch das eigentliche Ziel der Partitionierung verfehlt wird. Häufig kann man für Geoprogramme nachweisen oder experimentell belegen, dass die Abhängigkeit vom Kontext auf Umgebungen mit kleiner räumlicher Ausdehnung beschränkt bleibt (siehe z.B. [6]). In diesen Fällen sprechen wir davon, dass diese Programme

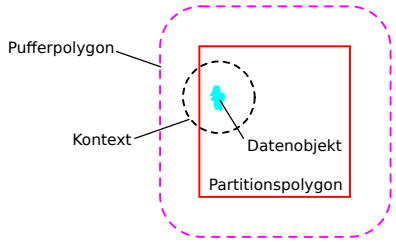


Abb. 2: Kontext, Partitions- und Pufferpolygon

lokale Berechnungen ausführen, und das in diesem Beitrag vorgestellte Konzept ist für solche Anwendungen einsetzbar.

Redundanz führt dazu, dass für einige Datenobjekte mehrere Ergebnisse in unterschiedlichen Partitionen berechnet werden. Wir bezeichnen solche Situationen als *Rekompositionskonflikte*. Bei der Rekomposition müssen diese aufgelöst und die Ergebnisse wieder zu einem Gesamtdatensatz zusammengesetzt werden, der keine Spuren der Partitionierung mehr enthält. Dabei sollen aus den Ergebnisdaten der Berechnung für eine Partition möglichst immer die Objekte übernommen werden, die sich innerhalb des Partitionspolygons befinden, da wir diese Ergebnisse als korrekt ansehen. Die aufgrund von fehlendem Kontext möglicherweise fehlerbehafteten Ergebnisdaten außerhalb des Polygons sind zu verwerfen und aus dem Ergebnis der Partition zu übernehmen, die diese Daten im Inneren enthält. Unterschiedliche Rekompositionsoperationen werden insbesondere benötigt, um verschiedene Konflikte zwischen ausgedehnten Objekten aufzulösen, die die Grenze zwischen benachbarten Partitionen überlappen. Beispiele passender Partitionierungs- und Rekompositionsoperationen für eine geographische Anwendung werden wir in Abschnitt 4 vorstellen.

3.3 Repräsentationsmodelle

Ein wesentliches Ziel beim Entwurf des Partitionierungsdienstes ist Flexibilität. Die von diesem Dienst angebotenen Operationen zur Partitionierung und Rekomposition sollen für eine möglichst große Vielfalt von Geoprogrammen anwendbar sein, um Berechnungen auf großen Datensätzen zu ermöglichen. Geographische Daten können jedoch auf viele verschiedene Arten strukturiert sein. Das Datenbankschema für einen geographischen Datensatz bezeichnen wir im Folgenden als *Repräsentationsmodell* dieser Daten. Um trotz der Heterogenität unterschiedlicher Repräsentationsmodelle nicht für jede Anwendung eigene Operationen anbieten zu müssen, identifizieren wir typische Strukturen bzw. Teilschemata, die in vielen Modellen auftreten. Sind die für eine Anwendung relevanten Informationen in einer solchen Struktur modelliert oder lassen sich in diese transformieren, können wir Operationen zur Partitionierung und Rekomposition einsetzen, die für ein solches Teilschema implementiert sind.

Der zentrale Teil des Schemas für einen geographischen Datensatz bildet häufig die in Abbildung 3 dargestellte Struktur. Die zu beschreibenden Merkmale der Erdoberfläche sind in den Daten durch Objekte dargestellt, die neben der räumlichen Beschreibung durch eine Geometrie noch einen innerhalb des Datensatzes eindeutigen Identifikator und eine Objektart besitzen. Letztere legt fest, um was für einen Typ (z.B. Straße oder Ackerfläche) von Objekt es sich handelt. Zur feineren Charakterisierung der Objekte können weitere Attribute verwendet werden, wobei die erlaubten Typen von

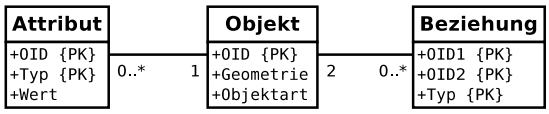


Abb. 3: Objekte, Attribute und Beziehungen

Attributen häufig von der Objektart abhängen (z.B. Breite der Fahrbahn für Straßen). Weiterhin benötigt man Beziehungen zwischen den Objekten, die ebenfalls von unterschiedlichem Typ sein können, um z.B. Über- und Unterführungen an Straßenkreuzungen zu modellieren.

Durch ein Repräsentationsmodell können für die Geometrien der Objekte verschiedene Einschränkungen festgelegt sein, die es beim Partitionieren und Rekomponieren zu berücksichtigen gilt. Eine häufige Einschränkung betrifft z.B. den Geometrietyp, wenn nur Punkte, Linien oder Flächen in einem Datensatz enthalten sind. Weitere gebräuchliche Einschränkungen sind die Forderung der Schnittfreiheit, d.h. Geometrien verschiedener Objekte dürfen sich nur an den Rändern überschneiden, oder das Verbot von Lücken, so dass der komplette Datenraum durch Flächenobjekte überdeckt sein muss. Ein Beispiel für Repräsentationsmodelle mit solchen Einschränkungen sind Landbedeckungsdaten [6]. Diese bestehen aus einer schnittfreien und lückenlosen Menge von Flächenobjekten, denen als Objektart jeweils eine Landbedeckungsklasse (z.B. Nadelwald) zugeordnet ist.

Repräsentationsmodelle mit linien- oder flächenhaften Geometrien, die nicht schnittfrei sind, werden ein wenig abwertend auch als *Spaghetti-Datenmodelle* bezeichnet [4]. Während sie für die Herstellung von Karten gut geeignet sind, bevorzugt man für raumbezogene Analysen sogenannte *topologische Datenmodelle (TDM)*. Die grundlegende Struktur eines topologischen Datenmodells ist in Abbildung 4 dargestellt. Knoten bilden punktförmige Objekte in einem TDM

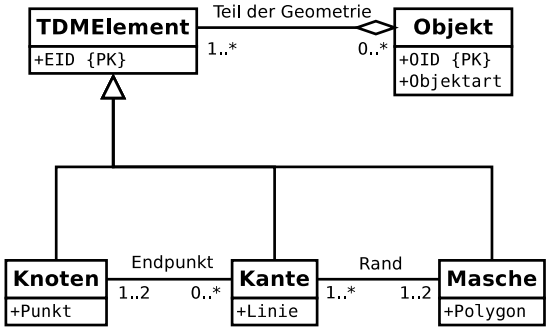


Abb. 4: Datenstruktur eines TDMs

ab und können außerdem die Endpunkte von Kanten darstellen. Kanten wiederum repräsentieren linienhafte Objekte und bilden die Ränder von Maschen. Und solche Maschen entsprechen Teilen von flächenhaften Objekten. Die drei Mengen verschiedenartiger TDM-Elemente sind jeweils schnittfrei, allerdings können ein Knoten oder eine Kante in einer Masche enthalten sein. Ein geographisches Objekt ist durch die TDM-Elemente repräsentiert, durch deren Aggregation man die Geometrie des Objekts rekonstruieren kann. Dabei kann dasselbe Element (z.B. eine Kante) zu mehreren Objekten gehören und repräsentiert in diesem Fall einen gemeinsamen Bestandteil der Geometrien der Objekte, der folglich im Gegensatz zum Spaghetti-Modell nur einmal abgespeichert werden muss. Ein weiterer Unterschied besteht darin, dass im TDM die topologischen Beziehungen der Elemente zueinander explizit gespeichert werden. Dadurch können z.B. benachbarte Flächen bestimmt werden, ohne dass man geometrische Berechnungen durchführen muss.

Weitere typische Schemata werden bei der Integration von Daten verwendet, um Zuordnungen von Objekten aus unterschiedlichen Datensätzen zu modellieren [7]. Wir verzichten aus Platzgründen auf eine detaillierte Beschreibung.

4. ANWENDUNGSBEISPIEL

Als anschauliches Beispiel für die Verwendung unterschiedlicher Partitionierungs- und Rekompositionsoperationen betrachten wir in diesem Abschnitt das geometrische Problem der Liniensegmentierung. Dieses besteht darin, eine Menge von Linienobjekten, die sich beliebig überschneiden dürfen, in eine schnittfreie Menge von Linien zu transformieren. Es gibt eine Reihe von nützlichen Anwendungen, wie z.B. zur Erzeugung der Menge von Kanten bei der Transformation von Spaghetti-Daten in ein topologisches Datenmodell. Die hier verwendete Implementierung besteht aus zwei Schritten. Zuerst werden mit Hilfe eines Plane-Sweep-Verfahrens [1] alle Paare sich schneidender Linien bestimmt. Anschließend wird jede Linie an allen Schnittpunkten unterteilt, so dass diese im Ergebnis durch mehrere Linien dargestellt wird, die sich mit anderen Linien nur noch an den Endpunkten überschneiden. Liegen dabei zwei Linien auf einem längeren Abschnitt übereinander, wird für diesen Abschnitt nur eine einzelne Linie ins Ergebnis übernommen.

Im Folgenden stellen wir für die Liniensegmentierung drei Strategien zur Partitionierung und Rekomposition aus dem in Abschnitt 3 beschriebenen Partitionierungsdienst vor. Es sei angemerkt, dass die für diese Strategien angebotenen Datenbankprozeduren in der Anwendbarkeit nicht auf dieses eine Problem eingeschränkt sind, sondern sich auch für viele weitere Geoprogramme einsetzen lassen. Beispielsweise wurde Variante 1 bereits in [6] erfolgreich für die Generalisierung von Flächendaten angewendet.

4.1 Clipping & Vereinigung

Wir wählen in dieser Variante für eine Partition alle Objekte aus, deren Geometrien sich mit dem Partitionspolygon überschneiden. Zusätzlich schneiden wir bei Objekten am Rand der Partition den Teil der Geometrie ab, der über das Partitionspolygon herausragt. Folglich verbleibt nur der innerhalb des Partitionspolygons liegende Anteil als geometrische Repräsentation des Objekts in der Partition. Dies wird allgemein auch als *Clipping* bezeichnet. Durch diese Art von Aufteilung kann ein Linienobjekt ggf. in mehreren Partitionen auftreten, allerdings jeweils mit einem unterschiedlichen Teil seiner Geometrie, weshalb wir die Partitionierung als disjunkt bezeichnen können. Bei der Liniensegmentierung für eine Partition werden alle Schnitte von Objekten gefunden, die innerhalb des Partitionspolygons liegen, und die Geometrien entsprechend aufgeteilt.

Betrachtet man die Ergebnisse der Liniensegmentierung für die einzelnen Partitionen gemeinsam, fällt auf, dass neben den korrekten Unterteilungen an den Linienschnittpunkten zusätzlich Unterteilungen an den Schnittpunkten mit den Rändern der Partitionen auftreten, was auf das Clipping zurückzuführen ist. Man betrachte z.B. die Situation in Abbildung 5, in der aus drei Linienobjekten a, b und c

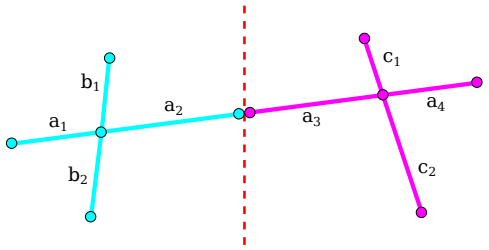


Abb. 5: Überflüssige Segmentierung

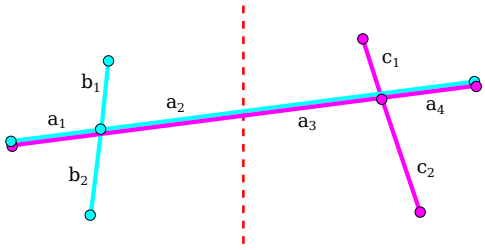


Abb. 6: Überlappende Linien

insgesamt acht segmentierte Linien erzeugt wurden. Bei der Unterteilung zwischen a_2 und a_3 liegt allerdings kein echter Schnitt vor. Da derartige Situationen ohne Partitionierung nicht auftreten würden, liegen hier Rekompositionskonflikte vor und müssen durch eine geeignete Operation bereinigt werden. Dazu bestimmen wir, welche Linien aus dem bereits zusammengesetzten Ergebnis welche anderen Linien aus der aktuellen Partition an der Partitionsgrenze berühren und fügen diese durch eine *Vereinigung* zu einer Linie zusammen.

Wir müssen allerdings auch berücksichtigen, dass ein Linienobjekt aus dem Ausgangsdatensatz mehrmals den Rand der Partition schneiden kann, weshalb wir ggf. auch Gruppen von mehr als zwei Linien zu einem Objekt vereinigen müssen. Ein weiterer Sonderfall liegt vor, wenn sich zwei Linien aus dem Ausgangsdatensatz genau auf dem Rand eines Partitionspolygons überschneiden, weil es sich dann bei dem nach obiger Vorschrift ermittelten Berührungspunkt um einen echten Schnittpunkt handelt und somit keine Vereinigung stattfinden darf. Wir können diese Situationen dadurch erkennen, dass sich zwei segmentierte Linien aus derselben Partition an einem solchen Punkt berühren.

4.2 Partitionsüberschneidung & Durchschnitt

Wie in der ersten Variante wählen wir alle Objekte aus, die das Partitionspolygon schneiden, verzichten aber auf Clipping. Die Intention dabei ist es, aufwändige Berechnungen zum Abschneiden und Vereinigen der Geometrien einzusparen. Dafür nehmen wir etwas Redundanz in Kauf, denn Linien aus dem Ausgangsdatensatz, die über den Rand der Partition hinausragen, werden in mehreren Partitionen mit ihrer vollständigen Geometrie repräsentiert, so dass eine nicht disjunkte Partitionierung vorliegt. Schnitte zwischen solchen Objekten werden demnach bei der Segmentierung mehrfach in unterschiedlichen Partitionen berechnet.

Beim Rekomponieren einer Partition müssen *Duplikate* aus den Ergebnissen entfernt werden, da die aus den mehrfach repräsentierten Objekten gebildeten Linien auch mehrfach in den Ergebnissen auftreten. Allerdings stimmen diese Duplikate in Bezug auf die Segmentierung nicht zwingend überein, denn der Schnitt einer über den Rand der Partition hinausragenden Linie mit einer Linie, die komplett außerhalb der Partition liegt, wird bei der Berechnung in dieser Partition nicht gefunden. Man betrachte z.B. die Situation in Abbildung 6. Während im Ergebnis der linken Partition (cyan) die Linie a am Schnittpunkt mit b in a_1 und a_2 unterteilt wurde, fehlt die Unterteilung am Schnittpunkt mit *c*. Für das Ergebnis der rechten Partition (magenta) hingegen ist die Situation genau umgekehrt.

Um Duplikate zu entfernen, verwerfen wir beim Rekomponieren einer Partition zunächst alle Linien, die komplett außerhalb des Partitionspolygons liegen, denn diese sind entweder bereits im Ergebnis enthalten oder werden beim Rekomponieren einer der nächsten Partitionen eingefügt. Für Situationen wie in Abbildung 6 bestimmen wir für die sich überlappenden Linien den gemeinsamen Teil durch eine geometrische *Durchschnittsoperation*. Z.B. fügen wir anstelle der zu langen Linien a_2 und a_3 nur deren Durchschnitt ins Gesamtergebnis ein.

4.3 Objektüberschneidung & Duplikate

Im Vergleich zur vorigen Variante können wir die Eliminierung der Duplikate bei der Rekomposition weiter vereinfachen, wenn wir bei der Partitionierung noch mehr Redundanz hinzufügen. Wir bezeichnen dazu die Linienobjekte, die das Polygon der Partition p schneiden, als p-Objekte. Um alle p-Objekte bei der Berechnung für diese Partition korrekt zu segmentieren, müssen wir bei der Partitionierung noch Linien von außerhalb der Partition hinzunehmen, die sich mit einem p-Objekt überschneiden.

Dadurch, dass bei der Partitionierung mehr Objekte redundant für mehrere Partitionen ausgewählt werden, entstehen auch mehr Duplikate, die bei der Rekomposition entfernt werden müssen (siehe Abbildung 7). Die meisten dieser Duplikate werden wir wieder los, indem wir (wie bei Variante 2) Linien außerhalb des Partitionspolygons verwerfen (z.B. a_1'/a_4'). Bei allen Linien im Ergebnis einer Partition, die den Rand des Partitionspolygons schneiden, können allerdings in dieser Variante nur echt übereinstimmende Duplikate auftreten, denn diese Linien wurden vollständig segmentiert. Anstatt Durchschnitte zu berechnen, reicht es somit aus, von den am Rand der Partition auftretenden Duplikaten (hier a_2/a_3) jeweils ein Objekt ins Ergebnis zu übernehmen.

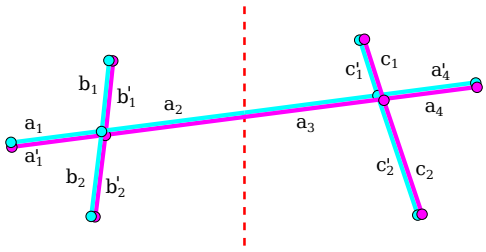


Abb. 7: Doppelte Linien

5. ERGEBNISSE

Um die Anwendbarkeit der in Abschnitt 4 vorgestellten Partitionierungs- und Rekompositionsoperationen zu demonstrieren und die Varianten miteinander zu vergleichen, führen wir Tests mit einem ca. 7,2GB großen kommerziell pro-

duzierten Datensatz durch, der das gesamte Bundesland Hessen umfasst. Diese Daten enthalten Informationen über Straßen und weitere für den Kraftfahrzeugverkehr relevante geographische Objekte in Form von Liniengeometrien, die für die kartographische Darstellung optimiert und insbesondere nicht schnittfrei sind. Partitionierung und Rekomposition sind in einer Datenbank (Oracle 11g) mit räumlicher Erweiterung (Oracle Spatial) implementiert. Zur Segmentierung verwenden wir ein Programm aus einer Java-Bibliothek für geometrische Berechnungen (JTS Topology Suite), das durch zahlreiche Optimierungen auf Kosten eines hohen Speicherverbrauchs sehr effizient arbeitet.

Wir führen für diesen Datensatz mit jeder der drei Varianten eine Segmentierung durch, wobei wir ein Partitionierungsschema aus 129 Quadraten mit jeweils 25km Kantenlänge verwenden. Wir messen und summieren dabei jeweils separat die Laufzeiten, die bei der Partitionierung, Segmentierung und Rekomposition für alle Partitionen benötigt werden. Diese Laufzeiten sind in Abbildung 8 dargestellt. Die Laufzeit für die Segmentierung ist erwartungsgemäß in

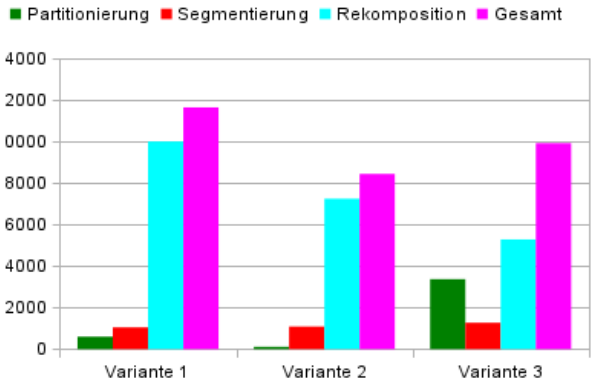


Abb. 8: Laufzeiten (in Sek.) der Prozessphasen

der ersten Variante am geringsten und steigt durch das Hinzufügen von mehr Redundanz in Variante 2 und 3 jeweils leicht an. Am meisten Zeit benötigt in allen Varianten die Rekomposition. Während diese in Variante 3 die geringste Laufzeit benötigt, ist dabei jedoch eine vergleichsweise aufwändige Partitionierung nötig. Die beste Gesamtlaufzeit hat somit Variante 2, bei der die Partitionierung am schnellsten geht und die Zeiten für Segmentierung und Rekomposition jeweils in der Mitte liegen.

6. FAZIT

Der in diesem Beitrag vorgestellte Partitionierungsdienst für geographische Daten in räumlichen Datenbanken besteht im Wesentlichen aus einer Sammlung von flexibel anwendbaren Partitionierungs- und Rekompositionsoperationen. Diese erlauben es, Geoprogramme mit sehr geringem Entwicklungsaufwand fit für die Bearbeitung großer Datenmengen zu machen. Dabei bleibt die Freiheit der Wahl einer Programmiersprache und von Software-Bibliotheken sowie die gute Wartbarkeit der Geoprogramme erhalten, da Zugriffe auf extern gespeicherte Daten nur während der Partitionierung und Rekomposition erfolgen müssen. Neben dem an dieser Stelle vorgestellten Anwendungsbeispiel eignet sich diese Vorgehensweise für eine Vielzahl weiterer geographischer Problemstellungen, sofern diese durch hinreichend lokale Algorithmen gelöst werden können, so dass die benötigte Redundanz bei der Partitionierung nicht zu groß wird.

Während für die in diesem Beitrag vorgestellten Experimente die Größe der Partitionen fest vorgegeben wurde, ist es für einen Anwender des Partitionierungsdienstes wünschenswert, stattdessen die Größe des verfügbaren Speichers angeben zu können, für die der Dienst dann ein geeignetes Partitionierungsschema berechnet. Daher arbeiten wir daran, Modelle aus der Literatur zum Schätzen der Partitionsgröße für räumliche Verbunde zu verallgemeinern, um diese auch auf andere Geoprogramme anwenden zu können. Dabei muss auch berücksichtigt werden, dass reale geographische Daten nicht gleichmäßig verteilt sind, und somit auch die Partitionsgröße innerhalb eines Partitionierungsschemas abhängig von der Datendichte variieren sollte.

Um das Spektrum von möglichen Anwendungen für den Partitionierungsdienst zu vergrößern, muss dieser um weitere Operationen und insbesondere weitere Repräsentationsmodelle erweitert werden. Außerdem werden wir genauer untersuchen, wie sich dieser Dienst möglichst gewinnbringend für die Fortführung von abgeleiteten Datensätzen einsetzen lässt. Dieser Ansatz basiert auf der Idee, bei Updates für Geobasisdaten zunächst möglichst kleine, aber räumlich zusammenhängende Gebiete zu identifizieren, in denen Änderungen stattgefunden haben. Für die Aktualisierung von abgeleiteten Datensätzen müssen dann unter Verwendung von Partitionierung und Rekomposition nur diese Gebiete an Stelle des kompletten Datensatzes neu berechnet werden.

7. DANKSAGUNG

Diese Arbeit wurde vom Bundesamt für Kartographie und Geodäsie im Rahmen des Projekts Wissensbasierter Photogrammetrisch-Kartographischer Arbeitsplatz (WiPKA) gefördert.

8. LITERATUR

[1] Berg, M. de ; Cheong, O. ; Kreveld, M. van ; Overmars, M. : *Computational Geometry: Algorithms and Applications*. 3.Aufl. Springer-Verlag, 2008

[2] Dittrich, J.-P. ; Seeger, B. : Data Redundancy and Duplicate Detection in Spatial Join Processing. In: *Proc. ICDE 2000*, San Diego, S. 535–546

[3] Güting, R. H. ; Schilling, W. : A Practical Divide-and-Conquer Algorithm for the Rectangle Intersection Problem. In: *Information Sciences* 42 (1987), Nr. 2, S. 95–112

[4] Hake, G. ; Grünreich, D. ; Meng, L. : *Kartographie*. 8.Aufl. Walter de Gruyter & Co., 2002

[5] Patel, J. M. ; DeWitt, D. J.: Partition Based Spatial-Merge Join. In: *Proc. SIGMOD 1996*, Montreal, S. 259–270

[6] Thiemann, F. ; Warneke, H. ; Sester, M. ; Lipeck, U. : A Scalable Approach for Generalization of Land Cover Data. In: *Proc. 14th AGILE Intl.Conf. on Geographic Information Systems*. Utrecht, 2011, S. 399–420

[7] Warneke, H. ; Schäfers, M. ; Lipeck, U. ; Bobrich, J. : Matching-Based Map Generalization by Transferring Geometric Representations. In: *Proc. Geoinformatik 2011*, Münster, S. 71–77

[8] Zhou, X. ; Abel, D. J. ; Truffet, D. : Data Partitioning for Parallel Spatial Join Processing. In: *GeoInformatica* 2 (1998), Nr. 2, S. 175–204

Cloud Data Management: A Short Overview and Comparison of Current Approaches

Siba Mohammad
Otto-von-Guericke University Magdeburg
siba.mohammad@iti.uni-magdeburg.de

Sebastian Breß
Otto-von-Guericke University Magdeburg
sebastian.bress@st.ovgu.de

Eike Schallehn
Otto-von-Guericke University Magdeburg
eike@iti.cs.uni-magdeburg.de

ABSTRACT

To meet the storage needs of current cloud applications, new data management systems were developed. Design decisions were made by analyzing the applications workloads and technical environment. It was realized that traditional Relational Database Management Systems (RDBMSs) with their centralized architecture, strong consistency, and relational model do not fit the elasticity and scalability requirements of the cloud. Different architectures with a variety of data partitioning schemes and replica placement strategies were developed. As for the data model, the key-value pairs with its variations were adopted for cloud storage. The contribution of this paper is to provide a comprehensible overview of key-characteristics of current solutions and outline the problems they do and do not address. This paper should serve as an entry point for orientation of future research regarding new applications in cloud computing and advanced requirements for data management.

1. INTRODUCTION

Data management used within the cloud or offered as a service from the cloud is an important current research field. However, there are only few publications that provide a survey and compare the different approaches. The contribution of this paper is to provide a starting point for researchers and developers who want to work on cloud data management. Cloud computing is a new technology that provides resources as an elastic pool of services in a pay-as-you-go model [5]. Whether it is storage space, computational power, or software, customers can get it over the internet from one of the cloud service providers. Big players in the market, such as Google [8], Amazon [13], Yahoo! [9], and Hadoop [7], defined the assumptions for cloud storage systems based on analyzing the technical environment and applications workload. First, a data management system will work on a cluster of storage nodes where components failure is the normal situation rather than the exception. Thus, fault tolerance and recovery must be built in. The system must be portable across heterogeneous hardware and software platforms. It will store tera bytes of data, thus parameters of I/O operations and block sizes must be adapted to large sizes. Based on these assumptions the requirements for a cloud DBMS are: elasticity, scalability, fault tolerance and self manageability [11]. The rest of the paper is organized as follows. First, we provide an overview of the architecture and the family tree of the cloud storage systems. Then, we discuss how different systems deal with the trade-off in the Consistency, Availability, Partition tolerance (CAP) theorem. After that, we discuss different schemes used for data partitioning and replication. Then, we provide a list of the cloud data models.

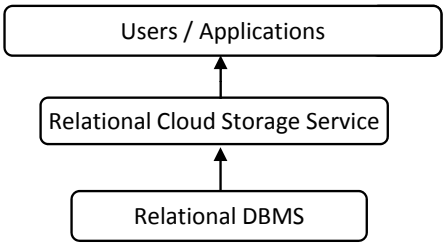


Figure 1: RDBMS as a service

2. ARCHITECTURE OVERVIEW

There are two main approaches to provide data management systems for the cloud. In the first approach, each customer gets an own instance of a Database Management System (DBMS), which runs on virtual machines of the service provider [10] as illustrated in Figure 1. The DBMS supports full ACID requirements with the disadvantage of loosing scalability. If an application requires more computing or storage resources than the maximum allocated for an instance, the customer must implement partitioning on the application level using a different database instance for each partition [1]. Another solution is on demand assignment of resources to instances. Amazon RDS is an example of relational database services, that supports MySQL, Oracle, and SQL Server.

In the second approach, data management is not provided as a conventional DBMS on a virtualized platform, but as a combination of interconnected systems and services that can be combined according to application needs. Figure 2 illustrates this architecture. The essential part is the dis-

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

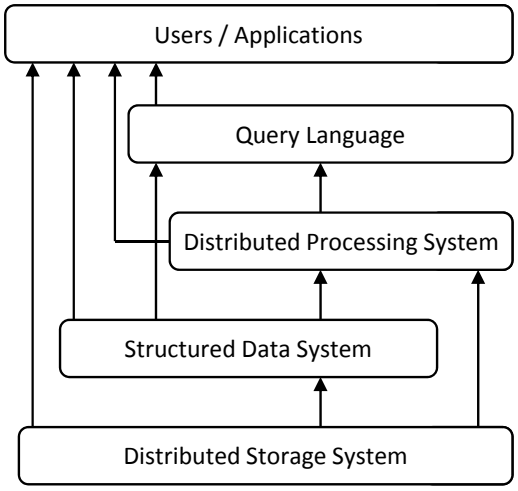


Figure 2: Cloud data management architecture

tributed storage system. It is usually internally used by the cloud services provider and not provided as a public service. It is responsible for providing availability, scalability, fault tolerance, and performance for data access. Systems in this layer are divided in three categories:

- Distributed File Systems (DFS), such as Google's File System (GFS).
- Cloud based file services, such as Amazon's Simple Storage Service (S3).
- Peer to peer file systems, such as Amazon's Dynamo.

The second layer consists of structured data systems and provides simple data models such as key-value pairs, which we discuss in Section 6. These systems support various APIs for data access, such as SOAP and HTTP. Examples of systems in this layer are Google's Bigtable, Cassandra, and SimpleDB. The third layer includes distributed processing systems, which are responsible for more complex data processing, e.g, analytical processing, mass data transformation, or DBMS-style operations like joins and aggregations. MapReduce [12] is the main processing paradigm used in this layer.

The final layer includes query languages. SQL is not supported. However, developers try to mimic SQL syntax for simplicity. Most query languages of cloud data management systems support access to one domain, key space, or table, i.e., do not support joins [4, 2]. Other functionalities, such as controlling privileges and user groups, schema creation, and meta data access are supported. Examples of query languages for cloud data are HiveQL, JAQL, and CQL. The previous components complement each other and work together to provide different sets of functionalities. One important design decision that was made for most cloud data query languages is not supporting joins or aggregations. Instead, the MapReduce framework is used to perform these operations to take advantage of parallel processing on different nodes within a cluster. Google pioneered this by providing a MapReduce framework that inspired other systems.

For more insight into connections and dependencies between these systems and components, we provide a family

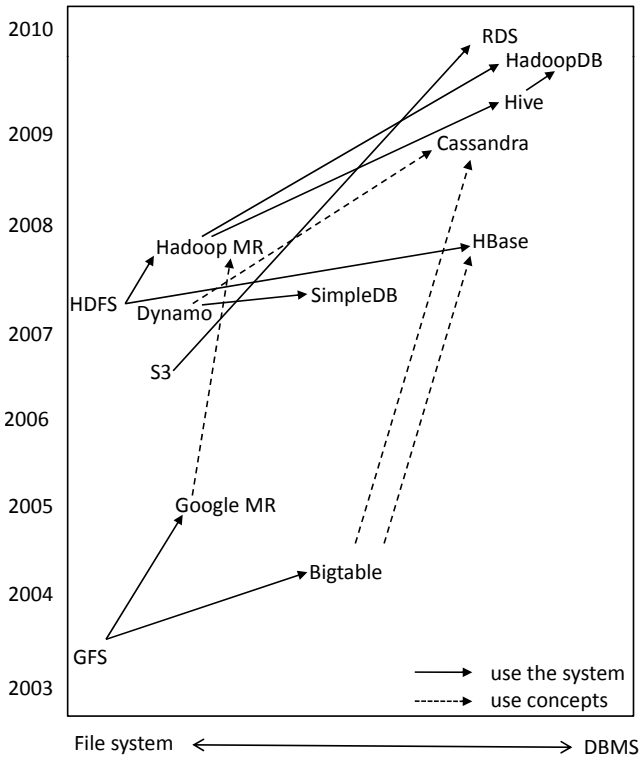


Figure 3: Family tree of cloud data management systems

tree of cloud storage systems as illustrated in Figure 3. We use a solid arrow to illustrate that a system uses another one such as Hive using HDFS. We use a dotted arrow to illustrate that a system uses some aspects of another system like the data model or the processing paradigm. An example of this is Cassandra using the data model of Bigtable. In this family tree, we cover a range of commercial systems, open source projects, and academic research as well. We start on the left side with distributed storage systems GFS and HDFS. Then, we have the structured storage systems with API support such as Bigtable. Furthermore, there are systems that support a simple QL such as SimpleDB. Next, we have structured storage systems with support of MapReduce and simple QL such as Cassandra and HBase. Finally, we have systems with sophisticated QL and MapReduce support such as Hive and HadoopDB. We compare and classify these systems based on other criteria in the coming sections. An overview is presented in Figure 4.

3. CONSISTENCY, AVAILABILITY, PARTITION TOLERANCE (CAP) THEOREM

Tightly related to key features of cloud data management systems are discussions on the CAP theorem [14]. It states that consistency, availability, and partition tolerance are systematic requirements for designing and deploying applications for distributed environments. In the cloud data management context these requirements are:

- Consistency: includes all modifications on data that must be visible to all clients once they are committed.

System / Property	Distributed Storage/File System				Structured Data Systems						Analytical Processing Systems		RDBMS as service
	Amazon S3	GFS	HDFS	Dynamo	SimpleDB	Bigtable	HBase	PNUTS	Cassandra	CouchDB	Hive	HadoopDB	RDS
Consistency Model													
ACID												X	X
BASE	X			X	X					X			
SCLA						X	X				X		
Tunable Consistency								X	X				
Partitioning Scheme													
Hash											X	X	
Range						X	X	X					
List								X			X		
Composite				X					X				
Partition Level		file	file	Key space		table	table	table	table			table	
Partition		chunk	chunk	set of items		tablet	region	tablet	set of items		bucket	chunk	
Data Models													
Key Value	X			X	X	X	X		X	X	X		
Row Oriented					X								
Wide Column						X	X		X		X		
Document Oriented										X			
Relational								X				X	X
Replication													
Rack aware		X	X				X		X		X	X	
Rack unaware	X					X			X				
Datacenter aware				X	X			X	X				
Replication Level	item	chunk	block	item	domain	DB file	DB file	tablet	record	DB	DB file	chunk	DB
Map/Reduce													
Internally											X	X	
Input for	X	X	X			X	X	X	X	X			
Interface													
Query Lang					X		X		X		X	X	X
API	X			X	X	X	X	X	X	X	X	X	X

Figure 4: Overview and classification of cloud data management systems(Gray field begins a new category of properties. Dark gray field means that the property is not applicable on a system)

At any given point in time, all clients can read the same data.

- Availability: means that all operations on data, whether read or write, must end with a response within a specified time.
- Partition tolerance: means that even in the case of components' failures, operations on the database must continue.

The CAP theorem also states that developers must make trade-off decisions between the three conflicting requirements to achieve high scalability. For example, if we want a data storage system that is both strongly consistent and partition tolerant, the system has to make sure that write operations return a success message only if data has been committed to all nodes, which is not always possible because of network or node failures. This means that its availability will be sacrificed.

In the cloud, there are basically four approaches for DBMSs in dealing with CAP:

Atomicity, Consistency, Isolation, Durability (ACID): With ACID, users have the same consistent view of data before and after transactions. A transaction is atomic, i.e., when one part fails, the whole transaction fails and the state of data is left unchanged. Once a transaction is committed, it is protected against crashes and errors. Data is locked while being modified by a transaction. When another transaction tries to access locked data, it has to wait until data is unlocked. Systems that support ACID are used by applications that require strong consistency and can tolerate its affects on the scalability of the application as already discussed in Section 2.

Basically Available, Soft-state, Eventual consistency (BASE): The system does not guarantee that all users see the same version of a data item, but guarantees that all of them get a response from the systems even if it means getting a stale version. Soft-state refers refers to the fact, that the current status of a managed object can be ambiguous, e.g. because there are several temporarily inconsistent replicas of it stored. Eventually consistent means that updates will propagate through all replicas of a data item in a distributed system, but this takes time. Eventually, all replicas are updated. BASE is used by applications that can tolerate weaker consistency to have higher availability. Examples of systems supporting BASE are SimpleDB and CouchDB.

Strongly Consistent, Loosely Available (SCLA): This approach provides stronger consistency than BASE. The scalability of systems supporting SCLA in the cloud is higher compared to those supporting ACID. It is used by systems that choose higher consistency and sacrifice availability to a small extent. Examples of systems supporting SCLA are HBase and Bigtable.

Tunable consistency: In this approach, consistency is configurable. For each read and write request, the user decides the level of consistency in balance with the level of availability. This means that the system can work in high consistency or high availability and other degrees in between. An example of a system supporting tunable consistency is Cassandra [15] where the user determines the number of replicas that the system should update/read. Another example is PNUTS [9], which provides per record time-line consistency. The user determines the version number to query at several points in the consistency time-line. In Figure 5, we classify different cloud data management systems based on the consistency model they provide.

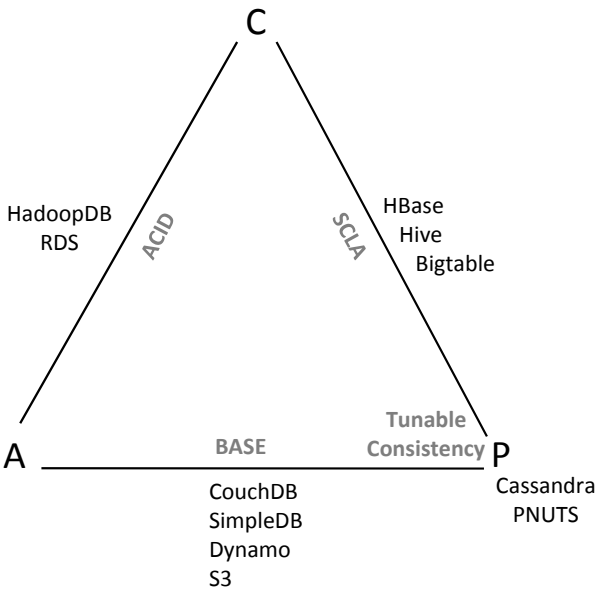


Figure 5: Classification of cloud data management systems based on CAP

4. PARTITIONING TECHNIQUES

Partitioning, also known as sharding, is used by cloud data management systems to achieve scalability. There is a variety of partitioning schemes used by different systems on different levels. Some systems partition data on the file level while others horizontally partition the key space or table. Examples of systems partitioning data on the file level are the DFSs such as GFS and HDFS which partition each file into fixed sized chunks of data. The second class of systems which partition tables or key space uses one of the following partitioning schemes [18, 6] :

List Partitioning A partition is assigned a list of discrete values. If the key of the inserted tuple has one of these values, the specified partition is selected . An example of a system using list as the partitioning scheme is Hive [20].

Range Partitioning The range of values belonging to one key is divided into intervals. Each partition is assigned one interval. A partition is selected if the key value of the inserted tuple is inside a certain range. An example of a system using range partitioning is HBase.

Hash Partitioning The output of a hash function is assigned to different partitions. The hash function is

applied on key values to determine the partition. This scheme is used when data does not lend itself to list and range partitioning. An example of a system using hash as the partitioning scheme is PNUTS.

There are some systems that use a composite partitioning scheme. An example is Dynamo, which uses a composite of hash and list schemes (consistent hashing). Some systems allow partitioning data several times using different partitioning schemes each time. An example is Hive, where each table is partitioned based on column values. Then, each partition can be hash partitioned into buckets, which are stored in HDFS.

One important design consideration to make is whether to choose an order-preserving partitioning technique or not. Order preserving partitioning has an advantage of better performance when it comes to range queries. Examples of systems using order preserving partitioning techniques are Bigtable and Cassandra. Since most partitioning methods depend on random position assignment of storage nodes, the need for load balancing to avoid non uniform distribution of data and workloads is raised. Dynamo [13] focuses on achieving a uniform distribution of keys among nodes assuming that the distribution of data access is not very skewed, whereas Cassandra [17] provides a load balancer that analyzes load information to lighten the burden on heavily loaded nodes.

5. REPLICATION TECHNIQUES

Replication is used by data management systems in the cloud to achieve high availability. Replication means storing replicas of data on more than one storage node and probably more than one data center. The replica placement strategy affects the efficiency of the system [15]. In the following we describe the replication strategies used by cloud systems:

Rack Aware Strategy: Also known as the Old Network Topology Strategy. It places replicas in more than one data center on different racks within each data center.

Data Center Aware Strategy: Also known as the New Network Topology Strategy. In this strategy, clients specify in their applications how replicas are placed across different data centers.

Rack Unaware Strategy: Also known as the Simple Strategy. It places replicas within one data center using a method that does not configure replica placement on certain racks.

Replication improves system robustness against node failures. When a node fails, the system can transparently read data from other replicas. Another gain of replication is increasing read performance using a load balancer that directs requests to a data center close to the user. Replication has a disadvantage when it comes to updating data. The system has to update all replicas. This leads to very important design considerations that impact availability and consistency of data. The first one is to decide whether to make replicas available during updates or wait until data is consistent across all of them. Most systems in the cloud choose availability over consistency. The second design consideration is to decide when to perform replica conflicts resolution, i.e., during writes or reads. If conflict resolution is done during write operations, writes could be rejected if the system can not reach all replicas or a specified number of them within a specific time. Example of that is the WRITE ALL operation in Cassandra, where the write fails if the system could not reach all replicas of data. However, some systems in the cloud choose to be always writeable and push conflict resolution to read operations. An example of that is Dynamo which is used by many Amazon services like the shopping cart service where customer updates should not be rejected.

6. DATA MODEL

Just as different requirements compared to conventional DBMS-based applications led to the previously described different architectures and implementation details, they also led to different data models typically being used in cloud data management. The main data models used by cloud systems are:

Key-value pairs It is the most common data model for cloud storage. It has three subcategories:

- Row oriented: Data is organized as containers of rows that represent objects with different attributes. Access control lists are applied on the object (row) or container (set of rows) level [19]. An example is SimpleDB.
- Document Oriented: Data is organized as a collection of self described JSON documents. Document is the primarily unit of data which is identified by a unique ID. Documents are the unit for access control [16]. Example of a cloud data management system with document oriented data model is CouchDB.
- Wide column: In this model, attributes are grouped together to form a column family. Column family information can be used for query optimization. Some systems perform access control and both disk and memory accounting at the column family level [8, 17, 3]. An example of that is Bigtable. Systems of wide column data model should not be mistaken with column oriented DB systems. The former deals with data as column families on the conceptual level only. The latter is more on the physical level and stores data by column rather than by row.

Relational Model (RM) The most common data model for traditional DBMS is less often used in the cloud. Nevertheless, Amazon's RDS supports this data model and PNUTS a simplified version of it.

7. SUMMARY AND CONCLUSION

The cloud with its elasticity and pay-as-you-go model is an attractive choice for outsourcing data management applications. Cloud service providers, such as Amazon and Microsoft, provide relational DBMSs instances on virtual machines. However, the cloud technical environment, workloads, and elasticity requirements lead to the development of new breed of storage systems. These systems range from highly scalable and available distributed storage systems with simple interfaces for data access to fully equipped DBMSs that support sophisticated interfaces, data models, and query languages.

Cloud data management systems faced with the CAP theorem trade-off provide different levels of consistency ranging from eventual consistency to strict consistency. Some systems allow users to determine the level of consistency for each data input/output request by determining the number of replicas to work with or the version number of the data item. List, range, hash, and composite partitioning schemes are used to partition data to achieve scalability. With partitioning comes the need for load balancing with two basic methods: uniform distribution of data and workloads, and analyzing load information. With data partitioned and distributed over many nodes, taking into consideration the possibility of node and network failures, comes the need for replication to achieve availability. Replication is done on the partition level using rack aware, data center aware, and rack unaware placement strategies. Cloud data management systems support relational data model, and key-value pairs data model. The key-value pairs is widely used with different variations: document oriented, wide column, and row oriented.

As outlined throughout this paper, several typical properties of traditional DBMS, such as advanced query languages, transaction processing, and complex data models, are mostly not supported by cloud data management systems. Some of them, because they are simply not required for current cloud applications. Others, because their implementation would lead to losing some of the important advantages, e.g., scalability and availability, of current cloud data management approaches. On the one hand, providing features of conventional DBMS appears to lead toward worthwhile research directions and is currently addressed in ongoing research. On the other hand, advanced requirements, which are completely different may arise from future cloud applications, e.g., interactive entertainment, on-line role playing games, and virtual realities, have interesting characteristics of continuous, collaborative, and interactive access patterns, sometimes under real-time constraints. In a similar way, new ways of human-computer interaction in real-world environments addressed, for instance, in ubiquitous computing and augmented reality are often very data-intensive and sometimes require expensive processing, which could be supported by cloud paradigms. Nevertheless, neither traditional DBMS nor cloud data management can currently sufficiently support those applications.

8. ACKNOWLEDGMENTS

We would like to thank the Syrian ministry of higher education for partially funding this research.

9. REFERENCES

[1] Amazon RDS FAQ. http://aws.amazon.com/rds/faqs/. [online; accessed 10-March-2012].

[2] Cassandra Query Language Documentation. http://caqel.deadcafe.org/cql-doc. [online; accessed 25-July-2011].

[3] HBase: Bigtable-like structured storage for Hadoop HDFS. http://wiki.apache.org/hadoop/hbase. [online; accessed 01-March-2012].

[4] Jaql Overview. http://www.almaden.ibm.com/cs/projects/jaql/. [online; accessed 31-August-2011].

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, 2009.

[6] S. S. Avi Silberschatz, Henry F. Korth. *Database System Concepts Fifth Edition*. McGraw-Hill, 2010.

[7] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[8] 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. In *In Proceedingss of the 7th Conference on Usenix Symposium on Operating Systems Design and Implementation - Volume 7*, pages 205–218, 2006.

[9] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc VLDB Endow.*, 2008.

[10] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In *5th Biennial Conference on Innovative Data Systems Research*, 2011.

[11] S. Das, S. Agarwal, D. Agrawal, and A. E. Abbadi. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. Technical report, 2010.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 2008.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 2007.

[14] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. *SIGOPS Oper. Syst. Rev.*, 1997.

[15] E. Hewitt. *Cassandra The Definitive Guide*. O Reilly Media, Inc, 2010.

[16] J. L. J. Chris Anderson and N. Slater. *CouchDB The Definitive Guide*. OReilly Media, Inc, 2010.

[17] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.

[18] S. B. e. a. Lance Ashdown, Cathy Baird. *Oracle9i Database Concepts*. Oracle Corporation., 2002.

[19] R. H. Prabhakar Chaganti. *Amazon SimpleDB Developer Guide Scale your application's database on the cloud using Amazon SimpleDB*. Packt Publishing Ltd, 2010.

[20] 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. *Proc. VLDB Endow.*, 2009.

Cloud-Services und effiziente Anfrageverarbeitung für Linked Open Data

[Work in Progress]

Heiko Betz, Kai-Uwe Sattler
Department of Computer Science and Automation
TU Ilmenau, Germany
{first.last}@tu-ilmenau.de

ABSTRACT

Der Verbreitungsgrad von Linked Open Data hat in den letzten Jahren massiv zugenommen. Stetig erscheinen neue Quellen, die RDF-Daten frei zur Verfügung stellen. Aktuell diskutiert die Bundesregierung über ein neues Gesetz, welches zur Offenlegung von Daten der öffentlichen Hand verpflichtet. Durch diese Maßnahme, steigt u. a. die Menge an Linked Open Data sehr schnell an. Es werden neue Verfahren benötigt, die mit sehr vielen RDF-Daten gleichzeitig eine effiziente, skalierbare und mit Garantien versehene Abfrage von Daten mittels SPARQL gewährleistet. In dieser Arbeit wird ein reines In-Memory Shared-Nothing-System vorgestellt, das die genannten Anforderungen in effizienter Weise erfüllt. Hierfür werden verschiedenste Optimierungsmaßnahmen ergriffen, die das Potenzial moderner Hardware umfassend ausnutzen.

1. EINFÜHRUNG

Die Anzahl an verfügbaren Quellen für Linked Open Data (LOD) wächst stetig an[1]. Unter anderem wurde vor kurzem von der Bundesregierung die Open Data-Initiative gestartet. Es sollen Daten, die durch Steuergelder finanziert wurden, der Öffentlichkeit frei zur Verfügung stehen[2]. Insgesamt geht die Anzahl an verfügbaren LOD-Quellen in diesem hochverteilten System in die Hunderttausende bzw. Millionen über. Es werden disjunkte bzw. teils überlappende Daten bereitgestellt, die häufig untereinander verlinkt sind, jedoch nicht immer einem gemeinsamen Qualitätsstandard entsprechen.

Die innere Struktur von LOD entspricht dem Ressource Description Framework [1] (RDF) Aufbau. In diesem werden Daten als Tripel repräsentiert (Subjekt, Prädikat und Objekt). Hierauf baut wiederum die Abfragesprache SPARQL [2] auf. Dies ist eine graph-basierte Anfragesprache für RDF-Daten. Sie definiert neben der Möglichkeit Daten abzufragen, ähnlich dem SQL aus der Datenbankwelt, ein Transportprotokoll zwischen verschiedenen SPARQL-Endpoints. In SPARQL selbst werden Basic Graph Patterns (BGP) für die Abfrage von Daten verwendet, die aus einem oder mehreren Tripeln bestehen. Hierdurch ergeben sich joinintensive Abfragen, die eine weitere detaillierte Betrachtung benötigen.

Fallunterscheidung.

Für die Beantwortung von (komplexen) quellenübergreifenden Anfragen müssen nun zwei Fälle unterschieden werden [7]:

1. Verteilter Ansatz: Daten bleiben in den Quellen. Anfragen werden von einem Koordinator entgegengenommen, in Subanfragen aufgesplittet, den entsprechenden Quellen zugesandt, von diesen bearbeitet und das Resultat vom Koordinator entgegengenommen.

2. Data Warehouse Ansatz: Daten werden von verschiedenen Quellen geladen, vorverarbeitet und lokal abgelegt (materialisiert). Anfragen können direkt durchgeführt werden. Die Quellen bleiben gleichzeitig erhalten.

Punkt 1 besitzt mehrere potenzielle Vorteile. Durch die Verteilung wird ein Single Point of Failure (SPOF) vermieden. Der Koordinator muss keine Daten speichern, da sie alle in den Quellen vorliegen. Er muss nur einen geringen Speicherplatz und nur eine geringe Rechenkapazität zur Verfügung stellen. Ein wichtiger Punkt, der für eine Verteilung der Quellen spricht, ist, dass die Daten nicht an andere Unternehmen herausgegeben werden müssen. Es können 'Firmengeheimnisse' bewahrt werden. Als Nachteile ergeben sich jedoch einige Punkte. Die Zerlegung von Anfragen in Subqueries und alle daraus folgenden Schritte sind ein hoch komplexes Problem (Parsing, Normalisierung, eingebettete Anfragen 'herausdrücken', Vereinfachung, Datenlokalisation, Optimierung). Zwei weiter nicht zu unterschätzende Nachteile sind die fehlenden Kontroll- und Überwachungsmechanismen der Quellen. Die Antwortzeit in einem solchen Szenario ist nach oben unbeschränkt. Es ergibt sich, dass keinerlei Garantien abgegeben werden können. Weiterhin ist durch die Verteilung das verwendete Schema dem Koordinator gänzlich unbekannt. Viele Quellen sind zudem nicht in der Lage, SPARQL-Anfragen zu verarbeiten. Sie liefern nur die Quellen über einen Webserver aus. Wären SPARQL-Anfragen

[1]Die LOD-Cloud, welche nur einen minimalen Ausschnitt darstellt und unter `http://www4.wiwiss.fu-berlin.de/lodcloud/state/` zu finden ist, umfasst mittlerweile ca. 31 Milliarden Einträge.
[2]`http://heise.de/-1429179`

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

möglich, wären Analyseanfragen à la Data Warehouse-Anfragen auch nur bedingt möglich.

Die zentralen Nachteile des Punktes 2 sind die (i.) benötigte Zeit zum Einsammeln aller Daten, die (ii.) Durchführung einer Vorberechnung und das Problem, dass niemals gewährleistet werden kann, dass (iii.) alle Daten aktuell sind. Zudem wird (iv.) ein SPOF, sowie ein (v.) extrem hoher Speicherplatz an einer einzigen zentralen Instanz in Kauf genommen. Ein weiterer Nachteil beider Fälle ist das (vi.) unbekannte Schema und die (vii.) geringe Datenqualität. Im zentralisierten Fall kann jedoch auf die letzten beiden Probleme effizient reagiert werden, indem entsprechende Indizes angelegt und/oder verschiedenste Optimierungen durchgeführt werden. Eine entsprechende Datenvorverarbeitung kann zudem davor gestartet werden, um die Datenqualität zu verbessern. Beides wird durch die einheitliche Form der Daten als reine Tripel unterstützt. Der große Vorteil, der sich aus dem zentralisierten Ansatz ergibt, ist die Möglichkeit, Anfragezeiten zu einem bestimmten Prozentsatz zu gewährleisten, da sich das gesamte System unter der Kontrolle einer Instanz befindet. Weiterhin kann ein Scale-Out (Erhöhung der Speicherkapazität (v.), Verbesserung der Antwortzeiten) und eine Replikation (Beseitigung SPOF (iv.), Verbesserung der Antwortzeiten) angestrebt werden. Zur Verringerung der Speicherkapazität (v.) können effiziente Kompressionstechniken verwendet werden, die zugleich lese- und schreiboptimiert sind, was zu einer weiteren Verringerung des benötigten Speicherplatzes führt. Durch die Verwendung von push und Bulk Load-Techniken zur Datenintegration anstelle von reinen pull-Techniken kann garantiert werden, dass die abgefragten Daten aktuell (iii.) sind. Das Einsammeln von Daten ist nur zu Beginn einmal nötig, wenn eine neue Datenquelle erschlossen wird (i. und ii.).

Der Mechanismus des Scale-Outs und der Replikation bedingen jedoch, dass eine Zerlegung der Anfrage durchgeführt werden muss. Wobei dieser eine Nachteil durch die oben genannten Vorteile aufgewogen wird. Insgesamt sprechen viele Faktoren für die Verwendung eines zentralisierten Ansatzes.

Garantien bezüglich den soeben erwähnten Antwortzeiten sind äußerst komplex in ihrer Umsetzung und benötigen im extremen Fall viele vorallokierte Kapazitäten, die während normaler Nutzung brachliegen und hohe Kosten verursachen. Stattdessen werden sie nur während Lastspitzen benötigt. Folglich werden aus diesem Grund häufig prozentuale Werte angegeben. Es sollen beispielsweiße 99,99 % aller Anfragen in der maximal erlaubten Zeitspanne beantwortet werden. Erst ein solches Vorgehen erlaubt es, dass während einer Lastspitze neue Kapazitäten hinzugenommen werden können (automatisierter Scale-Out). Das Ziel hierbei ist, zukünftige Anfragen wieder innerhalb der gesetzten Antwortzeit beantworten zu können. Am Ende der Lastspitze werden die zusätzlich allokierten Ressourcen wieder freigegeben. Letztendlich werden hierdurch Kosten für den Betreiber des Services eingespart. Wichtig zu erwähnen ist, dass die Garantie bezüglich der Antwortzeit nicht für jede beliebige SPARQL-Anfrage garantiert werden kann. Stattdessen soll dies nur für „gewöhnliche“ Anfragen gelten.

Projektziel.

Das Ziel dieses Projektes ist es, ein hochverfügbares, -skalierbares und -effizientes System aufzubauen, welches mehrere hundert Milliarden bzw. mehrere Billionen Tripel von RDF-Daten in einer materialisierten Form vorhält. Es soll hierbei als Framework und nicht als reine Datenablage angesehen werden. Neben SPARQL-Anfragen soll es den Benutzer im gesamten Workflow unterstützen. Unter Workflow werden die Folgenden Operationen verstanden: Einfügen/Löschen/Ändern von Tripeln sowie die Abfrage/Analyse von Daten. Unter dem Stichpunkt Analyse fallen sämtliche Data-Mining-Aufgaben, die in einem Data-Warehouse-Szenario möglich sind. Ein weiteres Ziel ist die Unterstützung von SLAs. Neben dem Garantieren von maximalen Antwortzeiten sollen auch Garantien bezüglich der Verfügbarkeit und der Aktualität von Daten, sowie einige andere mehr beachtete werden. Letztendlich soll das System als ein Cloud-Service für LOD-Daten propagiert werden. Das Ziel hierbei ist, Kosten für den Endbenutzer einzusparen. Er möchte nur für den von ihm selbst verursachten Aufwand bezahlen (Service on Demand). Im alternativen Fall müsste eine eigene Infrastruktur aufgebaut werden, was häufig zu viel höheren Kosten führt.

Diese Arbeit beschäftigt sich mit einem Teil aus dem soeben beschriebenen Zieles. Es soll ein erster Ansatz für einen Datenspeicher für Linked Open Data aufgezeigt werden, der mit sehr großen Datenmengen eine effiziente Anfrageverarbeitung ermöglicht. Hierfür ist ein zuverlässiges Scale-Out-Verfahren notwendig, welches vorgestellt wird. Des Weiteren soll das gewählte Verfahren auf eine wechselnde Anfragelast reagieren können.

2. STATE-OF-THE-ART

Für die Verarbeitung von LOD existieren bereits viele verschiedene Systeme, die RDF-Daten entgegennehmen und SPARQL-Queries ausführen. Einige bekanntere System sind Virtuoso [5], MonetDB [3], Hexastore [14], Jena [15] und RDF-3X [11]. Virtuoso ist ein weitverbreitetes relationales Datenbanksystem, das aus allen Kombinationen des Subjektes, Prädikates und Objektes Indizes erzeugt. Somit kann sehr schnell auf Daten zugegriffen werden. MonetDB ist ein OpenSource Column-Store, der eine Menge von Zwischenergebnissen im Speicher hält, um neue und ähnliche Anfragen schneller bearbeiten zu können. Hexastore ist eine In-Memory-Lösung. Es erzeugt aus allen möglichen Kombinationen von Tripeln Indizes, die daraufhin in einer Liste abgelegt werden. Für die Vermeidung von Dopplungen und zur Kompression werden Strings in einem Wörterbuch abgelegt. Jena ein bekanntes OpenSource-Projekt kann verschiedene Datenspeicher verwenden. Einerseits können die Daten in einem eigenen Format abgelegt werden, andererseits in einer relationalen Datenbank. Auch Jena verwendet ein Wörterbuch zur Kompression. RDF-3X verwendet auch mehrere Indizes, um alle möglichen Kombinationen abzuspeichern und legt diese in B+-Bäumen ab. Zudem enthält es eine ausgeklügelte Anfrageoptimierung, die speziell auf die Besonderheiten von RDF abgestimmt ist.

Werden alle vorgestellten Systeme genauer betrachtet, stellt man fest, dass diese einige Nachteile besitzen. Entweder sind diese auf bestimmte Operationen (lesen, ändern) und/oder Domänen getrimmt und/oder sind nicht skalierbar im Sinne eines Scale-Outs und somit ungeeignet für sehr große Datenmengen. Stattdessen setzten sie auf die Leistung einer einzigen Maschine, was zu einem erheblichen Performance-, Speicherplatz- und Verfügbarkeitsproblem werden kann. Weiterhin unterstützt keines der erwähnten Systeme Garantien bezüglich Antwortzeiten. Sie setzen zudem auf einen Permanentspeicher, welcher I/O-Zugriffe erfordert. Dies er-

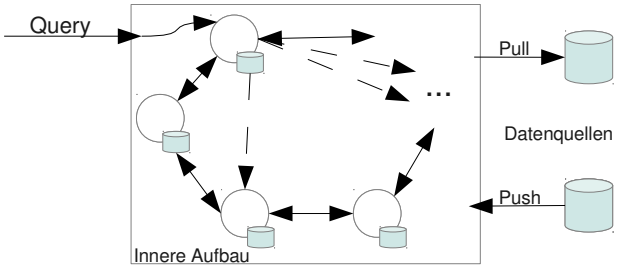


Figure 1: Aufbau des Systems mit mehreren unabhängigen Knoten, die untereinander verbunden sind. Annahme einer Query und Weitergabe an einem beliebigen Knoten. Einspielen von neuen Datenquellen durch Push und Pull.

höht wiederum sehr schnell die Antwortzeit. Eine Gewährung von Garantien wird erschwert. Einige dieser Ansätze sind zudem bereits mit wenigen Milliarden Tripeln überfordert (Jena). Andere verwenden alte Konzepte für neuere Probleme (Virtuoso), was nicht immer zu einer optimalen Lösung beiträgt.

Bezüglich den Garantien von Antwortzeiten, beschreiben die Autoren in [12] und [9] ein Verfahren zur Beantwortung von Anfragen in einem konstanten Zeitintervall. Dies wird durch verschiedenste Optimierungen erreicht. Unter anderem wird beschrieben, wie ein Row- und Column-Store miteinander verschmolzen wird und trotzdem eine effiziente Anfrageverarbeitung ermöglicht wird.

3. ARCHITEKTUR

Die Architektur des Gesamtsystems ist in Abbildung 1 aufgezeigt. Sie folgt dem typischen Scale-Out-Paradigma mit verschiedenen Shared-Nothing-Systemen (Kreise in der Mitte), die alle gleichberechtigt sind. Anfragen können von beliebigen Knoten entgegengenommen werden. Im nächsten Schritt wird ein Queryrewriting durchgeführt und an die entsprechenden Knoten weitergegeben. Die Empfänger nehmen die umgeschriebenen Anfragen entgegen, bearbeiten diese und senden letztendlich das Ergebnis zurück. Nachdem alle Einzelergebnisse eingetroffen sind werden diese mittels verschiedener Join-Operationen zu einem Ergebnis verknüpft. Dieses wird letztlich an den ursprünglich anfragenden Client als Ergebnis zurück gesandt. Auf der rechten Seite der Grafik ist ersichtlich, dass neue Daten mittels push- oder pull-Techniken eingespielt werden können und dies während der Laufzeit, mit Einhaltung der Antwortzeitgarantien.

3.1 Lokale Verarbeitung

Die lokale Verarbeitungseinheit liegt auf einem Knoten und besteht aus einem Java-Programm, sowie vorwiegend aus einer C++ In-Memory-Datenhaltung und Datenabfragemöglichkeit, die LODCache genannt wird. Dies ist eine hochperformante, reine In-Memory Lösung für LOD, die den L2-, wenn vorhanden L3-Cache und die RAM-Struktur moderner Hardware effizient ausnutzt. Allokierte Datenfelder, sogenannte Chunks, entsprechen genau der Größe des L2-Caches. Dies ermöglicht es sehr effizient auf Daten zuzugreifen, diese zu Bearbeiten und Berechnungen auf diesen durchzuführen. Des Weiteren können Anfragen an den LODCache gestellt, sowie Änderungen und Einfügeoperationen durchgeführt werden. Die Verwendung einer reinen In-Memory Lösung basiert auf der Grundlage, zukünftig Antwortzeiten garantieren zu können.

Linked Open Data bestehen ausschließlich aus Tripeln, die wiederum aus Strings bestehen. Werden die darin abgespeicherten Werte genauer untersucht, stellt man fest, dass dieselben Stringwerte mehrmals auftreten. Hier ist eine Kompression nützlich, um Speicherplatz einzusparen. Dies fällt umso mehr ins Gewicht, da der Data-Warehouse Ansatz angestrebt wird. Die naheliegenste Technik, ist der Einsatz eines *rein* lokalen Wörterbuches. In einem solchen wird jeder Stringwert auf eine eindeutige ID gemappt. Hierfür ist wiederum eine effiziente Anfrageverarbeitung von Nöten, die im folgenden Abschnitt beschrieben wird.

Kombinierte Row- und Column-Store.

Ein Chunk besteht intern aus einer Kombination von Row- und Column-Store [9]. Jede logische Zeile enthält drei Spalten für Subjekt, Prädikat und Objekt, wobei in einem jeden Feld nur ein einzelner Integerwerte abgelegt ist. Werden nun alle einzelnen Integerwerte eines Tripels mittels Shift-Operationen disjunkt zu einem Wert vereint, ergibt sich ein Wort mit 96 Bit Breite, wenn jeweils 32 Bit für Subjekt, Prädikat und Objekt angenommen werden. Somit wird die Datenzeile aus drei logischen Spalten zu einer physischen Spalte vereint, die in einem theoretischen Drittel der Zeit überprüft werden kann.

Moderne CPUs besitzen interne SIMD[3]-Register die eine Breite von 256-Bit besitzen. Diese wurde mit dem neuen AVX[4]-Befehlsatz von Intel eingeführt. In einem solchen Register werden mehrere logisch getrennte Einheiten zu einer physischen Einheit verbunden, die in einer Instruktion gleichzeitig bearbeitet werden können. Hierfür müssen die einzulesenden Daten auf eine volle 2-er Potenz ergänzt und die Wortgrenze bekannt sein.

Werden die Fakten vereint, muss zuerst das 96 Bit Wort auf 128 Bit, ergänzt werden. Mit Hilfe der verfügbaren 256-Bit können nun zwei logische Datenzeilen mit jeweils 128-Bit gleichzeitig überprüft werden. Theoretisch ergibt sich eine bis zu sechsfache Performancesteigerung. Mittels der Vermeidung von Sprung- und sonstigen Befehlen, die Leerlauf produzieren, ist dieses Verfahren sehr Cache freundlich. Einmal verwendete Daten können stur linear abgearbeitet werden. Zugriffe auf dem langsamen RAM werden damit vermieden. Insgesamt wird die Struktur moderner Hardware sehr gut ausgenutzt und eine Performancesteigerung ergibt sich.

Wie oben erwähnt, muss in diesem Fall eine Ergänzung von 96-Bit auf 128-Bit durchgeführt werden. Es verbleiben 32 ungenützte Bit. Diese können für eine weitere Spalte verwendet werden. In dieser kann die Sprache bzw. der Datentyp des (Literal)Objektes vermerkt werden [1]. Sollten entsprechende Filterbedingungen auftreten, können diese direkt mit übernommen werden, was zu einer weiteren Performancesteigerung beiträgt. Andererseits können die 32-Bit pro Spalte zu gering sein und entsprechend erweitert werden. Es ist auch eine Kombination aus beiden möglich (z. B. 40 Bit pro Spalte und 8 Bit für Sprache und Datentyp).

Durch die Verwendung von Integerwerten, die eine feste Bitbreite besitzen, wird auf einen expliziten und unvorher-

[3]Single instruction, multiple data
[4]Advanced Vector Extensions

sehbar langen Stringvergleich verzichtet. Dies erhöht weiter die Performance und verbessert die Antwortzeit.

Wie zu erkennen ist, wird auf eine Erzeugung von zusätzlichen Indixes verzichtet. Dies ergibt sich aus der Speicherrestriktion und dem nötigen Wartungsaufwand, der während jeder Änderung anfällt. Hierdurch können Operationen unvorhersehbar lange blockiert werden. Es sind keine Garantien mehr möglich. Anstelle werden wie in einem gewöhnlichen Column-Store alle Elemente stetig geprüft. Dies ist durch moderne Hardware und der ständig steigenden Parallelisierung problemlos möglich. Sind c Cores gegeben, muss jeder Core nur maximal $\lceil \frac{cC}{c} \rceil$ Elemente überprüfen, wobei cC die Anzahl der Chunks angibt.

Lokale Anfrageverarbeitung.

Der LODCache besitzt keinerlei Logik für die Optimierung von Anfragen. Aus diesem Grund ist eine vorgelagerte Java-Applikation vorhanden. Diese nimmt SPARQL-Anfragen entgegen, zerlegt diese, optimiert sie (lokale Optimierung) und überführt sie in die Sprache des LODCaches. Dieser führt die entsprechende Operation durch und übergibt das Ergebnis dem Java-Programm mittels JNI[5], dass die ermittelten Daten an den Aufrufer sendet.

Wird eine exakt Match Anfrage mit Subjekt$_1$, Objekt$_1$ und einem freien Prädikat `?p` an den LODCache gesandt, werden zuerst die Integerwerte der fest definierten Strings gesucht und in ein Wort mittels Shift und logischen OR-Operationen zu einer Maske (`m`) ergänzt. Dies wird mittels vier CPU-Operationen durchgeführt werden (COPY, SHIFT, OR, SHIFT). Die Operationen zum Filtern der Daten aus einem Chunk beschränken sich daraufhin nur noch auf ein logisches AND und einen Vergleich (CMP), je Eintrag und Chunk. Die genaue Berechnung ist im Folgenden nochmals dargestellt. Wobei `e` das zu überprüfende Element ist, `m` die Maske und `r` das Ergebnis als boolescher Wert.

```
r = (e AND m) CMP m
```

Werden Bereichsanfragen betrachtet, muss das Mapping der Strings auf Integerwerte eindeutig und ordnungserhaltend sein, da sonst immer das Wörterbuch zu Rate gezogen werden müsste. Dies bedeutet, jeder Integerwert muss mit der lexikografischen Ordnung seines gegenüberliegenden Strings übereinstimmen. Nur so können die in [9] aufgeführten Operationen angewandt und die Performance von SIMD-Befehlen ausgenutzt werden. Hierfür muss zuerst der minimale und maximale Integerwert der Bereichsanfrage ermittelt werden. Daraufhin werden zwei Masken erstellt und die Daten in den Chunks mit diesen, durch verschiedene logische Operationen, verglichen.

Ordnungserhaltender Baum.

Ein Problem tritt jedoch beim Einfügen von neuen Stringwerten auf. Diese werden meist lexikografisch zwischen zwei bereits bestehenden Elementen eingeordnet. Somit müssten sich in einem naiven Ansatz alle IDs aller nachfolgenden Stringwerte ändern. Das Wörterbuch und alle bestehenden Chunks müssen daraufhin überprüft und ggf. abgeändert werden.

Dies ist ein sehr großes und äußerst schwerwiegendes Problem, welches sich nicht vermeiden lässt. Es kann nur versucht werden, dass die Reorganisation möglichst selten und

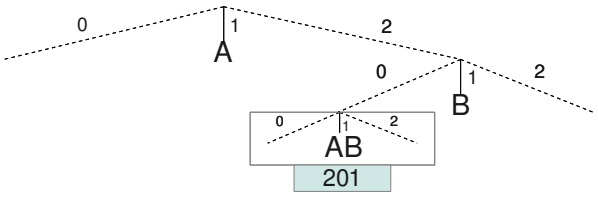


Figure 2: Aufbau des Baumes nach dem Einfügen von 'AB' (Rechteck), welches über den Pfad 201 erreichbar ist.

gleichzeitig effizient durchgeführt werden kann. Dies könnte naiv erreicht werden, indem jeder nachfolgende Mappingwert einige Zahlen zu seinem Vorgänger überspringt. Treten nun neue Einträge auf, können diese solange eingefügt werden, wie freie Plätze existieren. Ein Problem dieses Ansatzes ist jedoch, dass nicht bekannt ist, in welcher Reihenfolge Strings auftreten und es ist gänzlich unbekannt an welcher freien Position ein Element eingefügt werden muss/soll (in der Mitte, an der ersten oder letzten freien Position?). Eine falsche Position bedingt eine schnellere Reorganisation. Aus diesem Grund muss ein besser geeignetes Verfahren verwendet werden.

Ein solches wird für XML-Daten in [8] beschrieben. Die Autoren zeigen ein rekursives Verfahren, um Daten in einem Baum abzulegen und eine eindeutige ID zu generieren. Hierfür wird auf jeder Ebene jede mögliche Verzweigung durchnummeriert[6]. Nun wird zwischen geraden und ungeraden Elementen unterschieden. Die ungeraden Elemente sind vorhanden, um darin Werte sortiert aufzunehmen (durchgezogene Linie). Im Gegensatz zu den Ungeraden, denn diese spannen einen neuen Unterbaum auf (gestrichelte Linie). Soll ein neuer Wert eingefügt werden, wird dieser auf der höchsten Ebene (sortiert) hinzugefügt, die (i.) mindestens einen freien Platz besitzt und (ii.) lexikografisch den String korrekt einordnet. Das neue Element kann nun über die Konkatenation aller traversierten Baumverzweigungen eindeutig identifiziert werden und wird sinnvollerweise in einem String überführt. In Abbildung 2 ist ein Beispiel mit den Elementen 'A', 'B' und dem neuen Element 'AB' gegeben. Durch diesen Ansatz ist gewährleistet, dass sich neue Elemente immer zwischen zwei bereits bestehenden Elementen einsortieren lassen. Problematisch an diesem Ansatz ist jedoch, dass beliebig viele Unterbäume generiert werden können, der Baum zu einer Liste entartet und somit die Pfadlänge zu einem Element sehr lang werden kann. Dies widerspricht jedoch der oben genannten Forderung, nach einer festen Breite für Elemente, sowie der Verwendung eines Integerwertes. Aus diesem Grund muss eine Modifikation dieses Ansatzes verwendet werden.

Anstelle eines Strings, um den Pfad eindeutig zu identifizieren, werden nun beispielsweise 32 Bit Werte verwendet. Im Folgenden werden weiterhin jeweils 2 Bit pro Ebene verwendet. Wobei für jede neue Ebene, die neu hinzukommenden Bits an den bereits bestehenden Bits dahinter gehangen werden (von der größten zur kleinsten Wertigkeit). Es ergeben sich insgesamt maximal 16 Ebenen. Von den möglichen vier ($2^b = 2^2$) Werten pro Ebene können nur drei verwendet werden (`00`, `01`, `10`). Dies ergibt sich daraus, da der Wert `11` ungerade ist. In einem solchen müsste nach der

[5] Java Native Interface

[6] In den weiteren Ausführungen wird von 0 gestartet.

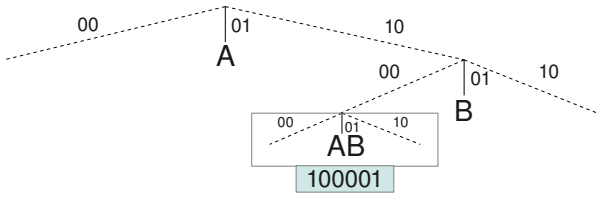


Figure 3: Aufbau des Baumes nach dem Einfügen von 'AB' (Rechteck), welches über den Pfad 100001 erreichbar ist.

oben genannten Definition ein Element ablegt werden. Jedoch existiert nach diesem kein weiteres gerades Element, womit nach einem falsch einsortierten Element kein weiteres lexikografisch dahinter liegendes Element eingefügt werden kann. Dies würde eine frühere Reorganisation erzwingen, was zu vermeiden ist. In Abbildung 3 ist das bereits weiter oben gezeigte Beispiel nochmals auf diesen Sachverhalt abgebildet.

Bewertung.

Ein großes Problem dieses Ansatzes ist jedoch die geringe Auslastung. So werden höchstens 50 % der möglichen Werte verwendet und auch nur genau dann, wenn keine zusätzlichen Ebenen vorhanden sind. Somit treten wiederum sehr schnelle Reorganisationen auf. Auf der anderen Seite kann eine Reorganisation sehr schnell durchgeführt werden. Sie ist auf wenige logische Operationen pro Wörterbuch- und Chunk-Eintrag beschränkt (SHIFT- und OR-Operationen). Gleichzeitig kann durch eine Reorganisation die Ebenenanzahl verringert und somit die Anzahl an Bits pro Ebene vergrößert werden. Die theoretisch mögliche Auslastungsgrenze steigt automatisch an.

3.2 Globale Verarbeitung

Die globale Architektur besteht aus der Zusammenfassung aller lokalen Verarbeitungseinheiten zu einer globalen Einheit. Diese sind mittels der Technik eines Chord-Ringes [13] miteinander verbunden. Er besteht aus n unabhängigen und gleichwertigen Elementen, die in einem geschlossenen Ring angeordnet sind. Wobei jedes Element einen Vorgänger und einen Nachfolger besitzt (siehe Abbildung 1; Doppeltpfeile mit durchgehender Linie). Neben diesen Verbindungen existieren noch sogenannte Finger (siehe Abbildung 1; einfacher Pfeil mit gestrichelter Linie). Dies sind zusätzliche unidirektionale Verbindungen, die mehrere Nachfolger überspringen und somit für eine schnellere Kontaktaufnahme mit einem beliebigen Element vorhanden sind. Wären diese nicht vorhanden, müsste eine Anfrage von einem Element zum anderen solange weitergereicht werden, bis der Empfänger erreicht ist ($O(n)$). Dies wird durch die Finger auf $O(\log_2 n)$ verkürzt.

Fragmentierungsfunktion.

Für die Umsetzung eines Scale-Outs wird eine Fragmentierungs- und Allokationsfunktion benötigt. Erstere definiert wie welche Daten aufgeteilt werden, letztere auf welchen Knoten welches Element abgelegt wird. Beide sollten möglichst einfach zu berechnen sein, da sie für jede Anfrage benötigt werden (Datenlokalisation). Der Chord-Ring definiert hier bereits eine Hashfunktion $h(x)$. Diese erzeugt durch die Eingabe von Daten einen Hashwert, der genau einem Knoten im Chord-Ring zugeordnet ist. Eine einfache Datenlokalisationsfunktion ist hierdurch gegeben. Der Vorteil, der hieraus entsteht ist ein globaler und flüchtiger Index, der keinerlei Wartung benötigt und auf allen n Elementen gleichermaßen zur Verfügung steht.

Für die Indizierung von RDF-Tripeln werden alle einstelligen Kombinationen aus Subjekt, Prädikat und Objekt erzeugt (s→op, p→so, o→sp). Diese werden nun mittels $h(x)$ auf den Ring verteilt, indem das zu indizierende Element in $h(x)$ gegeben wird und der entsprechende Knoten bestimmt wird. Für dasselbe Literal ergibt sich immer derselbe Hash und somit ist immer derselbe Knoten zuständig.

Globale Anfrageverarbeitung.

Im ersten Schritt wird eine beliebige SPARQL-Anfrage einem Client an einem beliebigen Knotenelement gesandt. Der Empfänger wird zum Koordinator dieser Anfrage. Nun zerlegt er die SPARQL-Anfrage und überprüft, ob er alle Daten lokal besitzt. Ist dies der Fall, liest er die Daten aus und führt die gesamte Berechnung durch. Ansonsten leitet er die umgeschriebenen Subanfragen an die zuständigen Knoten weiter. Hierfür wird die Hashfunktion $h(x)$ benötigt, in dem der nicht variable Teil einer jeden WHERE-Bedingung eingetragen wird. Während des Rewriting-Vorganges werden FILTER-Statements beachtet und in die Subqueries einbezogen, falls dies möglich ist (globale Optimierung). Durch diese Technik arbeiten mehrere Knoten parallel an derselben Ausgangsquery. Als Nebeneffekt wird die Datenmenge verkleinert. Nach dem Erhalt der Subqueries arbeiten die Knoten diese ab (lokale Anfrageverarbeitung; siehe Kapitel 3.1) und senden das Ergebnis an den Koordinator zurück. Dieser führt die benötigten Joins und die restlichen FILTER-, GROUP BY-, HAVING-, ORDER BY-, LIMIT-, OFFSET-, Projektions-, usw. Operationen durch. Zum Schluss wird das Ergebnis an den Client gesandt.

Bewertung.

Ein Problem dieses Ansatzes ist die mehrfache redundante Datenhaltung derselben Daten in maximal drei verschiedenen Knoten und die dafür zweimal höhere Speicherkapazität. Dies ist jedoch nur bedingt ein Nachteil. Durch den automatisierten Scale-Out-Mechanismus kann ein lokales Element entlastet werden, indem ein neuer Knoten einen Teilbereich der Hashwerte und die darin abgebildeten Daten für sich beansprucht. Ein weiterer Vorteil ist die Möglichkeit auf sehr große Anfragelasten dynamisch reagieren zu können. Es können automatisch neue Knoten hinzugefügt werden, die einen Teil der Anfragelast übernehmen. Die Aufteilung des Speicherplatzes und Anfragelast, wird jeweils durch die konsistente Hash-Funktion $h(x)$ garantiert [10]. Durch den beschriebenen Scale-Out, wächst die maximale Pfadlänge nur bedingt an, da diese durch $O(\log_2 n)$ definiert ist (was einem Vorteil entspricht). Ein weiterer Vorteil ist, dass zu jeder WHERE-Bedingung maximal ein Knoten involviert ist. Es müssen keine knotenübergreifenden Daten für bspw. Bereichsanfrage gesammelt werden. Dies ist nur zwischen verschiedenen WHERE-Bedingungen nötig, die mittels Join verknüpft werden.

4. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Ansatz vorgestellt, der mehre-

re Hundert Milliarden bzw. mehrere Billionen RDF-Tripel Linked Open Data effizient verwalten kann. Es wurde eine Unterscheidung in lokaler und globaler Verarbeitung durchgeführt. Die lokale Verarbeitungseinheit besteht aus einem hochoptimierten In-Memory C++-Programm zur Datenhaltung und -abfrage, das die Strukturen moderner Hardware effizient ausnutzt. Im selben Abschnitt wurde eine Möglichkeit aufgezeigt, Bereichsanfragen effizient zu verarbeiten, indem ein ordnungserhaltendes Mapping von Strings auf Integerwerten dargestellt wurde. Dies wird durch einen ordnungserhaltenden und updatefreundlichen Baum garantiert.

Die globale Verarbeitungseinheit besteht aus dem Zusammenschluss mehrerer lokaler Komponenten und verwendet hierzu die Technik des Chord-Ringes. Jedes Element ist gleichberechtigt, es existiert somit kein zentraler Koordinator. Für die Verbindung untereinander existieren Finger, die eine maximale Anzahl an Weiterleitungen garantieren. Zur Verarbeitung von beliebigen SPARQL-Anfragen werden diese von einem Knoten entgegengenommen, optimiert und an den betreffenden Knoten gesandt. Zur Datenlokalisation wird eine einfache Hashfunktion und kein global zu pflegender Index benötig. Des Weiteren ist dieses Verfahren unabhängig gegenüber der Anzahl an Tripeln und dem benötigten Speicherplatz, da ein automatischer Scale-Out-Mechanismus existiert.

5. AUSBLICK

In der weiteren Forschung müssen einige Punkte näher betrachtet werden, die als Motivation zu diesem Ansatz dienen. Hierunter fällt die Einhaltung der garantierten Antwortzeit aller Anfragen bis zu einem vorher definierten Prozentsatz. Zur Unterstützung dieser Forderung ist eine Replikation der Daten denkbar.

Dies führt zum nächsten Problem, der effizienten Replikation. Bis zu diesem Zeitpunkt werden alle Daten nur auf einem Knoten vorgehalten. Stürzt dieser ab, sind all seine Daten verloren und nachfolgende Anfragen können nicht mehr beantwortet werden. Als Ausweg bestünde die Möglichkeit, ein ähnliches Vorgehen umzusetzen, wie es in Amazons Dynamo [4] implementiert ist.

Zur weiteren Performancesteigerung ist es notwendig den LODCache weiter zu optimieren. Es existiert zum Beispiel die Möglichkeit auf einer SandyBridge-CPU eine Schleife direkt im CPU eigenen Loop-Cache abzulegen. Eine Performancesteigerung von über 100 % soll möglich sein. Jedoch ist die Bedingung hierfür, dass alle Instruktionen höchstens 28 μ-Operationen lang sind.

Der oben beschriebene Ansatz zum Mapping von Strings auf ordnungserhaltende Integerwerte muss weiter erforscht werden. Es ist u. a. notwendig, die Operationen zur Reorganisation effizienter anzuordnen. Eine Möglichkeit zur Erhöhung der Auslastung wird außerdem angestrebt. In diesem Zusammenhang soll gleichzeitig ein Verfahren entwickelt werden, um Updates auf bestehende Daten möglichst effizient durchzuführen.

Für die weitere Entwicklung und Evaluierung des Systems wird zukünftig ein Benchmark eingesetzt. Es wurde der Berlin SPARQL Benchmark [6] ausgewählt. Dieser erzeugt akzeptierte Resultate und ist für das Testen von beliebigen SPARQL-Systemen entwickelt worden. Für Skalierungstests kann ein Skalierungsfaktor angepasst werden, der die Anzahl an automatisch erzeugten Tripeln vergrößert.

Die Performanz des Systems wird durch verschiedenartige SPARQL-Anfragen ermittelt, die sich in Äquivalenzklassen einteilen lassen. Den einfachen SPARQL-Anfragen, den Anfragen, die eine Datenmanipulation erfordern sowie den analytischen SPARQL-Anfragen.

6. REFERENCES

[1] Rdf - semantic web standards. *http://www.w3.org/RDF*.

[2] Sparql query language for rdf. *http://www.w3.org/TR/rdf-sparql-query*.

[3] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[5] O. Erling and I. Mikhailov. RDF support in the virtuoso DBMS. In S. Auer, C. Bizer, C. Müller, and A. V. Zhdanova, editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[6] FU-Berlin. Berlin sparql benchmark. *http://www4.wiwiss.fu-berlin.de/bizer/berlinsparqlbenchmark*.

[7] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, pages 411–420, 2010.

[8] M. P. Haustein, T. Härder, C. Mathis, and M. W. 0002. Deweyids - the key to fine-grained management of xml documents. *JIDM*, 1(1):147–160, 2010.

[9] R. Johnson, V. Raman, R. Sidle, and G. Swart. Row-wise parallel predicate evaluation. *PVLDB*, 1(1):622–634, 2008.

[10] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In F. T. Leighton and P. W. Shor, editors, *STOC*, pages 654–663. ACM, 1997.

[11] T. Neumann and G. Weikum. RDF-3X: A RISC-Style Engine for RDF. In *VLDB*, Auckland, New Zealand, 2008.

[12] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-time query processing. In G. Alonso, J. A. Blakeley, and A. L. P. Chen, editors, *ICDE*, pages 60–69. IEEE, 2008.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

[14] C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *VLDB*, Auckland, New Zealand, 2008.

[15] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *Proc. First International Workshop on Semantic Web and Databases*, 2003.

SLO-basiertes Management in relationalen Datenbanksystemen mit nativer Multi-Tenancy-Unterstützung

Andreas Göbel
Lehrstuhl für Datenbanken und Informationssysteme
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, 07743 Jena
andreas.goebel@uni-jena.de

KURZFASSUNG

Das an Bedeutung und Akzeptanz gewinnende Geschäftsmodell Software as a Service ermöglicht Unternehmen die Konzentration auf ihr Kerngeschäft durch das Beziehen von Dienstleistungen über das Internet. Die Festlegung von Service Level Agreements gewährleistet eine hohe Qualität des Dienstes. Um die operationalen Kosten des Service Providers gering zu halten, bedarf es des Einsatzes einer Multi-Tenancy-Architektur, die zu neuen Herausforderungen für den Einsatz von Datenbankverwaltungssystemen führt.

In diesem Beitrag werden die Problemstellungen der Realisierung von Multi Tenancy in heutigen Datenbankverwaltungssystemen aufgezeigt und eine native Unterstützung von Multi Tenancy in jenen Systemen motiviert. Es wird hervorgehoben, dass die Integration von mandantenspezifischen Service Level Agreements zur Steigerung der Qualität des Dienstes beiträgt. Hierzu wird die Verwendung dieser bereitgestellten Daten zur Ressourcenverwaltung und -überwachung sowie der Lastverteilung von Mandanten verdeutlicht.

Kategorien und Themenbeschreibung

H.2.7 [**Database Management**]: Database Administration—*Data dictionary/directory*; H.2.1 [**Database Management**]: Logical Design—*Schema and subschema*

Allgemeine Begriffe

Design, Management, Measurement

Stichworte

Relational Database, Multi Tenancy, Service Level Agreement

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

1. EINLEITUNG

Software as a Service (SaaS) bezeichnet ein Geschäftsmodell, bei dem Unternehmen von einem Drittanbieter Anwendungen in Form von Services über das Internet beziehen. Der Ansatz steht dem herkömmlichen On-Premise-Modell gegenüber, bei dem Unternehmen die Lizenzen für Software käuflich erwerben und die Anwendungen anschließend auf eigener Hardware betreiben. Durch das Betreiben und Warten der Anwendung sowie der notwendigen Hardware durch den Drittanbieter (SaaS-Anbieter) können die als Mandanten bezeichneten Unternehmen auf den Erwerb und die Wartung der Hardware verzichten. Service Level Agreements (SLAs) legen dabei als Bestandteil der Dienstleistungsverträge durch die Definition verschiedener Service Level Objects (SLOs) die Qualität der Services fest.

Laut Studien renommierter Marktanalyseunternehmen wird die Bedeutung von SaaS in den nächsten Jahren weiter zunehmen. So prognostiziert Pierre Audoin Consultants SaaS für das Jahr 2015 einen Umsatzanteil am Software-Markt in Höhe von zehn Prozent und begründet die zunehmende Bedeutung u.a. mit entsprechenden Anpassungen der Produktportfolios sowie Akquisitionen von bedeutenden Software-Anbietern wie Oracle, SAP und Microsoft [18].

SaaS-Angebote sind aktuell insbesondere in den Bereichen E-Commerce, Kundenbeziehungsmanagement und bei kollaborativen Aufgaben wie E-Mail zu finden. Zunehmend werden Produkte in den Bereichen Humankapital-Management und Unternehmensressourcenplanung angeboten. Um einen möglichst großen Markt ansprechen zu können, ermöglichen SaaS-Anbieter eine individuelle Anpassung des Services. Die Angebote richten sich auch an kleine und mittlere Unternehmen (KMU), denen aufgrund begrenzter finanzieller Möglichkeiten die Mittel für den Erwerb und Support einer vergleichbaren On-Premise-Lösung fehlen. Dieser so genannte Long Tail [3] kann durch Preisvorteile gegenüber On-Premise-Angeboten und entfallende Kosten der Mandanten für die Beschaffung und Wartung von Hardware bedient werden [10]. Es bedarf hierzu einer hohen Wirtschaftlichkeit des Services durch eine Konsolidierung von Mandanten auf physischen Ressourcen in Form von Multi-Tenancy-Anwendungen.

2. MULTI TENANCY

Die Architektur von SaaS-Angeboten ist in großem Maße vom Geschäftsmodell des Anbieters abhängig. Sie wird beispielsweise beeinflusst durch das zur Verfügung stehende

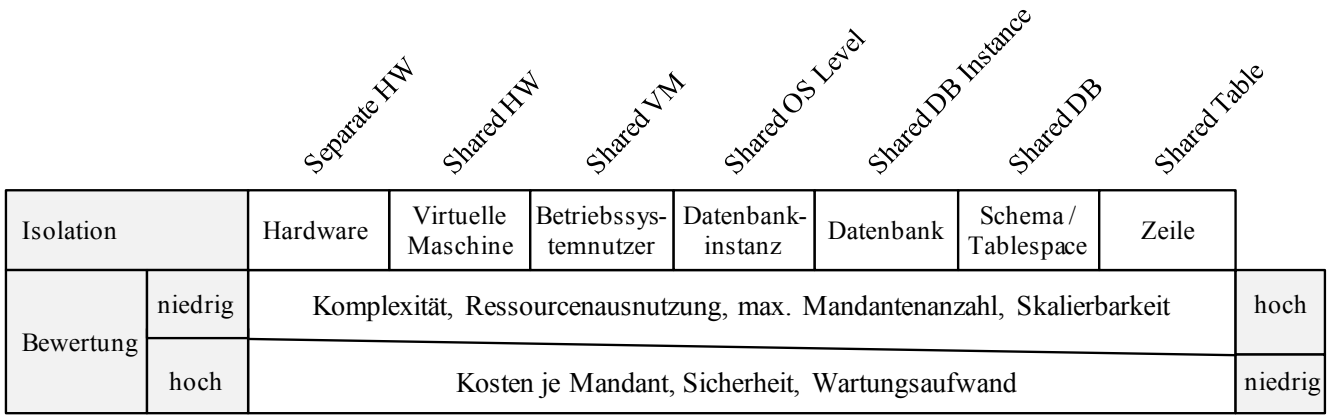


Abbildung 1: Ansätze zur Mandantenisolation im DB-Layer, angelehnt an [19]

Entwicklungskapital, den Zielmarkt sowie die Anzahl und Charakteristika der Mandanten. Zu den Charakteristika gehören u.a. die benötigte Datenmenge, die voraussichtliche Workload und die Anforderungen der Mandanten bezüglich Verfügbarkeit, Performance, Datenisolation und Individualisierung der Anwendung.

Um die in der Einleitung motivierte hohe Wirtschaftlichkeit eines SaaS-Angebots zu erzielen, werden vermehrt Multi-Tenancy-Architekturen eingesetzt. Diese erlauben allen Service-Nutzern die gemeinsame Verwendung von Hardware-Ressourcen durch das Anbieten **einer** gemeinsamen Anwendungsinstanz [9]. Sowohl auf der Applikations- als auch der Datenbankseite sind verschiedene Ansätze zur Separierung von Mandanten denkbar.

2.1 Multi Tenancy in DBMS

Es existiert ein breites Realisierungspektrum für Multi Tenancy in einem Datenbankserver. In Abbildung 1 werden die wichtigsten Ansätze zusammengefasst sowie Vorteile und Herausforderungen aufgezeigt. Neben der Möglichkeit, auf eine Mandantenkonsolidierung zu verzichten (*Separate Hardware*), können Mandanten beispielsweise durch die Zuweisung separater virtueller Maschinen (*Shared HW*), durch die Nutzerverwaltung des Betriebssystems (*Shared VM*) oder durch die Verwendung getrennter Datenbankinstanzen (*Shared OS Level*) bzw. Datenbanken (*Shared DB Instance*) voneinander isoliert und dennoch auf einem Datenbankserver verwaltet werden. Aufgrund des initialen Ressourcenbedarfs und der gesonderten Administration dedizierter Datenbanken, Datenbankinstanzen, Betriebssysteme oder virtueller Maschinen führen diese Ansätze für den SasS-Anbieter zu hohen Kosten je Mandant. Sie sollten daher nur bei zwingender Notwendigkeit eines hohen Isolations- bzw. Sicherheitslevels oder bei einer geringen Anzahl von Mandanten verwendet werden.

Der Ansatz *Shared Database* erlaubt durch die Mandantenkonsolidierung in einer Datenbank die gemeinsame Nutzung von Datenbankprozessen und Hauptspeicherinhalten. Die Zuweisung zu eigenen Datenbankobjekten wie Tabellen und Indizes führt bei Nutzung der Datenbankzugriffskontrollen zu einer logischen Isolation der Mandanten. Durch die Zuweisung der mandantenspezifischen Objekte zu separaten Speicherorten (Tablespaces) kann zudem eine physische Trennung erreicht werden. Das Hinzufügen und Löschen von Mandanten sowie mandantenspezifische Schemaänderungen bedürfen bei diesem Ansatz das Absetzen von DDL-Statements, die bei einigen DBMS zu Problemen mit dem fortlaufenden Betrieb führen können. Die hohe Anzahl an Tabellen führt zudem zu einem hohen Hauptspeicherbedarf sowie partiell gefüllten Seiten des Datenbankpuffers. [6, 12]

Bei dem Ansatz *Shared Table* werden Objekte der Datenbank von Mandanten gemeinsam genutzt. Tabellen enthalten somit Tupel verschiedener Mandanten, weshalb die Verwendung einer zeilenbasierten Zugriffskontrolle nötig ist. Eine zusätzliche Tabellenspalte legt hierbei die Zugehörigkeit des Tupels zum entsprechenden Mandanten fest. Durch den Verzicht auf mandantenspezifische Datenbankobjekte ist die Größe des Datenbankkatalogs nahezu unabhängig von der Mandantenanzahl. Die maximale Anzahl unterstützter Mandanten ist somit laut [12] lediglich durch die maximale Anzahl unterstützter Tabellenzeilen beschränkt, was den Ansatz für das Bedienen des im Abschnitt 1 angesprochenen Long Tails prädestiniert. Durch die hohe Ressourcenausnutzung reduziert sich der Overhead bezüglich Fest- und Hauptspeicherbedarf pro Mandant auf ein Minimum. Administrative Operationen und Updates der Anwendung werden bei diesem Ansatz in der Regel für alle Mandanten ausgeführt, was den Wartungsaufwand des Anbieters reduziert, die Individualität des Services jedoch einschränkt. So können die in [11] angesprochenen individuellen Anforderungen von Mandanten in Bezug auf Aspekte der Datenbankadministration und -konfiguration wie Backup-Strategien und Archivierungsintervalle, die Replikationsart und Replikatanzahl oder Vorgaben zur Arbeitsweise des Datenbankoptimierers nicht erfüllt werden. Aufgrund der Konsolidierung von vielen Mandantendaten innerhalb einer Tabelle liegen die größten Herausforderungen dieses Ansatzes in der Gewährleistung der Isolation der Mandantendaten und -Performance sowie der Erarbeitung eines Datenbankschemas, welches den Mandanten die Anpassung der vom SaaS-Anbieter zur Verfügung gestellten Anwendungen erlaubt.

2.2 Schemaflexibilität

Um seinen Service einer möglichst breiten Zielgruppe anbieten zu können, sind SaaS-Anbieter bemüht, Mandanten weitreichende Anpassungsmöglichkeiten zu bieten. Diese umfassen laut [10] unter anderem eine Anpassung der Benutzer-

oberfläche im Sinne eines Corporate Identity, die Anpassung an Geschäftsabläufe durch Modifikation von Geschäftsregeln, individuelle Regelungen bezüglich der Zugangskontrollen sowie die Möglichkeit zur Erweiterung des Datenbankschemas durch zusätzliche Tabellenspalten oder komplette Tabellen. Diese Anforderung stellt sich aufgrund des notwendigen Zusammenführens individueller Datenbankschemata der Mandanten auf ein Gesamtschema der Datenbank insbesondere beim Ansatz *Shared Table* als Herausforderung dar. Aulbach et al. stellen in [4, 5] eine Reihe von Ansätzen vor, die in folgende Kategorien eingeteilt werden können:

Vertikale Speicherung: Dieser Ansatz basiert auf dem Entity-Attribute-Value-Modell [17], welches beispielsweise im medizinischen Bereich Anwendung findet. Jeder Attributwert eines mandantenspezifischen Datensatzes wird auf einen Datensatz des Gesamtschemas abgebildet, der zur Identifizierung neben dem Attributwert den zugehörigen Mandanten-, Tabellen- und Spaltenname sowie die Zeilennummer enthält. Ein mandantenspezifischer Datensatz muss hierbei zur Laufzeit durch Verbundoperationen erzeugt werden, um die Attributwerte zu einem Datensatz zusammenzufügen.

Horizontale Speicherung: Ein mandantenspezifischer Datensatz wird direkt auf Datensätze des Gesamtschemas abgebildet. Flexibilität kann beispielsweise durch die Nutzung einer universellen Tabelle geboten werden, welche durch eine Vielzahl von generischen Spalten mit flexiblen Datentypen die Speicherung beliebiger Datensätze erlaubt und die Verwaltung des Schemas in die Anwendung verschiebt.

XML: Heutige Datenbankmanagementsysteme bieten zunehmend native Unterstützung des XML-Datentyps zur Speicherung semistrukturierter Daten. Durch dessen Verwendung können verschiedene mandantenspezifische Schemata innerhalb einer Tabelle verwaltet werden.

Hybride Speicherung: Die vorherigen Speicherformen können miteinander verbunden werden, um ihre Stärken zu kombinieren.

2.3 Native Unterstützung durch DBMS

Die in Abschnitt 2.2 vorgestellten Ansätze zur Unterstützung individueller Mandantenschemata können bei heutigen Datenbanksystemen nur mit Hilfe einer überhalb des DBMS liegenden Schicht zur Transformation der mandantenspezifischen Datenbankanfragen und vom DBMS erhaltenen Ergebnisse realisiert werden. Diese Schicht regelt die Zugriffskontrolle und verwaltet die Schemata der Mandanten, sodass die Schemainformationen vom DBMS nicht zur Optimierung genutzt werden können. Zudem führt sie zu erhöhten Wartungsaufwand und beeinflusst unter Umständen die Skalierbarkeit des Systems. [6, 20]

Bisherige Konzepte und prototypische Implementierungen [6, 20] zur nativen Unterstützung von Mandanten in relationalen Datenbanksystemen verlagern im Wesentlichen die Transformationsschicht inklusive der benötigten Metadaten ins Datenbanksystem. Sie verfolgen hiermit das Ziel einer effizienten Mandantenkonsolidierung sowie der Abbildung mandantenspezifischer Schemata auf ein Gesamtschema, um die oben aufgezeigten Nachteile einer externen Transformation anzugehen. Des Weiteren wird in [7] ein Konzept vorgestellt, welches Mandanten durch die Unterstützung mehrerer paralleler Basisschema-Versionen einen verzögerten Wechsel auf eine neue Version der SaaS-Applikation und dessen Erweiterungen erlaubt.

3. SLO-BASIERTE VERWALTUNG

Ein bisher kaum betrachtetes, jedoch nicht minder bedeutsames Forschungsgebiet im Zusammenhang mit der nativen Unterstützung von Multi Tenancy in Datenbanksystemen ist die Integration von abgeleiteten Richtlinien aus den Service Level Agreements (SLAs) der Mandanten sowie die Verwaltung der Mandantendaten auf Basis jener Richtlinien. Durch die zentrale Haltung der Richtlinien als Bestandteil des Datenbankkatalogs können sie von verschiedenen DBMS-Komponenten und externen Werkzeugen zur Verbesserung der Dienstqualität verwendet werden.

SLO-basiertes Management kann mittels automatisiertem und proaktivem Agieren den Administrationsaufwand für den SaaS-Anbieter reduzieren. Zudem kann es die Lastverteilung und Migration von Mandanten unterstützen, um Mandanten stets die Ressourcen zur Verfügung zu stellen, die ihren Anforderungen genügen. Die Priorisierung von Mandanten bei der Verarbeitung ihrer Systemanfragen kann durch eine SLO-basierte Ressourcenverwaltung und -überwachung realisiert werden. Das SLO-basierte Management ist somit ein essentielles Mittel, um auf der einen Seite die Betriebskosten der SaaS-Anbieter durch eine hohe Ressourcenausnutzung gering zu halten und auf der anderen Seite den Mandanten eine möglichst hohe Service-Qualität zu bieten.

Die Einsetzbarkeit in verschiedenen Aufgabenfeldern sowie die damit verbundenen Vorzüge für Administratoren und Mandanten begründen intensive Forschung in folgenden Themengebieten:

- Ableitung von geeigneten mandantenspezifischen Richtlinien aus Dienstleistungsverträgen,
- Repräsentation der Richtlinien im Katalog des Datenbankverwaltungssystems,
- Omnipräsentes Monitoring der Einhaltung von Richtlinien,
- Entwicklung und Realisierung geeigneter Algorithmen zur Sicherstellung der Richtlinien.

3.1 Service Level Objects

Als Bestandteil des Dienstleistungsvertrags zwischen dem SaaS-Anbieter und den Mandanten legen Service Level Agreements die zugesicherte Qualität des SaaS-Angebots fest. Hierzu spezifizieren sie beispielsweise Kennzahlen oder Abstufungen bezüglich der folgenden Anforderungen an den Services in Form von Richtlinien, welche als Service Level Objects (SLOs) bezeichnet werden.

- Verfügbarkeit: Systemzugänglichkeit, Wartungsfenster, Wiederherstellungszeiten in Fehlerfällen
- Performance: Geschwindigkeit der Datenverarbeitung, Reaktionszeiten der Schnittstellen
- Sicherheit: Datenschutz, Datensicherheit, Art der Isolation von anderen Mandanten

Die SLOs sollten u.a. aussagekräftig, erreichbar, messbar und verständlich sein [22]. Bestandteil der SLAs sind neben den

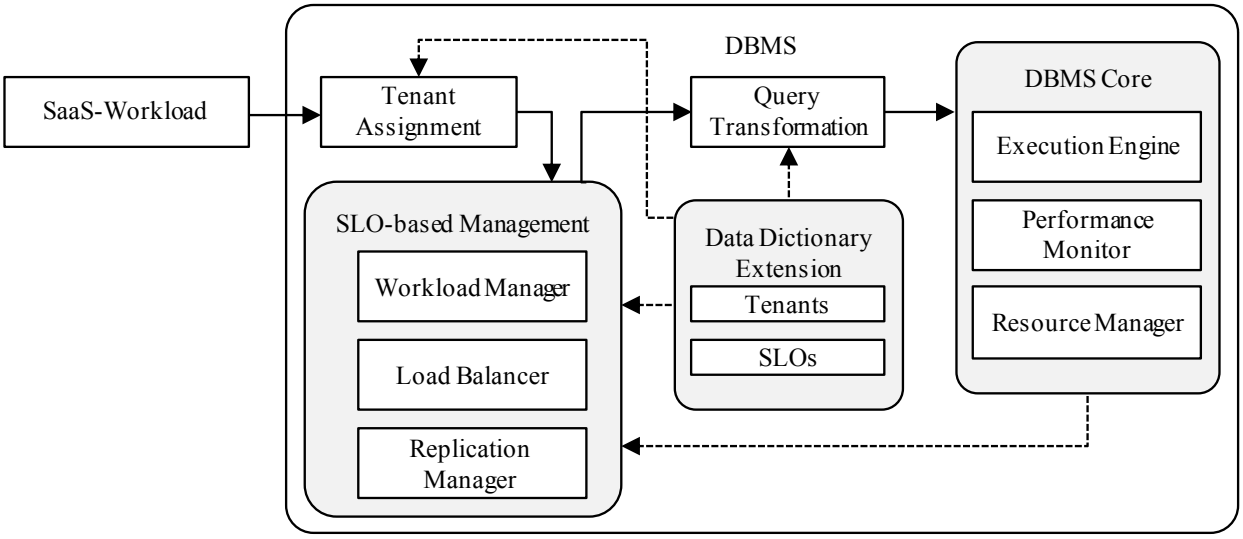


Abbildung 2: Integration von SLO-basiertem Management ins DBMS

SLOs entsprechende Regelungen für den Fall, dass der SaaS-Anbieter die zugesicherten SLOs nicht erfüllen kann. In der Regel erfolgt dies über gestaffelte Strafzahlungen oder lediglich über die Verminderung der anfallenden Grundgebühren für Mandanten.

3.2 SLO-Repräsentation

Nach der Einigung der SaaS-Anbieter und Mandanten über zu gewährleistende SLOs und dem Vertragsabschluss erfolgt die Überführung der finalen Bestimmungen in Anforderungen an die zugrunde liegende Technologie und Hardware des Anbieters. SLOs werden vorrangig technologieunabhängig definiert und gelten in der Folge für den gesamten Service, weshalb eine adäquate Ableitung von mandantenspezifischen Richtlinien für die einzelnen Systemkomponenten wie dem Datenbanksystem eine bedeutsame und komplexe Aufgabe darstellt. Zudem verdeutlicht dieser Sachverhalt, dass die Einhaltung jener Richtlinien, analog zu Multi Tenancy, in allen Schichten des Gesamtsystems von Bedeutung ist. Mit zunehmender Mandantenkonsolidierung auf den zur Verfügung stehenden Systemressourcen nimmt die Beeinflussung unter Mandanten bezüglich der Performance zu, was die Erstellung passender Richtlinien weiter erschwert.

Die resultierenden Richtlinien sollten entsprechend einem adäquaten Modell erstellt werden. Sie bestehen aus einer Kombination aus Anforderungen, beispielsweise bezüglich der Performance, Verfügbarkeit oder Sicherheit sowie einer Priorität. Diese spiegelt die Bedeutsamkeit des Mandanten für das Unternehmen wider und basiert typischerweise auf der Größenordnung der entsprechenden Vertragsstrafen eines Mandanten [14]. Unter Umständen können hierbei jedoch Aspekte eine Rolle spielen, die nicht direkt aus dem Dienstleistungsvertrag abgeleitet werden können wie die Reputation des Mandanten oder seine Bedeutung als strategischer Partner des SaaS-Anbieters.

Die native Unterstützung von Multi Tenancy im DBMS bringt in der Regel eine Katalogerweiterung [20] um Tabellen zur Repräsentation der Mandanten und der zugehörigen Datenbankobjekte wie Tabellen oder Indizes mit sich. Die Definition und anschließende Überwachung der SLOs bedarf wiederum einer Erweiterung des Katalogs um die SLOs jedes Mandanten sowie seiner zugewiesenen Bedeutsamkeit bzw. Priorität. Durch die Zuordnung eines Mandanten zu einer SLO-Kategorie wie 'Standard', 'Premium' oder 'Individuell' kann bei der SLO-Zuweisung der Overhead bezüglich des Fest- und Hauptspeicherbedarfs je Mandant reduziert werden. Abbildung 2 verdeutlicht diese Erweiterung und kennzeichnet, dass Katalogerweiterungen beispielsweise für die Zuweisung von Mandanten zu Anfragen (*Tenant Assignment*) und die in Abschnitt 2.3 angesprochene Transformationskomponente (*Query Transformation*) benötigt wird.

3.3 Workload Management

Einige Datenbankverwaltungssysteme bieten mittels Workload Management die Möglichkeit zur Überwachung von Abfragen und den von ihnen benötigten Ressourcen. IBM DB2 for Linux, UNIX and Windows stellt hierfür beispielsweise den DB2 Workload Manager [1] und Oracle den Oracle Database Resource Manager und Oracle Scheduler [2] bereit. Diese Anwendungen bieten ein breites Spektrum an Mitteln zur Überwachung von Anfragen, welches u.a. das Aufteilen von Systemressourcen zwischen Abfragen, die Priorisierung, Ab- und Unterbrechung von Abfragen sowie eine Zeitplanung von Abfragen enthalten kann.

Das DBMS muss fortwährend den Auslastungszustand der Ressourcen messen und für das Workload Management bereitstellen. Abbildung 2 verdeutlicht, mit welchen Komponenten des DBMS-Kerns der Workload Manager typischerweise kommuniziert, um die Verarbeitung von Abfragen zu steuern und zu überwachen [14]:

- *Execution Engine* (Verwaltung der Ausführung von Abfragen),
- *Performance Monitor* (Überwachung der Verarbeitung von Abfragen),
- *Resource Monitor* (Regelung der Ressourcenallokation der Abfragen).

Die Erweiterung des Datenbankkatalogs um Mandanten und SLOs stellt dem Datenbankverwaltungssystem die nöti-

gen Mittel zur Verfügung, um neben der korrekten Ausführung nebenläufiger Transaktionen eine auf SLOs basierende Parallelisierung von Mandanten zu erzielen. Verschiedene Nutzungsprofile und -zeiträume der Mandanten sowie übliche Nutzungsschwankungen erlauben eine Überbuchung der zur Verfügung stehenden physischen Systemressourcen wie der CPU, dem Hauptspeicher oder der Bandbreite zum Festspeicher. Dies ermöglicht eine ausgezeichnete Auslastung der Ressourcen und hält somit die operativen Kosten des SaaS-Anbieters gering. Im (Ausnahme-)Fall hoher Lastspitzen durch einen parallelen intensiven Zugriff einer Großzahl von Mandanten, welche die Ressourcen gemeinsam nutzen, kann die Einhaltung aller offerierten SLOs nicht gewährleistet werden. Lastspitzen können aufgrund von regelmäßigen Vorgängen wie Gehaltsbuchungen oder beispielsweise aufgrund von unvorhergesehenen Nutzungsschwankungen unregelmäßig auftreten [6]. Um die Häufigkeit dieser Situation zu reduzieren und beim Auftreten adäquat zu reagieren, bedarf es geeigneter Strategien zur Ablaufplanung der Abfragen sowie der Ressourcenzuteilung. Aus ökonomischer Sicht des Anbieters gilt es hierbei, die Summe der zu zahlenden Vertragsstrafen zu minimieren. In [13] wird dieses Ziel durch einen dynamischen Controller und der Zuhilfenahme zweier Kostenfunktionen verfolgt, womit zudem eine dauerhafte Übererfüllung der SLOs von Mandanten mit hoher Priorität auf Kosten von Mandanten mit geringer Priorität vermieden wird.

Häufig wird in diesen Modellen lediglich die Minimierung von Strafzahlen bezüglich der aktuellen Überauslastung betrachtet und die Korrelation von Entscheidungen bei verschiedenen Lastspitzen außer Acht gelassen. So ist es möglich, dass die SLOs eines Mandant mit geringer Priorisierung bei Lastspitzen wiederholt nicht erreicht werden können. Aus ökonomischer Sicht ist dies für den SaaS-Anbieter augenscheinlich eine optimale Strategie, sie kann jedoch zur Verärgerung oder gar Kündigung des Mandanten führen. Folglich gilt es, frühere Entscheidungen in die Priorisierung von Mandanten und Ressourcenzuweisung bei Lastspitzen einzubeziehen. Dieses Ziel kann nur mit Hilfe einer dynamischen Priorisierung erreicht werden.

Neben der Ablaufsteuerung und Überwachung von Abfragen sowie dem Eingreifen im Falle einer Überlastung besteht eine wesentliche Aufgabe des SLO-basierten Managements in dem Verhindern von häufigen Überlastungen eines Systems. Hierzu sind die Ressource sowie der Schweregrad, die Häufigkeit und eine gegebenenfalls existierende Regelmäßigkeit der Überlastungen zu beobachten. Durch ein proaktives Vorgehen kann zukünftigen Überlastungen vorgebeugt werden, indem Maßnahmen autonom ergriffen werden oder der Administrator in Form von Hinweisen bei seiner Tätigkeit unterstützt wird. Ein mögliches Vorgehen ist die Zuweisung von Mandanten zu anderen physischen Ressourcen durch das Verschieben auf einen anderen Server mit dem Ziel einer Lastverteilung.

3.4 Verteilung von Mandantendaten

Ein SaaS-Anwendung sollte die maximale Anzahl an unterstützten Mandanten möglichst nicht beschränken. Um auch bei vielen Mandanten eine ausreichende Performance zu erreichen, müssen die Daten der Mandanten mittels Tabellen- und Indexpartitionierung auf verschiedene Festspeicher oder gemäß eines verteilten Datenbanksystems auf verschiedene Datenbankknoten verteilt werden. Die Verteilung der Mandanten durch einen *Load Balancer* kann zudem gemäß Abschnitt 3.3 zur Abfederung von Lastspitzen genutzt werden. Entgegen statischer Ansätze [15] zur Berechnung einer optimalen Verteilung von Mandantendaten auf eine Menge von Datenbankknoten sollte die Lastverteilung analog zum Workflow Management dynamisch agieren.

Um unnötige Kommunikation zwischen Datenbankknoten zu vermeiden, sind die Daten eines Mandanten nicht über mehrere Knoten zu verteilen. Dies sollte nur möglich sein, wenn die Anforderungen des Mandanten die zur Verfügung stehenden Ressourcen des Rechners übersteigen.

Die Mandanten legen durch die Nutzung einer SaaS-Anwendung ihre Daten in die Hände des Anbieters und dessen auf Sicherheit spezialisierte IT-Abteilung [11]. Durch die Verteilung von Mandanten auf verschiedene Rechner ist neben dem verringerten Einfluss der Mandanten bezüglich ihrer Performance (Performance-Isolation zwischen Mandanten [8]) ein höherer Isolationsgrad der auf dem Festspeicher abgelegten Mandantendaten erreichbar bis hin zu einer physischen Isolation. Der Isolationsgrad spiegelt sich zum Teil in entsprechenden SLOs der Dienstverträge wider. Er kann durch die in Abbildung 2 dargestellte SLO-Katalogerweiterung im DBMS hinterlegt und vom Lastbalancierer bei der Verteilung der Mandanten auf Rechner berücksichtigt werden.

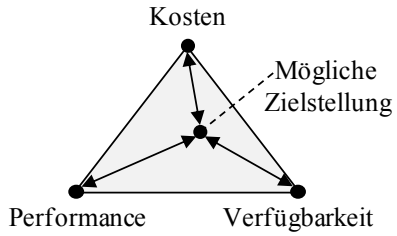


Abbildung 3: Zielkonflikte des SaaS-Anbieters

Verfügbarkeit besitzt gemäß Abschnitt 3.1 eine außerordentliche Bedeutsamkeit bei Dienstleistungsverträgen im SaaS-Umfeld. Ist der Dienst für die Mandanten nicht erreichbar, so kann dies schwerwiegende Folgen haben, die von einer Verzögerung der Arbeitsprozesse bei Mandanten bis hin zu finanziellen Einbußen reichen. Entsprechend werden in den Dienstleistungsverträgen nur geringe (geplante und ungeplante) Ausfallzeiten des Services zugelassen, bei deren Verletzung der SaaS-Anbieter mit erheblichen Strafzahlungen rechnen muss. Entsprechend liegt es im Interesse des Anbieters, eine hohe Verfügbarkeit des Dienstes sicherzustellen. Abbildung 3 verdeutlicht, dass die Erreichung eines hohen Verfügbarkeitsniveaus auf der einen Seite zu zusätzlichen Kosten für den Anbieter führt und auf der anderen Seite die Performance des Services einschränkt. Eine hohe Verfügbarkeit fordert das Vermeiden von Single Points of Failures und kann beispielsweise durch verschiedene Formen der Replikation erreicht werden. Der SaaS-Anbieter könnte durch einen *Replication Manager* die Art und den Umfang der Replikation von Mandantendaten aufgrund von verschiedenen SLOs der Mandanten variieren, um individuelle Anforderungen zu unterstützen.

3.5 SLA-Management in DaaS

Die Verwendung von Datenbanksystemen zur Datenverwaltung einer SaaS-Anwendung stellt nur eine Möglichkeit zur Bereitstellung von Datenbanksystemen als Dienst inner-

halb einer Cloud dar, was als Database as a Service (kurz DaaS oder DbaaS [21]) betitelt wird. Sowohl in kommerziellen DaaS-Angeboten als auch in der DaaS-Forschung spielt Multi Tenancy eine bedeutende Rolle, um die operationalen Kosten von DaaS-Anbietern zu senken. Die Bereitstellung von Daten für eine SaaS-Anwendung unterscheidet sich jedoch im Zusammenhang mit Multi Tenancy erheblich von anderen Dienstmodellen.

- Mandanten in einer SaaS-Anwendung besitzen und erweitern ein gemeinsames Basisschema und greifen zudem in der Regel lesend auf gemeinsame Anwendungsdaten zu. Bei anderen DaaS-Dienstmodellen existieren meist keine mandantenübergreifenden Daten und keine oder vernachlässigbare Ähnlichkeit der Datenbankschemata von Mandanten. Dies wirkt sich auf die Konsolidierungsmöglichkeiten innerhalb einer Datenbank aus.

- SaaS-Anwendungen bestimmen die Art der Workload für das verwendete Datenbanksystem, was sich das SLO-basiertes Management zu Nutze machen kann. Bei anderen DaaS-Dienstmodellen ist die Workload hingegen mandantenspezifisch und unvorhersehbar.

- SLOs sind bei SaaS-Angeboten mit der kompletten Anwendung verknüpft, während bei anderen DaaS-Dienstmodellen meist konkrete Vorgaben für das DBMS definiert sind.

Diese Punkte verdeutlichen, dass Ergebnisse aktueller Forschung im Bereich SLO-basierter Hardware-Provisionierung und Steuerung der Abfrageverarbeitung für DaaS [16] nur sehr eingeschränkt auf Datenbankdienste für SaaS-Anwendungen übertragen werden können und gesonderte Forschung für jenen Bereich vonnöten ist.

4. ZUSAMMENFASSUNG UND NÄCHSTE SCHRITTE

In diesem Beitrag wurde gezeigt, dass eine Integration von Service Level Objects in ein Multi-Tenancy-Datenbanksystem in Form einer Erweiterung des Datenbankkatalogs von verschiedenen DBMS-Komponenten verwendet werden kann, um die Überwachung der SLOs während des Betriebs zu gewährleisten. Als mögliche Verwendungsbeispiele wurden das Workload Management, die Lastverteilung und eine individuelle Replikationssteuerung aufgeführt.

In den weiteren Arbeiten soll der Entwurf der SLO-Integration konkretisiert, verschiedene Implementierungsvarianten verglichen und die Realisierbarkeit anhand eines Prototyps gezeigt werden. Anschließende Performance-Tests sollen Aufschluss darüber geben, ob der zusätzliche Overhead durch die mandantenspezifischen Metadaten und das SLO-basierte Management die Skalierbarkeit und den laufenden Betrieb des Datenbanksystems beeinträchtigen.

5. LITERATUR

[1] *DB2 Workload Manager for Linux, Unix, and Windows*. IBM Corp., 2008.

[2] *Oracle® Database Administrator's Guide 11g Release 2*. Oracle, 2011.

[3] C. Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[4] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD*, pages 1195–1206. ACM, 2008.

[5] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. A comparison of flexible schemas for software as a service. In *SIGMOD*, pages 881–888. ACM, 2009.

[6] S. Aulbach, D. Jacobs, J. Primsch, and A. Kemper. Anforderungen an Datenbanksysteme für Multi-Tenancy- und Software-as-a-Service-Applikationen. In *BTW*, pages 544–555. GI, 2009.

[7] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper. Extensibility and Data Sharing in evolving multi-tenant databases. In *ICDE*, pages 99–110, 2011.

[8] D. Banks, J. Erickson, M. Rhodes, and J. S. Erickson. Multi-tenancy in Cloud-based Collaboration Services. *Information Systems Journal*, 2009.

[9] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart. Enabling multi-tenancy: An industrial experience report. In *ICSM*, pages 1–8, 2010.

[10] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail. 2006.

[11] A. Göbel. Anforderungen von Cloud-Anwendungen an Datenbanksysteme. In *Workshop Database as a Service*, 2010.

[12] D. Jacobs and S. Aulbach. Ruminations on Multi-Tenant Databases. In *BTW*, pages 514–521, 2007.

[13] S. Krompass, D. Gmach, A. Scholz, S. Seltzsam, and A. Kemper. Quality of Service Enabled Database Applications. In *ICSOC*, pages 215–226, 2006.

[14] S. Krompass, A. Scholz, M.-C. Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of Service-enabled Management of Database Workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[15] T. Kwok and A. Mohindra. Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications. In *ICSOC*, pages 633–648. Springer-Verlag, 2008.

[16] W. Lang, S. Shankar, J. Patel, and A. Kalhan. Towards Multi-Tenant Performance SLOs. In *ICDE '12*, 2012.

[17] P. M. Nadkarni and C. Brandt. Data extraction and ad hoc query of an entity–attribute–value database. *Journal of American Medical Informatics Association*, 5(6):511–527, 1998.

[18] Pierre Audoin Consultants. Entry of the global players confirms Global SaaS Trends. 2012.

[19] B. Reinwald. Database support for multi-tenant applications. *IEEE Workshop on Information and Software as Services*, 1:2, 2010.

[20] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. Native support of multi-tenancy in RDBMS for software as a service. In *EDBT/ICDT*, pages 117–128. ACM, 2011.

[21] M. Seibold and A. Kemper. Database as a Service. *Datenbank-Spektrum*, 12(1):59–62, 2012.

[22] R. Sturm, W. Morris, and M. Jander. *Foundations of Service Level Management*. SAMS Publishing, Apr. 2000.

*i*ETL: Flexibilisierung der Datenintegration in Data Warehouses

Sebastian Schick[1], Gregor Buchholz[2], Meike Klettke[3], Andreas Heuer[1], Peter Forbrig[2]

[1]Lehrstuhl für Datenbank- und Informationssysteme; [2]Lehrstuhl für Softwaretechnik
[3]Institut für Informatik
Universität Rostock, 18051 Rostock
vorname.nachname@uni-rostock.de

ABSTRACT

Data Warehouses bestehen aus zwei Hauptkomponenten: einer flexiblen Anfrageschnittstelle zur Datenanalyse (OLAP) und einer relativ starren ETL-Komponente zum Laden der Daten ins Data Warehouse. In diesem Artikel soll vorgestellt werden, wie die Datenintegration bedarfsabhängig zu flexibilisieren ist, welche Vorteile sich daraus ergeben und welche Herausforderungen bei der Entwicklung eines solchen interaktiven ETL (*i*ETL)-Prozesses bestehen.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*; H.2.7 [**Database Management**]: Database Administration—*data warehouse and repository*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*query formulation, search process*

General Terms

Data Warehouse

Keywords

Data Warehouse, ETL-Prozess, Szenario, Datenintegration

1. MOTIVATION

Institutionen des öffentlichen Sektors sehen sich mehr noch als industrielle Verwaltungen einer Vielzahl von Softwarelösungen zur Unterstützung ihrer Prozesse gegenüber. Während in Industrien wie dem Automobilbau oder dem Bankenbereich fünf bis zehn Kernkompetenzen in der IT dargestellt werden, finden sich im Kerngeschäft öffentlicher Verwaltungen leicht zwischen 100 und 150 Prozesse verschiedener Dienstleistungskomplexe [11]. Diese Aufgaben- und Informationsfülle spiegelt sich in der Heterogenität der vorzuhaltenden Lösungen, der Vielfalt der Schnittstellen zum Informationsaustausch sowie der Menge vorzufindender Datenbanklösungen und Datenablagen in Excel und ähnlichen Formaten wider. Dem gegenüber steht der zunehmende Bedarf an zentralen Beobachtungs- und Steuerungsinstrumenten der Bereiche Business- und Geo-Intelligence. Der Verbreitung fachübergreifender Systeme steht oft der sehr hohe Datenbeschaffungsaufwand (sowohl initial als auch prozessbegleitend) im Weg. Insbesondere die Erschließung neuer Datenquellen bei der Ausweitung von Kennzahlensystemen auf neue Fachgebiete oder bei Auftreten tagesaktueller *ad-hoc*-Abfragen mit teils „exotischen“ Fragestellungen geht stets mit großem manuellen Aufwand bei der Informationssuche und -transformation einher. Es geht also um eine Lösung zur Identifizierung und Integration heterogener Datenquellen im Data Warehouse (DW)-Umfeld, die den Anwender in verschiedensten Datenquellen enthaltene Informationen finden und in seinen Bestand übernehmen lässt. Nicht im Fokus stehen von technischem Personal eingerichtete turnusmäßige Beladungen oder Real-Time-Data-Warehousing sondern die zunächst einmalige Integration aus Quellen mit überwiegend statischem Inhalt und geringer Komplexität durch den Anwender. Kapitel 2 illustriert dies anhand zweier Szenarien aus der Anforderungsanalyse. Abb. 1 zeigt in durchgezogenen Pfeilen bestehende Daten- und Kontrollflüsse und in unterbrochenen Linien die zu entwickelnden Verbindungen. Interessant dabei ist, wie der Prozess aus Nutzersicht verlaufen kann und mehr noch, mittels welcher Architekturen und Datenstrukturen die daraus ermittelten Anforderungen am besten umzusetzen sind.

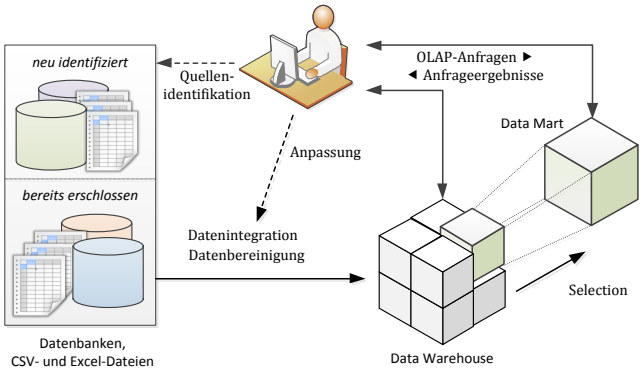


Abbildung 1: Interaktiver ETL-Prozess eines DW

*Die Arbeit dieser Autoren wird durch das BMWi im ZIM-Projekt KF2604606LF1 der GeoWare GmbH und der Universität Rostock gefördert.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

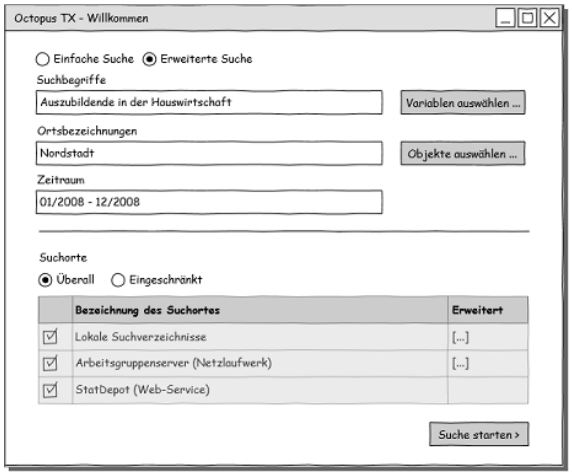


Abbildung 2: Eingabemaske Szenario 1

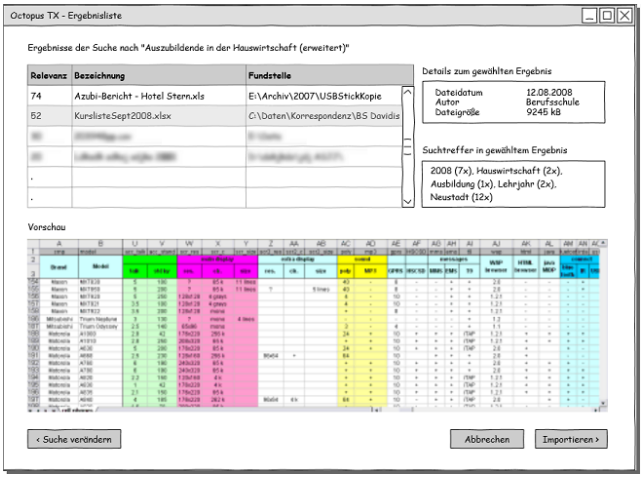


Abbildung 3: Ergebnisliste Szenario 1

2. ANWENDUNGSSZENARIEN

In der Anforderungsanalyse dieses Projektes sind Szenarien ([3], S. 52) zur Veranschaulichung der gewünschten Funktionalität entstanden, die beim Entwickeln einer Lösung helfen sollen. Die folgende Wiedergabe dieser Beispielanwendungen beginnt nach dem Skizzieren des technischen Kontextes jeweils mit der Beschreibung einer Bedarfssituation, an die sich eine mögliche Lösung aus Anwendersicht anschließt. Das folgende Kapitel 3 schlägt dann ein Konzept zur Umsetzung dieser Anforderung vor.

Szenario 1 – Kontext: Die Klassifikationshierarchie des DW mit ihren Attributen ist dem System bekannt. Ebenso wurden die möglichen Datenquellen (Web-Services, Suchpfade im Dateisystem) bereits konfiguriert. Die drei Dimensionen der Daten im DW sind: *Zeitraum*, *Geo-Objekte* (Hierarchie geographischer Bezugselemente) und *Kenngrößen*.

Situation: Wenige Tage vor der Jahresversammlung des Gaststätten- und Hotelverbandes wird Herr B. im Amt für Ausbildungsförderung mit dem Zusammenstellen einer Statistik beauftragt. Sie soll die Entwicklung der Auszubildendenzahlen der vergangenen Jahre in diesem Bereich aufzeigen. Dazu fragt er die dafür relevanten Informationen in einem DW-System an und erkennt an der grauen Einfärbung des entsprechenden Knotens in seiner DW-Anwendung, dass die Daten zur Kenngröße „Auszubildende in der Hauswirtschaft“ für den Stadtteil „Nordstadt“ im Jahr 2008 fehlen. Dass sie nicht wie sonst automatisch übermittelt wurden, liegt seiner Meinung nach an der Umbenennung des Stadtteils (früher: „Neustadt“) im Vorjahr.

Lösungsausblick: Über einen Rechtsklick auf den ausgegrauten Knoten lässt Herr B. eine Anfrage an das Suchsystem generieren, was ihn zur Eingabemaske (Abb. 2) führt. Die Suchfelder für die drei Dimensionen sind schon vorbelegt; per direkter Texteingabe oder über die Schaltflächen „Variablen auswählen“ und „Objekte auswählen“ könnte Herr B. die Suchkriterien verändern; die jeweiligen Dialoge stellen die möglichen Werte der jeweiligen Dimension zur Auswahl. Er tippt in das Suchfeld der Ortsbezeichnungen „Neustadt“ ein und bekommt nach Abschluss der Suche als Ergebnis eine Liste von Datenquellen zu sehen (Abb. 3). Das erste Ergebnis ist offenbar ein Bericht eines konkreten Hotels und damit nicht relevant. Herr B. wählt also den zweiten Treffer und betrachtet ihn in der Voransicht. Er erkennt, dass die gesuchten Daten (eine Auflistung der Auszubildenen mit Wohn- und Ausbildungsadresse) in dem Suchergebnis enthalten sind und wechselt via „Importieren“ zum Import-Modul für Excel-Dateien. Dort kann er einzelne Spalten und Bereiche der Excel-Tabelle als Werte der drei Dimensionen markieren und den Import in sein DW-Systems anstoßen. Anschließend widmet er sich der Aufbereitung der Statistiken.

Szenario 2 – Kontext: Die Konfiguration entspricht der von Szenario 1. Hier wird jedoch die Anfrage nicht vom DW-System vorbelegt.

Situation: Vor dem Beitritt zu einem Verband zur Förderung von Solarenergieanlagen an privaten Immobilien sollen für 2010 Anzahl, Verteilung und Höhe der Landesförderung von Anlagen ermittelt werden. Frau B. ist damit beauftragt und stellt fest, dass zu den Förderungen bislang keine Daten im DW existieren, jedoch hat sie kürzlich davon gehört, dass ein Mitarbeiter einem anderen per Mail eine solche Übersicht schicken wollte.

Lösungsausblick: Sie startet das Suchsystem und spezifiziert ihre Anfrage: „solaranlagen 2010 +förderung +privat -gewerblich“ (siehe Abb. 4). Der Begriff „solaranlagen“ und das Jahr „2010“ sollen im Ergebnis enthalten sein. Ebenso „förderung“ und „privat“, die ihr besonders wichtig sind und für die das „+“ eine erhöhte Priorität der Fundstellen bewirken soll. Kommt hingegen „gewerblich“ in der Fundstelle vor, soll die Priorität des Ergebnisses sinken. Als Suchort schließt Frau B. die Datenquelle „StarDepot“ aus, dann startet sie die Suche. In einer Ansicht ähnlich zu Abb. 3 nutzt sie die Vorschau, um die Datei mit den gesuchten Informationen zu identifizieren. Anschließend bereitet sie die Daten für den Import vor und übernimmt sie in den eigenen Datenbestand.

3. LÖSUNGSKONZEPT

In Abschnitt 2 wurden Anwendungsszenarien und mögliche Lösungsausblicke vorgestellt, die eine Anpassung des ETL-Prozesses notwendig machen. In diesem Abschnitt stellen wir dafür eine erweiterte DW-Architektur vor.

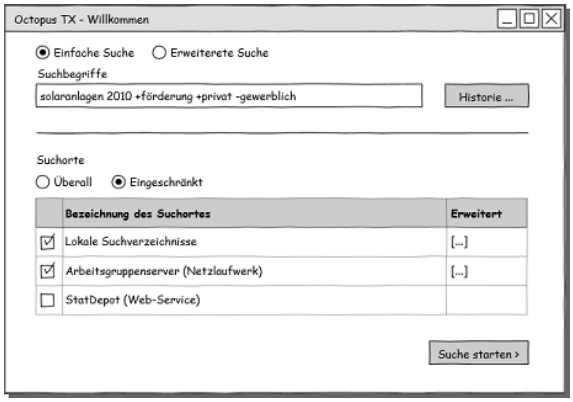


Abbildung 4: Eingabemaske Szenario 2

3.1 Ausgangspunkt

Der Anwender soll bei der Quellenidentifikation und Datenintegration im ETL-Prozess in einem DW unterstützt werden. Dafür müssen die verfügbaren Datenquellen so aufbereitet werden, dass eine Recherche und eine anschließende Auswahl von geeigneten Datenquellen möglich ist. Die Heterogenität der Datenquellen erschwert die automatische Integration in das DW. In föderierten Datenbanken gab es umfangreiche Untersuchungen zu Heterogenitäten der einzelnen Datenquellen bzgl. Syntax und Semantik von Werten, Attributen, Relationen und Modellen [6]. Die Transformation von Daten aus heterogenen Formaten in eine einheitliche Repräsentationsform stellt das Hauptproblem bei der Integration dar. Der Anwender muss deshalb bei der Datenintegration und insbesondere bei der Datentransformation unterstützt werden. Angepasste Nutzerinterfaces sollen den technikunerfahrenen Anwender unterstützen.

Der Prozess des Füllens eines DW mit Daten wird als ETL-Prozess bezeichnet, ETL steht hierbei für Extract, Transform und Load. Die Basisdaten in den meisten Anwendungen sind heterogene Daten, die in ein einheitliches Format, das multidimensionale Modell des DW integriert werden sollen. Bei diesem Prozess werden die neuen oder veränderten Daten aus den Basisdatenquellen ausgewählt (**E**xtraction), in eine einheitliche Darstellung umgewandelt (**T**ransformation), dabei vervollständigt, Duplikate bereinigt und eventuell voraggregiert. Anschließend erfolgt das Laden in die Datenbank (**L**oad) [5]. Im vorgestellten Ansatz soll eine Wissenskomponente den ETL-Prozess unterstützen, indem einzelne Komponenten um semantische und ontologische Konzepte erweitert werden. Wir schlagen deshalb eine Erweiterung des klassischen ETL-Prozesses in folgenden Bereichen vor:

- **Quellenidentifikation**: Methoden des Information-Retrieval sollen den Anwender bei der Identifikation und Vorauswahl von Datenquellen unterstützen.
- **Datenintegration**: Die flexible Integration heterogener Datenquellen soll durch semiautomatische Techniken gefördert werden.
- **Datenextraktion** (als Teil der Datenintegration): Der Anwender soll durch geeignete Nutzerinterfaces die Abbildungs- und Transformationsvorschriften effizient bestimmen können.
- **Wissensbasis**: Sämtliche Komponenten sollen zur Flexibilisierung um semantische und ontologische Konzepte erweitert werden.

Wir schlagen deshalb vor, das Referenzmodell für die Architektur von DW aus [2] derart zu erweitern, dass der Anwender bei der Identifikation passender Datenquellen unterstützt, der Integrationsprozess heterogener Datenquellen erleichtert und die Flexibilisierung der Datenextraktion mit geeigneten Konzepten ermöglicht wird. Die Architektur ist in Abbildung 5 dargestellt. Datenflüsse zwischen den Komponenten sind als durchgezogene Pfeile umgesetzt, der Kontrollfluss wird mit unterbrochenen Linien markiert.

3.2 Die Wissenskomponente

Die zentrale Komponente in der vorgestellten Architektur bildet der Data Warehouse Manager (DWM) (siehe Abb. 5). Der DWM steuert in einem klassischen DW nach [2] alle Komponenten, die zur Anfrage und Darstellung der Daten notwendig sind: Monitore, Extraktoren, Ladekomponenten und Analysekomponenten. Zusätzlich erhält der DWM in der hier vorgestellten Architektur Schnittstellen

- zur zentralen Wissenskomponente, die für die Planung und Ausführung der Quellenidentifikation und Datentransformation im ETL-Prozess benötigt wird und
- zur Search Engine zwecks Quellenidentifikation.

Konzept.

Die Wissenskomponente (**knowledge component**) stellt Informationen über Klassifikations- und Dimensionshierachien, semantische Verknüpfungen und die Typisierung sowie Metadaten einzelner Attribute bereit (siehe Abb. 5). Das domänenspezifische Wissen wird durch Quellenangaben, Synonyme und Muster (Format- bzw. Modell-Pattern) ergänzt. Die Wissenskomponente ist für die Prozesse der Quellenidentifikation und Datenintegration neuer Datenquellen unabdingbar.

- Der **Metadata Manager** stellt eine Schnittstelle bereit, über die andere Komponenten Anfragen an die Wissensbasis und das Metadaten-Repository stellen und Antworten anfordern können.
- Die **Knowledge Base** beinhaltet ein Wissensmodell für die Speicherung und Verwaltung der semantischen und ontologischen Informationen.
- Das **Metadata Repository** beinhaltet alle weiteren Metadaten, die vom DWM benötigt werden.

Herausforderungen.

Die Umsetzung einer Wissenskomponente erfordert den Aufbau einer Wissensbasis zum Anwendungsgebiet des DW, womit je nach Anwendungsszenario ein hoher manueller Aufwand verbunden ist. Ein Teil der Wissensbasis kann aus den hierarchischen Klassifikationsattributen der Dimensionen des DW übernommen werden.

Zusätzlich zu diesen hierarchischen Informationen werden Wörterbücher benötigt, die die Verbindung zwischen Konzepten der Wissensbasis und Suchbegriffen herstellen. Diese Wörterbücher sind initial zur erstellen und sollen beim Einsatz des *i*ETL-Tools von einer lernenden Komponente erweitert und anpasst werden.

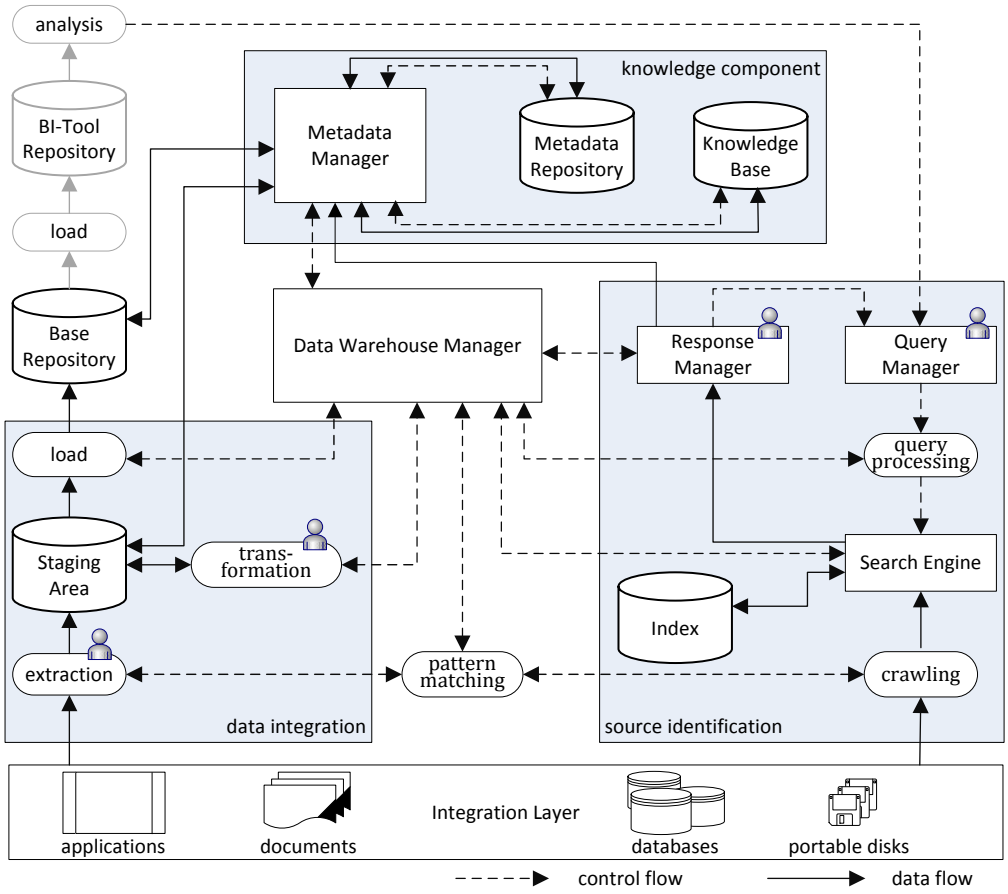


Abbildung 5: Architektur für den flexiblen ETL-Prozess

3.3 Quellenidentifikation

Liegt in einem klassischen DW die Auswahl der Datenquellen bzw. der Quelldaten vorrangig bei den zuständigen Administratoren, so soll in diesem Ansatz der Anwender des DW in die Quellenidentifikation einbezogen und dabei unterstützt werden.

Konzept.

Eine Suchkomponente soll bei der Quellenidentifikation im ETL-Prozess dem Anwender die Auswahl weiterer Datenquellen erleichtern. Dafür soll ein Index über alle verfügbaren Datenquelle aufgebaut werden, die über den **Integration Layer** verfügbar sind. Der Prozess der Quellenidentifikation (*source identification*) ist in Abbildung 5 rechts dargestellt. Die Komponenten sind an die Architektur einer Web-Suchmaschine angelehnt, wie sie beispielsweise in [1] (Seite 460) vorgeschlagen wird. Sie werden im Folgenden vorgestellt.

- Der **Query Manager** stellt über eine Schnittstelle einfache oder erweiterte Suchmasken bereit. Für die Suche werden vorerst die existierenden Dimensionen aus dem DW als Suchparameter verwendet:
  - *Was*: Kenngrößen, die in einer Datenquelle enthalten sein müssen.
  - *Wo*: Geo-Objekte, die durch die Werte beschrieben werden.
  - *Wann*: Datumsangaben und Zeitbereiche.
- Das **Query Processing** beschreibt eine Vorverarbeitung der Anfrage unter Verwendung der Wissensbasis. Dabei soll eine Anpassung der Anfrage hinsichtlich der Struktur (Ort, Zeit, Schlüsselwörter), die Expansion der Anfrage mit Hilfe der Wissensbasis sowie ein Vokabularmapping und weitere Vorverarbeitungsschritte wie eine lexikografische Analyse oder Stoppworteleminierung stattfinden.
- Die **Search Engine** soll die Indizierung von Datenquellen, eine Anfrageverarbeitung und die Bereitstellung von Ergebnislisten übernehmen. Die Indizierung (mit gleichen Methoden wie bei der Anfrageverarbeitung) soll durch eine Strukturanalyse auf Basis von Mapping-Mustern (die während der Datenintegration erzeugt werden) umgesetzt werden. Neben dem Bereitstellen von Anfrageergebnissen aus heterogenen Datenquellen sollen Suchergebnisse mit Informationen über den Fundort innerhalb der Datenquellen angereichert werden, wofür domänenspezifische Informationen und Musteranalysen aus der Wissensbasis zum Einsatz kommen sollen.
- Das **Crawling** beschreibt den Schritt, in dem verfügbare Datenquellen durchsucht und für die Indizierung bereitgestellt werden. In diesem Schritt ist die Nut-

zung vorhandener Mapping-Muster zu Strukturanalysen und semantischen Auswertungen geplant.

- Der **Response Manager** wird für die Präsentation der Anfrageergebnisse genutzt. Dem Nutzer soll eine Vorauswahl von Datenquellen durch eine vereinfachte Datenvorschau ermöglicht werden, eine semiautomatische Identifizierung möglicher Indikatoren, Variablen und Zeiträume in den Anfrageergebnissen soll dabei erfolgen (siehe Abb. 3). Die ausgewählten Quellen werden hier an den DWM übergeben, der in einem nächsten Schritt die Datenintegration anstoßen kann.

Herausforderung.

Eine Herausforderung ist das Design des Anfrageinterfaces und der Ergebnisdarstellung. Hierfür müssen die Anforderungen der Anwender bestimmt werden.

- **Anfrageinterfaces**: Wie können Suchanfragen auf Ordner, Datenquellen oder Dateitypen eingegrenzt werden und welche der Anfragetypen Context (Phrase, Boolean, etc.), Pattern Matching oder strukturierte Anfragen (formularbasiert) können genutzt werden. Außerdem ist zu klären, ob die Klassifizierung der Anfrage durch vorhandene Kategorien der Wissensbasis möglich und sinnvoll ist.

- **Ergebnisdarstellung**: Wie muss eine zielgerichtete Präsentation der Anfrageergebnisse unter Verwendung der Wissensbasis umgesetzt werden und wie ist dabei eine semiautomatische Identifizierung potentieller Indikatoren, Variablen und Zeiträume möglich. Daneben sollen relevante Elemente der Ergebnismenge hervorgehoben und die Identifizierung von Strukturen innerhalb eines Treffers durch Anwendung von Mapping-Mustern möglich sein.

- **Query Processing** (basierend auf der Wissensbasis): Wie kann die Wissensbasis für die Anfrageerweiterung in Form einer facettierten Suche genutzt und wie können Methoden des Relevance Feedback zur Verbesserung der Qualität der Ergebnisse genutzt werden.

- **Search Engine**: Wie kann die Integration externer Anwendungen (Enterprise Search, ERP, CRM, etc.) umgesetzt werden, wenn die bereitgestellte Anfrageschnittstellen nur Teilergebnisse liefern oder der Umfang der Datenbasen zu groß ist. Die Integration unterschiedlicher Datenformate soll ebenso unterstützt werden, wie die Duplikaterkennung, wenn Inhalte und Daten aus unterschiedlichen Quellen genutzt werden. Weiterhin sollen Mapping-Muster für den Prozess der Indizierung und Extraktion genutzt werden und Klassifikation durch die Mapping-Muster unterstützt werden.

3.4 Datenintegration

Die Datenintegration muss bedarfsabhängig und flexibel angepasst werden, wenn durch die Quellenidentifikation neue Datenquellen zu integrieren sind. Die Flexibilisierung soll durch Anwendung von semantischen und ontologischen Konzepten erreicht werden, wodurch domänenspezifisches Wissen ausgenutzt wird. Die Architektur in Abbildung 5 ist dabei an die Referenzarchitektur angelehnt.

Konzept.

- Mit **extraction** wird die Übertragung von Daten aus externen Quellen in den Arbeitsbereich (*staging area*) beschrieben. Die Auswahl der Datenquellen wurde im Vorfeld durch den Anwender (Quellenidentifikation) durchgeführt. Der Prozess muss um semiautomatische Methoden des Schema Matchings und Mappings erweitert werden (**pattern matching**).

  Die bei der Datenextraktion und -transformation erzeugten Mapping-Mustern sollen für eine spätere Wiederverwendung in der Wissensbasis vorgehalten werden und bei jeder Datenintegration auf ihre Anwendbarkeit hin überprüft werden. Passende Muster für die Extraktion einer Datenquellen werden dem Anwender angeboten. Die Schritte der Extraktion und Transformation müssen durch angepasste, graphische Tools unterstützt werden.

- Mit **transformation** wird die Abbildung der Daten hinsichtlich struktureller und inhaltlicher Aspekte beschrieben. Neben der Datentransformation (z. B. Konvertierung von Kodierungen, Vereinheitlichung von Datumsangaben, etc.) sollen hier auch eine Datenbereinigung, Duplikaterkennung und eine Datenfusion stattfinden (siehe auch [2]). Für diesen Schritt sind ebenfalls Informationen aus der Wissensbasis notwendig. Vorschläge für Transformationsvorschriften können aus der Wissensbasis abgeleitet werden.

- Mit **load** wird die Übertragung der Daten aus dem Arbeitsbereich in das **Base Repository** beschrieben. Die Daten stehen dann für die weitere Verarbeitung durch unterschiedliche BI-Tools (Business Intelligence Tools) zur Verfügung. Durch ein erneutes Laden werden die Daten in ein externes **BI-Tool Repository** geladen (grau hinterlegt) und stehen so in DW-Anwendungen für weitere OLAP-Analysen zur Verfügung.

Herausforderungen.

Die Herausforderungen bei der Datenintegration liegen bei der Tool-Unterstützung des Anwenders, sowie bei der semantischen Unterstützung des Transformationsprozesses. Das Schema einer Datenquelle ist in der Regel unbekannt, weshalb es mit Hilfe geeigneter Werkzeuge extrahiert werden muss.

- **Transformation**: Die Datentransformation aus dem Format der Basisdatenquelle ins Zielformat kann nicht vollständig automatisiert werden. Herausforderung ist hier die Entwicklung von Nutzerinterfaces zur Eingabe der benötigten Informationen durch den Fachanwender. Die dabei entstehenden Transformationsmuster sollen gespeichert werden, damit sie für andere Datenquellen verwendet werden können.

  Welche vorhandenen Ansätze der Datenintegration können für die Datenbereinigung, Duplikaterkennung und Datenfusion angewendet werden und wie kann eine Plausibilitätsprüfung der Daten unterstützt werden. Für eine Plausibilitätsprüfung können z. B. Regeln definiert werden, die die Wissensbasis einbeziehen. Ein möglicher Ansatzpunkt ist hier die Angabe von check-constraints.

3.5 Einsatz des Verfahrens

Das im Projekt zu entwickelnde Verfahren wird sich nicht auf alle DW anwenden lassen. Voraussetzung ist, dass es eine Wissensbasis zu dem Anwendungsgebiet des DW gibt. Da diese Wissensbasis eine zentrale Rolle beim Finden der relevanten Datenquellen und bei der Transformation der Daten ins DW spielt, muss eine solche Wissensbasis für einen flexiblen ETL-Prozess vorhanden sein. Teile der Wissensbasis lassen sich aus den Klassifikationsattributen der Dimensionen des DW generieren; die Zuordnung dieser Klassifikationshierarchie zu den korrespondierenden Suchbegriffen für die Datenquellen muss für das jeweilige Anwendungsgebiet ergänzt werden.

4. RELATED WORK / STAND DER TECHNIK

4.1 Datenintegration

Jede Datenintegration bewirkt das Zusammenführen von Daten aus heterogenen Datenbanken und Informationssystemen. Es gibt Klassifikationen, die die Heterogenitäten der einzelnen Datenquellen systematisieren. Heterogenitäten können bzgl. Syntax und Semantik von Werten, Attributen, Relationen, Modellen existieren ([6]). Eine Standardarchitektur, die das Zusammenführen von heterogenen Formaten in heterogenen Datenbanken vornimmt, wurde bereits im Jahr 1990 in [10] vorgeschlagen.

Eine dabei bestehende Aufgabe ist Matching und Mapping heterogener Datenbanken. Es gibt mehrere Mapping-Tools, die eine intuitiv bedienbare Oberfläche anbieten, um dem Benutzer das Entwickeln von Datentransformationskomponenten zu erleichtern (wie Altova MapForce[1], oder IBM Data Integrator), dieser Prozess ist jedoch nicht automatisierbar. Einen Überblick über Forschungsansätze in dieser Richtung findet man in [9]. Dabei spielen vor allem Ontologie-basierte Ansätze eine große Rolle (vgl. [7] und [4]).

4.2 ETL

Beim ETL-Prozess in einem DW werden die Basisdaten (meist heterogene Daten) in ein einheitliches Format, das multidimensionale Modell des DW integriert [5]. Man kann den ETL-Prozess eines DW als Spezialfall föderierter Datenbanken sehen. Für neue Datenquellen bedeutet der ETL-Prozess also manuellen Aufwand, der eine Interaktion mit einem Benutzer erfordert; im laufenden Prozess kann das Laden neuer Daten dann automatisch ausgeführt werden. Es stehen Tools zur Vereinfachung dieses Prozesses für die Anwender zur Verfügung, Beispiele dafür sind Talend[2] und IBM Data Stage[3].

4.3 Verwendung von Ontologien im ETL-Prozess

Die Idee, Ontologien zur Beschreibung von Objekten einzusetzen, ist weit verbreitet. Im DW-Bereich gibt es einen Vorschlag, Ontologien zu verwenden, um die Metadaten des Data Warehouses daraus abzuleiten [8]. In unserem Ansatz soll die Kopplung dieser beiden Gebiete auf andere Weise erfolgen: Aus den Klassifikationsattributen des DW soll eine Wissensbasis gebildet werden, die um Wörterbücher ergänzt wird.

[1] `www.altova.com/mapforce.html`
[2] `www.talend.com`
[3] `www.ibm.com/software/data/infosphere/datastage`

5. ZUSAMMENFASSUNG, AUSBLICK

Die flexible, durch situativ entstandenen Datenbedarf initiierte Integration bislang unerschlossener Datenquellen in ein Data Warehouse erfordert eine Anreicherung des ETL-Prozesses um interaktive Schritte. Um diesen Prozess für den Fachanwender handhabbar zu halten, bedarf es zusätzlicher Komponenten zur Speicherung und Nutzung von domänenspezifischem Wissen (*knowledge component*), die das Finden (*source identification*) und Integrieren (*data integration*) neuer Daten erleichtern bzw. erst ermöglichen.

Geleitet von Anwendungsszenarien wurde ein Konzept zur Architektur eines solchen Systems vorgestellt. Die Herausarbeitung technischer Herausforderungen zeigt den zu gehenden Weg: Die Details der einzelnen Komponenten sind zu konkretisieren, bislang nicht kombinierte Techniken zu verbinden und eine angemessene Nutzerschnittstelle zu entwickeln.

6. REFERENCES

[1] Baeza-Yates, R. und B. Ribeiro-Neto: *Modern information retrieval: the concepts and technology behind search*. Addison-Wesley, Pearson, Harlow [u.a.], 2. Aufl., 2011.

[2] Bauer, A. und H. Günzel: *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. dpunkt-Verl., Heidelberg, 2. überarb. und aktualisierte Aufl., 2004. Literatur- und URL-Verz. S. 545–576.

[3] Courage, C. und K. Baxter: *Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques*. Morgan Kaufmann, 1. Aufl., 2005.

[4] Doan, A. und A. Y. Halevy: *Semantic Integration Research in the Database Community: A Brief Survey*. AI Magazine, 26(1):83–94, 2005.

[5] Inmon, W.: *Building the data warehouse*. Wiley, 2005.

[6] Kim, W. und J. Seo: *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*. Computer, 24(12):12–18, Dez. 1991.

[7] Noy, N. F.: *Semantic integration: a survey of ontology-based approaches*. SIGMOD Rec., 33(4):65–70, Dez. 2004.

[8] Pardillo, J. und J.-N. Mazón: *Using Ontologies for the Design of Data Warehouses*. CoRR, abs/1106.0304, 2011.

[9] Rahm, E. und P. A. Bernstein: *A survey of approaches to automatic schema matching*. VLDB Journal, 10(4):334–350, 2001.

[10] Sheth, A. P. und J. A. Larson: *Federated database systems for managing distributed, heterogeneous, and autonomous databases*. ACM Comput. Surv., 22(3):183–236, Sep. 1990.

[11] Vitako: *IT-Monitor kommunal*. Vitako aktuell. Bundesarbeitsgemeinschaft der Kommunalen IT-Dienstleister e.V, 2007.

Ereignismuster für die Verwaltung von komplexen Tupelereignissen in Probabilistischen Datenbanken

Sebastian Lehrack
Brandenburgische Technische Universität Cottbus
Institut für Informatik, Postfach 10 13 44
D-03013 Cottbus, Deutschland
slehrack@informatik.tu-cottbus.de

ABSTRACT

Probabilistische Datenbanken haben sich als adäquate Technik zur Verwaltung und Verarbeitung von umfangreichen unsicheren Datenmengen etabliert. Eine der größten Herausforderungen für probabilistische Datenbanken ist eine effiziente Anfrageverarbeitung. Eine zentrale Rolle spielen dabei Ableitungsformeln, welche komplexe Tupelereignisse in Form von aussagenlogischen Formeln verkörpern. Eine *direkte* Abspeicherung und Verarbeitung von komplexen logischen Formeln wird von relationalen Datenbanksystemen jedoch nicht unterstützt. Diese Arbeit stellt Ereignismuster als geeignetes Mittel vor, um Ableitungsformeln mittels eines RBDMS verwalten zu können.

1. MOTIVATION

Probabilistische Datenbanken standen in den letzten Jahren im Mittelpunkt intensiver Untersuchungen (für einen Überblick verweisen wir auf [18]). In einer solchen Datenbank gehört ein Tupel lediglich mit einer gewissen Wahrscheinlichkeit zu einer Datentabelle oder zu einem Anfrageergebnis. Die Wahrscheinlichkeit drückt dabei entweder eine *Unsicherheit* über die Datengrundlage oder die Anfrageantwort aus.

Eine der größten Herausforderungen für probabilistische Datenbanken ist eine effiziente Anfrageverarbeitung. Ein zentrales Konzept sind dabei Ableitungsformeln, welche komplexe Tupelereignisse in Form von aussagenlogischen Formeln darstellen [4]. Eine direkte Abspeicherung und Verarbeitung von komplexen logischen Formeln wird von relationalen Datenbanksystemen jedoch *nicht* unterstützt. Diese Arbeit stellt Ereignismuster als geeignetes Mittel vor, um Ableitungsformeln in einer strukturierten Form mittels eines RBDMS verwalten zu können.

Fortlaufendes Beispiel: Die grundlegenden Ideen werden in den nächsten Kapiteln anhand eines fortlaufenden Beispielszenarios aufgezeigt. Dieses Szenario ist motiviert durch die Neuentwicklung des CISAR-Projektes[1] [5], welches als internet-basiertes Geo-Informationssystem für Archäologie und Gebäudegeschichte entwickelt worden ist. Die hier vorgestellten Techniken werden umfassend in dem Nachfolgesystem OpenInfRA eingesetzt [17].

Arte

tid	aid	type	sond	age	ET_{Arte}
t_1	art1	vase fragment	3	300	$(X_1 = t_1)$
t_2	art2	spear head	10	500	$(X_2 = t_2)$
t_3	art3	vase fragment	4	300	$(X_3 = t_3)$

ArteExp

tid	expert	aid	culture	conf	$ET_{ArteExp}$
t_4	Peter	art1	roman	0.3	$(X_4 = t_4)$
t_5	Peter	art1	greek	0.4	$(X_4 = t_5)$
t_6	Kathleen	art1	roman	0.4	$(X_5 = t_6)$
t_7	John	art2	egyptian	0.6	$(X_6 = t_7)$

ArteMat

tid	method	aid	culture	conf	$ET_{ArteMat}$
t_8	XRF	art1	roman	0.3	$(X_7 = t_8)$
t_9	XRF	art1	greek	0.3	$(X_7 = t_9)$
t_{10}	ICS-MS	art2	punic	0.8	$(X_8 = t_{10})$
t_{11}	XRF	art2	egyptian	0.5	$(X_9 = t_{11})$

Figure 1: Datentabellen *Arte*, *ArteExp* und *ArteMat* des fortlaufenden Beispielszenarios (die unterstrichenen Attribute kennzeichnen den jeweiligen Ereignisschlüssel (siehe Kap. (2)))

In dem stark vereinfachten Beispielszenario werden die deterministische Tabelle *Artefakte* (*Arte*) und die zwei probabilistischen Tabellen *Artefakte klassifiziert bei Experten*[2] (*ArteExp*) und *Artefakte klassifiziert bei Material* (*ArteMat*) verwendet, siehe Abb. (1). In der Datentabelle *Arte* werden Informationen über mehrere Artefakten gespeichert, welche während einer archäologischen Ausgrabung gefunden worden sind. Dabei wird mittels der *Sondage*-Nummer (Attribut *sond*) die geographische Fundstelle eines Artefaktes beschrieben.

Zusätzlich geben mehrere Spezialisten verschiedene Expertisen über die Ursprungskultur eines Artefakts (Attribut *culture*) ab, siehe Tabelle *ArteExp*. Diese Einschätzungen werden mit einem Konfidenzwert annotiert (Attribut *conf*),

[1] http://www.dainst.org/en/project/cisar/

[2] Die Spalten *tid*, *conf* und $ET_{(...)}$ gehören nicht zu den eigentlichen Datentabellen. Mittels der Spalte *tid* werden Tupel adressiert. Die Bedeutung von *conf* und $ET_{(...)}$ wird in den nächsten Abschnitten erläutert.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

```
select aid, type, culture
from    (  select aid, culture
           from ArteExp
         union
           select aid, culture
           from ArteMat
        ) origin
     inner join
        ( select *
          from Arte
        ) prop
     on ( origin.aid = prop.aid )
```

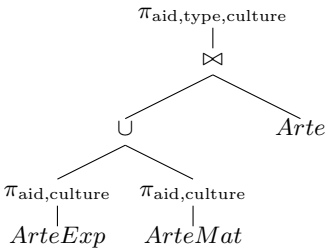


Figure 2: Beispielanfrage Q_e als QSQL2-Anfrage und als abstrahierte Algebraanfrage

welche die Wahrscheinlichkeit ausdrückt, dass das entsprechende Artefakt zu der bestimmten Kultur gehört. Neben den subjektiven Expertenmeinungen werden auch objektive Methoden einbezogen. Diese *archäometrischen Methoden* (z.B. XRF und ICS-MS[3]) basieren auf einer Materialanalyse. In Kombinationen mit den Fundstellen und dem Artefaktalter kann die Materialzusammensetzung wichtige Hinweise auf die Ursprungskultur geben, welche dann ebenfalls mittels Konfidenzwerten quantifiziert werden.

Basierend auf den eingeführten Datentabellen soll exemplarisch die folgende Anfrage Q_e bearbeitet werden: *Bestimme alle Artefakte mit ihren jeweiligen Typ und ihren möglichen Ursprungskulturen.* Um diese Anfrage zu beantworten wird sie zunächst in der Anfragesprache QSQL2 [8] formuliert, siehe Abb. (2). Anschließend wird von dem eigentlichen SQL-Syntax abstrahiert, indem sie in Form einer Algebraanfrage in den nächsten Kapiteln weiter verarbeitet wird.

Die Arbeit ist wie folgt gegliedert. In Kap. (2) werden zunächst die Grundlagen für die weiteren Betrachtungen gelegt. Danach steht der wesentliche Beitrag dieser Arbeit in Form der Ereignismuster in Kap. (3) im Mittelpunkt. In den Kap. (4) und (5) wird die Arbeit mit einer Diskussion über verwandte Arbeiten und einer Zusammenfassung abgeschlossen. Der Beweis des Satzes (1) wird im Anhang (A) geführt.

2. GRUNDLAGEN

In einer probabilistischen Datenbanken werden mehrere möglichen Datenbankinstanzen, welche auch *Welten* genannt werden, gleichzeitig verwaltet und abgefragt. Dabei wird die „reale Welt“ als unbekannt angenommen. Um diese Unsicherheit abzubilden wird ein Wahrscheinlichkeitsmaß über der Menge aller möglichen Welten vereinbart. Konkret wird hier auf die Definition von Suciu et al. [18] zurück gegriffen.

Definition 1. Angenommen werden k Relationennamen: $R_1, \ldots, R_k$. Eine *unvollständige Datenbank* ist dann eine endliche Menge von Dateninstanzen $\mathbf{W} = \{W^1, W^2, \ldots, W^n\}$, wobei jede Dateninstanz (Welt) durch $W^i = (R_1^i, \ldots, R_k^i)$ beschrieben ist. Eine *probabilistische Datenbank* ist ein Wahrscheinlichkeitsraum $\mathbf{D} = (\mathbf{W}, \mathbf{P})$ über einer unvollständigen Datenbanken $\mathbf{W}$. Damit ist $\mathbf{P} : \mathbf{W} \to [0, 1]$ eine Funktion, sodass $\sum_{W \in \mathbf{W}} \mathbf{P}(W) = 1$ gilt. Es wird vorausgesetzt, dass $\forall W \in \mathbf{W} : \mathbf{P}(W) > 0$ gilt.

Im Allgemeinen ist die Semantik des Wahrscheinlichkeitsmaß $\mathbf{P}$ nicht vordefiniert. Konkret wird hier der Spezialfall der block-independent-disjoint Datenbanken (BID) [2] benutzt, da Tupel- *und* Attributunsicherheit untersützt werden soll. Eine BID ist eine probabilistische Datenbank in der die gegebenen Tupel in Blöcke unterteilt werden. Dabei kann ein Block sich nicht über mehrere Relationen erstrecken. Es wird vereinbart, dass alle Tupel innerhalb eines Blockes mit disjunkten Tupelereignissen verbunden sind. Ein Tupelereignis beschreibt das Vorhandensein oder das Nicht-Vorhandensein eines Tupel in einer beliebigen Welt. Wegen der Disjunktheit der Tupelereignisse kann maximal ein Tupel eines Blockes in einer bestimmten Welt vorhanden sein. Dagegen sind Tupel von verschiedenen Blöcken mit gegenseitig unabhängigen Tupelereignissen assoziiert.

Für die Definition von Blöcken werden Ereignisschlüssel in Form von Attributmengen angewendet[4]. So wird ein Block wird von Tupeln gebildet, welche die gleichen Werte für die Attribute des Schlüssel haben. Anschließend wird für jeden Block eine unabhängige Zufallsvariable X_k eingeführt. Das konkrete Eintreten einer Zufallsvariable ($X_k = t_{id}$) repräsentiert ein Tupelereignis und wird quantifiziert mit dem Konfidenzwert des Tupels t_{id}, siehe die Spalten *conf* und $ET_{(\ldots)}$ in der Abb. (1)[5]. Die kombinierte Wahrscheinlichkeitsverteilung aller Blockvariablen bilden schließlich $\mathbf{P}$. Um eine Algebraanfrage auf einer probabilistischen Datenbank auszuführen, wird folgende Anfragesemantik verwendet [18].

Definition 2. Sei Q eine Algebraanfrage und $\mathbf{D} = (\mathbf{W}, \mathbf{P})$ eine probabilistische Datenbank. Die Menge aller *möglichen Antworttupel* einer Anfrage Q ist definiert als $Q_{\text{poss}(\mathbf{W})} = \{t \mid \exists W \in \mathbf{W} : t \in Q(W)\}$. Zusätzlich werden die Eintrittswahrscheinlichkeiten aller möglichen Antworten in der Funktion $\Pr_Q : Q_{\text{poss}(\mathbf{W})} \to (0, 1]$ als $\Pr_Q(t) := \sum_{W \in \mathbf{W} : t \in Q(W)} \mathbf{P}(W)$. berechnet.

Dies bedeutet, dass die Anfrage Q konzeptionell in jeder Welt separat ausgeführt wird. Dann wird die Ergebnisrelation $Q_{\text{poss}(\mathbf{W})}$ gebildet, in dem alle möglichen Antworten der verschiedenen Auswertungen gesammelt werden. Zusätzlich wird die Eintrittswahrscheinlichkeit einer möglichen Antwort durch das Aufsummieren der Welten, in der das Antworttupel in der Antwort auftritt, gebildet. Offensichtlich gibt die Def. (2) nur die Semantik der Anfrageauswertung vor. Eine einzelne Auswertung in allen Welten ist praktisch

[3]XRF und ICP-MS stehen für die *x-ray fluorescence* und die *inductively coupled plasma mass spectrometry* Methode.

[4]In dem Beispielszenario werden die Ereignisschlüssel von *Arte*, *ArteExp* und *ArteMat* als $\{aid\}$, $\{exp, aid\}$ und $\{method, aid\}$ vereinbart, siehe Abb. (1).

[5]Da die Tabelle *Arte* deterministisch ist, wird $\mathrm{P}(X_1 = t_1) = \ldots = \mathrm{P}(X_3 = t_3) = 1$ gesetzt.

nicht umsetzbar, da die Anzahl aller Welten exponentiell in Datengröße anwachsen kann.

3. EREIGNISMUSTER

In diesem Kapitel wird die praktische Auswertung einer Algebraanfrage Q auf einer probabilistischen Datenbank diskutiert. Dabei wird das Konzept der Ableitungsformeln vorgestellt und eine neue Technik zur Verwaltung von komplexen Tupelereignissen eingeführt.

3.1 Ableitungsformeln

Entsprechend Def. (2) muss zur Auswertung einer Anfrage die Ergbeniswahrscheinlichkeit $\Pr_Q(t)$ für jedes Ergebnistupel berechnet werden. Fuhr und Röllecke schlugen in [4] vor, diese Wahrscheinlichkeiten mittels von *Ableitungsformeln* ϕ_t zu berechnen. Dabei werden alle Ableitungsformeln neben dem eigentlichen Datenanteil der Tupel verwaltet. Prinzipiell ist eine Ableitungsformel ϕ_t durch eine aussagenlogische Formel gegeben, welche mittels den Zufallsvariablen $(X_k = t_{id})$[6] und den logischen Operatoren $\wedge, \vee$ und $\neg$ gebildet wird. Fuhr und Röllecke haben geschlussfolgert, dass $\Pr_Q(t)$ durch Wahrscheinlichkeit $\mathbf{P}(\phi_t \equiv \text{true})$ ermittelt werden kann, d.h. $\Pr_Q(t) = \mathbf{P}(\phi_t \equiv \text{true})$. Somit kann $\Pr_Q(t)$ durch $\mathbf{P}(\phi_t \equiv \text{true})$ berechnet werden, ohne dass über alle Welten von $\mathbf{W}$ iteriert werden muss (siehe Def. (2)). Im Folgenden wird $\mathbf{P}(\phi_t \equiv \text{true})$ durch $\mathbf{P}(\phi_t)$ abgekürzt. Die Konstruktion von ϕ_t greift auf die Struktur der betrachteten Anfrage Q zurück.

Definition 3. Angenommen Q ist eine Algebraanfrage und $\mathbf{D} = (\mathbf{W}, \mathbf{P})$ ist eine probabilistische Datenbank. Die Ableitungsformel ϕ_t ist dann wie folgt rekursiv definiert:

$$
\begin{aligned}
Q \equiv R \;&:\; \phi_t := (X_k = t_{id}), \text{wenn } t \in \text{TupBl}(k) \\
Q \equiv \sigma_c(Q_1) \;&:\; \phi_t := \phi_{t_1} \\
Q \equiv \pi_{\mathcal{A}}(Q_1) \;&:\; \phi_t := \bigvee_{\hat{t} \in Q_1, \hat{t}[\mathcal{A}] = t} \phi_{\hat{t}} \\
Q \equiv Q_1 \bowtie Q_2 \;&:\; \phi_t := \phi_{t_1} \wedge \phi_{t_2} \\
Q \equiv Q_1 \cup Q_2 \;&:\; \phi_t := \phi_{t_1} \vee \phi_{t_2} \\
Q \equiv Q_1 \setminus Q_2 \;&:\; \phi_t := \phi_{t_1} \wedge \neg(\phi_{t_2})
\end{aligned}
$$

wobei X_k eine Blockvariable ist und $\text{TupBl}(k)$ alle Tupel eines Blockes k zurück gibt (siehe Kap. (2)). Falls t_i nicht in einer Eingabeanfrage existiert (d.h. $t_i \notin Q_i$), wird $\phi_{t_i} :=$ false gesetzt (siehe Vereinigung- und Differenzoperator).

Abb. (3) zeigt die Ableitungsformeln für die Ergebnistupel der Beispielanfrage Q_e. Die Wahrscheinlichkeit $\mathbf{P}(\phi_t)$ kann mit Standardalgorithmen berechnet werden (z.B. [12, 6]). Wie man dort sieht, besitzen Ereignistupel im Allgemeinen unterschiedliche Ableitungsformeln. Mehrere Verfahren (z.B. [4, 3, 13]) müssen eine komplexe Ableitungsformel für jedes einzelne Tupel, welches innerhalb der Anfrageverarbeitung entsteht, verwalten. Praktische Anwendungsszenarien haben jedoch bereits gezeigt, dass Ableitungsformel mit einer Größe von 10 MB für ein Ergebnistupel auftreten können [14]. Dementsprechend folgt der hier entwickelte Ansatz der Argumentation von Antova et al. [1] und Das Sarma et al. [16], dass die Verwaltung und die Verarbeitung von komplexen Ableitungsformeln durch relationale Datenbanksysteme nicht zielführend sind, da solche Systeme nicht für die Verarbeitung von komplexen logischen Formeln ausgelegt sind.

[6]Ein solches Ereignis wird auch als *Basisereignis* bezeichnet.

Algorithm 1: gen(Φ^{pa}, t)

Data: Klauselmustermenge Φ^{pa}, Tupel $t \in R^{ev}$
Result: Formel in DNF kodiert als Φ_t

```
1  Φ_t := ∅;
2  foreach CP ∈ Φ^pa do
3  |   C := (true);
4  |   foreach ET_{R_i} in CP do
5  |   |   if t[ET_{R_i}] ≠ null then
6  |   |   |   C := C • t[ET_{R_i}];
7  |   |   else
8  |   |   |   C := C • (false);
9  |   |   end
10 |   end
11 |   if simpl(C) ≠ false then
12 |   |   Φ_t := Φ_t ∪ {simpl(C)};
13 |   end
14 end
15 return Φ_t;
```

3.2 Verwaltung von Ableitungsformeln mittels Ereignismustern und Basisereignismengen

Um eine logische Formel in einer strukturierten Form zu verwalten wird sie in die bekannte disjunktive Normalform (DNF) transformiert.

Definition 4. Angenommen ϕ_t ist eine Ableitungsformel. Ein *Literal* ist gegeben durch ein negiertes oder unnegiertes Basisereignis von ϕ_t, d.h. $L \equiv \neg(X_k = t_{id})$ oder $L \equiv (X_k = t_{id})$. Eine Konjunktion von Literalen wird als *Klausel* C bezeichnet, d.h. $C = L_1 \wedge \ldots \wedge L_m$. Eine Formel ϕ_t^{dnf} in DNF ist eine Disjunktion von Klauseln, d.h. $\phi_t^{\text{dnf}} = C_1 \vee \ldots \vee C_r$, sodass $\phi_t \equiv \phi_t^{\text{dnf}}$.

Da relationale Standardoperatoren zur Manipulation von DNF-Formeln verwendet werden sollen (siehe Def. (6) unten), empfiehlt es sich DNF-Formeln in Form von Tupeln und Mengen zu kodieren.

Definition 5. Um eine Formel ϕ_t^{dnf} in DNF zu kodieren, wird eine *Menge von Klauseltupeln* (bezeichnet als Φ_t) benutzt, d.h. $\Phi_t := \{C_1, \ldots, C_r\}$, wobei C_i ein Klauseltupel ist. Ein Klauseltupel $C_i = (L_1, \ldots, L_{m_i})$ korrespondiert dann mit einer Klausel $L_1 \wedge \ldots \wedge L_{m_i}$ von ϕ_t^{dnf}.

Die grundlegende Idee des hier vorgestellten Ansatzes lässt sich dann in folgenden vier Schritten beschreiben:

(i) das Bilden einer Menge von Klauselmustern (gekennzeichnet als Φ^{pa}), welche direkt von der Algebraanfrage Q abgeleitet wird (d.h. unabhängig von den aktuellen Daten in der Datenbank),
(ii) das Generieren einer Menge relevanter Basisereignissen für jedes Ergebnistupel während der relationalen Anfrageverarbeitung (verwaltet in einer erweiterten Ergebnisdatenrelation R^{ev}),
(iii) die Konstruktion einer DNF-Formel kodiert als Φ_t für jedes Ergebnistupel mittels des Generierungsalgorithmus gen(Φ^{pa}, t), $t \in R^{ev}$ und
(iv) die Berechnung von $\mathbf{P}(\phi_t^{\text{dnf}})$ mit Hilfe eines Standardalgorithmus (z.B. [12, 6]).

Bevor der Generierungsalgorithmus gen(Φ^{pa}, t) diskutiert wird, werden zunächst die Bedeutung von Φ^{pa} und R^{ev} näher beleuchtet.

	tid	aid	type	culture	ϕ_t	$\mathbf{P}(\phi_t)$
r_1	(t_4, t_6, t_8, t_1)	art1	vase fragment	roman	$([(X_4 = t_4) \vee (X_5 = t_6)] \vee (X_7 = t_8)) \wedge (X_1 = t_1)$	0.706
r_2	(t_5, t_9, t_1)	art1	vase fragment	greek	$[(X_4 = t_5) \vee (X_7 = t_9)] \wedge (X_1 = t_1)$	0.58
r_3	(t_7, t_{11}, t_2)	art2	spear head	egyptian	$[(X_6 = t_7) \vee (X_9 = t_{11})] \wedge (X_2 = t_2)$	0.8
r_4	(t_{10}, t_2)	art2	spear head	punic	$[(X_8 = t_{10}) \vee (False)] \wedge (X_2 = t_2)$	0.8

Figure 3: Die Ergebnistupel von Q_e mit ihren Ableitungsformeln (ϕ_t) und ihren Ergebniswahrscheinlichkeiten ($\mathbf{P}(\phi_t)$)

Mustermenge Φ^{pa}: Die Mustermenge Φ^{pa} besteht aus einer Menge von Klauselmustern $\{CP_1, \ldots, CP_l\}$. Ein Klauselmuster $CP = (ET_{R_{i_1}}, \ldots, ET_{R_{i_m}})$ ist wiederum gegeben durch ein Tupel von *Basisereignismustern*. Dabei symbolisiert ein Muster ET_{R_i} genau ein Basisereignis $(X_k = t_{id})$ der Basisrelation R_i. Falls z.B. das Klauselmuster $CP = (ET_{ArteExp}, ET_{ArteMat})$ betrachtet wird, verkörpert CP alle Klauseln in denen das erste Basisereignis aus der Relation $ArteExp$ stammt und das zweite Basisereignis aus $ArteMat$ genommen wird.

Basisereignisse in R^{ev}: Die Ereignisdatenrelation R^{ev} besteht aus allen Datentupeln, sowie aus jeweils einer Menge von Basisereignissen für jedes Datentupel. Dabei werden genau die Basisereignisse gespeichert, welche notwendig sind um die Ableitungsformel für ein bestimmtes Datentupel zu bilden. Zu diesem Zwecke werden die Datenrelationen durch zusätzliche Spalten erweitert, welche Basisereignisse speichern. Alle Basisereignisse einer spezifischen Zeile gehören dabei zu dem jeweiligen Datentupel. Jede neue Spalte wird mit einer Basisrelation R_i assoziiert und entsprechend mit einem Musternamen ET_{R_i} bezeichnet, siehe Abb. (1) und (4).

Algorithmus $\text{gen}(\Phi^{pa}, t)$**:** Nach der Konstruktion von Φ^{pa} und R^{ev} generiert der Algoritmus $\text{gen}(\Phi^{pa}, t)$ eine Ableitungsformel (kodiert als Φ^{pa}) für ein Tupel der Relation R^{ev}. Im Wesentlichen ersetzt der Algorithmus der Basisereignismuster mit den jeweiligen zuordenbaren Basisereignissen (siehe Zeilen 2 bis 11 in Algorithmus (1)). Die Vergleichskriterien sind durch die Musternamen ET_{R_i} und die korrespondieren Bezeichnungen der Ereignisspalten von R^{ev} gegeben. Der Operator $\bullet$ konkateniert zwei Tupel. Bevor eine erzeugte Klausel C zu der Ergebnisformel Φ_t hinzugenommen wird, werden alle Klauseln C logisch vereinfacht (siehe Zeilen 12 und 13). Hierfür werden logische Gesetze wie Idempotenz und Kontradiktion eingesetzt.

Die Klauselmustermenge Φ^{pa} und die Ereignisdatenrelation R^{ev} werden rekursiv über die Struktur der betrachteten Algebraanfrage Q definiert. Prinzipiell wird Φ^{pa} in einer Art und Weise konstruiert, dass die erzeugten Muster der grundlegenden Semantik der Def. (3) genügt und alle möglichen Ereignisse erzeugt werden können, die nötig sind um ϕ_t^{dnf} zu erzeugen. Exemplarisch wird die Ereignisdatenrelation der Beispielanfrage Q_e in Abb. (4) gezeigt.

Definition 6. Sei Q eine positive Algebraanfrage[7] und $\mathbf{D} = (\mathbf{W}, \mathbf{P})$ eine probabilistischen Datenbank. Die Klauselmustermenge Φ^{pa} und die Ereignisdatenrelation R^{ev} werden dann rekursiv mittels der folgenden Regel gebildet[8]:

$$
\begin{aligned}
Q \equiv R_i \;:\; & \Phi^{pa} := \{(ET_{R_i})\}, \\
& R^{ev} := \{t \bullet (X_k = t_{id}) \mid t \in R_i\} \\
Q \equiv \sigma_c(Q_1) \;:\; & \Phi^{pa} := \Phi_1^{pa}, R^{ev} := \sigma_c(R_1^{ev}) \\
Q \equiv \pi_{\mathcal{A}}(Q_1) \;:\; & \Phi^{pa} := \Phi_1^{pa}, \\
& R^{ev} := \pi_{\mathcal{A} \cup \{\text{all } ET_{R_i} \text{ columns}\}}(R_1^{ev}) \\
Q \equiv Q_1 \bowtie Q_2 \;:\; & \Phi^{pa} := \Phi_1^{pa} \times \Phi_2^{pa}, R^{ev} := R_1^{ev} \bowtie R_2^{ev} \\
Q \equiv Q_1 \cup Q_2 \;:\; & \Phi^{pa} := \Phi_1^{pa} \cup \Phi_2^{pa}, \\
& R^{ev} := R_1^{ev} \bowtie^{\text{full outer}} R_2^{ev}
\end{aligned}
$$

Um die endgültige Ableitungsformel ϕ_t^{dnf} für ein Ergebnistupel t zu bilden, werden die generierten Φ_t-Formeln aller Tupel in R^{ev} mit den selben Datenwerten wie das Ergebnistupel t disjunktiv verknüpft:

$$\phi_t^{\text{dnf}} := \bigvee_{\hat{t} \in R^{ev}, \hat{t}[\text{datAttr}] = t} \text{gen}(\Phi^{pa}, \hat{t}),$$

wobei datAttr die Menge aller Datenattribute der Ergebnisdatenrelation R^{ev} repräsentiert, d.h. alle Attribut ohne die jeweiligen ET_{R_i} Spalten[9].

Satz 1. Sei R^{ev} und ϕ_t^{dnf} konstruiert wie in Def. (6) angegeben, dann gilt (i) $Q_{\text{poss}(\mathbf{W})} = \pi_{\text{datAttr}}(R^{ev})$ und (ii) $\forall t \in Q_{\text{poss}(\mathbf{W})} : \Pr_Q(t) = \mathbf{P}(\phi_t^{\text{dnf}})$.

4. VERWANDTE ARBEITEN

Eine umfassende Monographie über probabilistische Datenbank wurde kürzlich von Suciu et al. [18] veröffentlicht. Daneben wurden in den letzten Jahren mehrere probabilistische Datenbanksysteme erfolgreich umgesetzt (z.B. [15, 7, 19]). In vorangegangenen Arbeiten [10, 11] des Autors wurde ein probabilistisches Daten- und Anfargemodell entworfen, welches Konzepte aus dem Gebiet des Information Retrievals mit Technologien der Datenbankwelt kombiniert. Die dabei

[7] Genau wie die Systeme [15], [19] und [7] fokussiert sich der hier vorgestellte Ansatz momentan auf Anfragen ohne Differenzoperationen.

[8] Um die DNF-Repräsentation von Def. (5) zu bewahren werden Tupel verflacht. Wenn z.B. $\Phi_1^{pa} = \{(L_1, L_2)\}$ und $\Phi_2^{pa} = \{(L_3, L_4)\}$ gegeben sind, dann ergibt sich $\Phi_1^{pa} \times \Phi_2^{pa} = \{(L_1, L_2, L_3, L_4)\}$ anstelle von $\Phi_1^{pa} \times \Phi_2^{pa} = \{((L_1, L_2), (L_3, L_4))\}$.

[9] Diese Notation von datAttr wird in der restlichen Arbeit weiter verwendet.

tid	aid	type	culture	$\mathtt{ET}_{ArteExp}$	$\mathtt{ET}_{ArteMat}$	$\mathtt{ET}_{Arte}$
(t_4, t_8, t_1)	art1	vase fragment	roman	$X_4 = t_4$	$X_7 = t_8$	$X_1 = t_1$
(t_5, t_9, t_1)	art1	vase fragment	greek	$X_4 = t_5$	$X_7 = t_9$	$X_1 = t_1$
(t_6, t_8, t_1)	art1	vase fragment	roman	$X_5 = t_6$	$X_7 = t_8$	$X_1 = t_1$
(t_7, t_{11}, t_2)	art2	spear head	egyptian	$X_6 = t_7$	$X_9 = t_{11}$	$X_2 = t_2$
(t_{10}, t_2)	art2	spear head	punic	null	$X_8 = t_{10}$	$X_2 = t_2$
$\Phi_e^{pa} = \{(ET_{ArteExp}, ET_{Arte}), (ET_{ArteMat}, ET_{Arte})\}$						

Figure 4: Die Ergebnisdatenrelation R_e^{ev} für die Beispielanfrage Q_e.

entwickelten Techniken werden in dem erweiterten probabilistischen Datenbanksystem *ProQua*[10] umgesetzt. Im Gegensatz zu anderen Systemen (z.B. [15, 7, 19]) unterstützt ProQua logik-basierte Ähnlichkeitsanfragen, sowie die Gewichtung von Teilanfragen innerhalb seiner Anfragesprache QSQL2 [8, 9].

5. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Konzept vorgestellt, welches mit der Hilfe von Ereignismustern und Basisereignismengen komplexe Tupelereignisse verwalten kann. Diese Tupelereignisse entstehen bei der Auswertung einer komplexen positiven Algebraanfrage auf einer probabilistischen BID-Datenbank. Der wesentliche Vorteil dieser Methode liegt in dem natürlichen Ablegen von Ableitungsformeln in einer strukturierten Form innerhalb einer erweiterten Datenrelation. Dadurch können die Tupelereignisse direkt mittels eines RDBMS verarbeitet werden.

Danksagung: Sebastian Lehrack wurde innerhalb der Projekte SCHM 1208/11-1 und SCHM 1208/11-2 von der Deutschen Forschungsgesellschaft unterstützt.

APPENDIX

A. BEWEIS FüR SATZ (1)

Es wird angenommen, dass $\Pr_Q(t) = \mathbf{P}(\phi_t)$, falls ϕ_t gebildet wurde wie in Def. (3) spezifiziert (siehe [4]). Somit muss gezeigt werden, dass (i) $Q_{\text{poss}(\mathbf{W})} = \pi_{\text{datAttr}}(R^{ev})$ und (ii) $\forall t \in Q_{\text{poss}(\mathbf{W})} : \phi_t \equiv \phi_t^{\text{dnf}}$.

Induktion über die Anzahl der Operatoren n von Q

- **Induktionsanfang:** $n = 1 : Q = R_i$, d.h. zu zeigen
  (i) $t \in \pi_{\text{datAttr}}(\{t \bullet (X_k = t_{id}) \mid t \in R_i\}) \overset{?}{\Leftrightarrow} t \in R_i$,
  (ii) $\forall t \in R_i : \bigvee_{\substack{\hat{t} \in \{t \bullet (X_k = t_{id}) \mid t \in R_i\}, \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(ET_{R_i}, \hat{t}) \overset{?}{\equiv} (X_k = t_{id})$,

  Beweis:

  (i) $\pi_{\text{datAttr}}(\{t \bullet (X_k = t_{id}) \mid t \in R_i\}) = R_i$ ✓
  (ii) $\forall \hat{t} \in \{t \bullet (X_k = t_{id}) \mid t \in R_i\}$: nach Konstruktion gibt es eineindeutiges $t \in R_i$, sodass $\hat{t} = t \bullet (X_k = t_{id})$

  $$\Rightarrow \forall t \in R_i : \bigvee_{\substack{\hat{t} \in \{t \bullet (X_k = t_{id}) \mid t \in R_i\}, \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(ET_{R_i}, \hat{t}) \equiv \text{gen}(ET_{R_i}, t \bullet (X_k = t_{id})) \equiv (X_k = t_{id}) \checkmark$$

[10] http://dbis.informatik.tu-cottbus.de/ProQua/

- **Induktionsschritt**: $n \to n + 1$ mit der Induktionsannahme (IA), dass für alle Anfragen mit bis zu n Operatoren gilt:

  (i) $t \in R_1^{ev} \Leftrightarrow t \in Q_1$ und
  (ii) $\forall t \in Q_1 : \bigvee_{\substack{\hat{t} \in R_1^{ev}, \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) \equiv \phi_{t_1}$.

- **Operator: $\mathbf{Q \equiv \sigma_{sc}(Q_1)}$**, d.h. zu zeigen

  (i) $t \in \pi_{\text{datAttr}}(\sigma_{sc}(R_1^{ev})) \overset{?}{\Leftrightarrow} t \in \sigma_{sc}(Q_1)$,
  (ii) $\forall t \in \sigma_{sc}(Q_1) : \bigvee_{\substack{\hat{t} \in \sigma_{sc}(R_1^{ev}), \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) \overset{?}{\equiv} \phi_{t_1}$

  Beweis:

  (i) $t \in \pi_{\text{datAttr}}(\sigma_{sc}(R_1^{ev})) \Leftrightarrow \exists \hat{t} \in R_1^{ev} \wedge \hat{t}[\text{datAttr}] = t \wedge sc(t) = \text{true} \overset{IA}{\Leftrightarrow} \hat{t}[\text{datAttr}] \in Q_1 \wedge \hat{t}[\text{datAttr}] = t \wedge sc(t) = \text{true} \Leftrightarrow t \in \sigma_{sc}(Q_1)$ ✓

  (Bed. $sc(t)$ ist nur über Attribute von datAttr definiert)

  (ii) $\forall t \in \sigma_{sc}(Q_1) \Rightarrow sc(t) = \text{true} \wedge (\hat{t}[\text{datAttr}] = t \Rightarrow sc(\hat{t}) = \text{true}) \Rightarrow$
  $\forall t \in \sigma_{sc}(Q_1) : \bigvee_{\substack{\hat{t} \in \sigma_{sc}(R_1^{ev}), \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) = \bigvee_{\substack{\hat{t} \in R_1^{ev}, \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) \overset{IA}{\equiv} \phi_{t_1}$ ✓
  (Bed. $\hat{t}[\text{datAttr}] = t$ ist strenger als $sc(t)$)

- **Operator: $\mathbf{Q \equiv \pi_{\mathcal{A}}(Q_1)}$**, d.h. zu zeigen

  (i) $t \in \pi_{\text{datAttr}}(\pi_{\mathcal{A} \cup \{\text{all ETs}\}}(R_1^{ev})) \overset{?}{\Leftrightarrow} t \in \pi_{\mathcal{A}}(Q_1)$,
  (ii) $\forall t \in \pi_{\mathcal{A}}(Q_1) : \bigvee_{\substack{\hat{t} \in \pi_{\mathcal{A} \cup \{\text{all ETs}\}}(R_1^{ev}), \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) \overset{?}{\equiv} \bigvee_{\substack{\hat{t} \in Q_1, \\ \hat{t}[\mathcal{A}] = t}} \phi_{\hat{t}}$,

  Beweis:

  (i) $t \in \pi_{\text{datAttr}}(\pi_{\mathcal{A} \cup \{\text{all ETs}\}}(R_1^{ev})) \Leftrightarrow t \in \pi_{\mathcal{A}}(R_1^{ev}) \Leftrightarrow \exists \hat{t} : \hat{t}[\mathcal{A}] = t \wedge \hat{t} \in R_1^{ev} \overset{IA}{\Leftrightarrow} \exists \hat{t} : \hat{t}[\mathcal{A}] = t \wedge \hat{t} \in Q_1 \Leftrightarrow t \in \pi_{\mathcal{A}}(Q_1)$✓

  ($\mathcal{A}$ besteht nur aus Datenattributen, d.h. $\mathcal{A} = \text{datAttr}$)

  (ii) $\forall t \in \pi_{\mathcal{A}}(Q_1) : \bigvee_{\substack{\hat{t} \in \pi_{\mathcal{A} \cup \{\text{all ETs}\}}(R_1^{ev}), \\ \hat{t}[\text{datAttr}] = t}} \text{gen}(\Phi_1^{pa}, \hat{t}) =$

  $$\bigvee_{\substack{\hat{t} \in \pi_{\mathcal{A} \cup \{\text{all ETs}\}}(R_1^{ev}), \\ \hat{t}[\mathcal{A}] = t}} \left(\bigvee_{\substack{\dot{t} \in R_1^{ev}, \\ \dot{t}[\text{datAttr}_1] = \hat{t}}} \text{gen}(\Phi_1^{pa}, \dot{t}) \right) \overset{IA}{=}$$

$$\bigvee_{\substack{\hat{t}\in\pi_{\mathcal{A}\cup\{\text{all ETs}\}}(R_1^{ev}),\\ \hat{t}[\mathcal{A}]=t}} \phi_{\hat{t}} = \bigvee_{\substack{\hat{t}\in Q_1,\\ \hat{t}[\mathcal{A}]=t}} \phi_{\hat{t}} \checkmark$$

(da $\mathcal{A} \subseteq \text{datAttr}_1$ und Idempotenz gilt)

- **Operator: $\mathbf{Q \equiv Q_1 \bowtie Q_2}$**, d.h. zu zeigen

  (i) $t \in \pi_{\text{datAttr}}(R_1^{ev} \bowtie R_2^{ev}) \overset{?}{\Leftrightarrow} t \in Q_1 \bowtie Q_2$,
  (ii) $\forall t \in Q_1 \bowtie Q_2 : \bigvee_{\substack{\hat{t}\in R_1^{ev}\bowtie R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa} \times \Phi_2^{pa}, \hat{t}) \overset{?}{\equiv} \phi_{t_1} \wedge \phi_{t_2}$

  Beweis:

  (i) $t \in \pi_{\text{datAttr}}(R_1^{ev} \bowtie_{jc} R_2^{ev}) \Leftrightarrow \exists t_1, t_2 : t = t_1[\text{datAttr}_1] \bullet t_2[\text{datAttr}_2] \wedge jc(t_1, t_2) = \text{true} \Leftrightarrow t_1 \in \pi_{\text{datAttr}_1}(R_1^{ev}) \wedge t_2 \in \pi_{\text{datAttr}_2}(R_2^{ev}) \overset{IA}{\Leftrightarrow} t_1 \in Q_1 \wedge t_2 \in Q_2 \wedge jc(t_1, t_2) = \text{true} \Leftrightarrow t_1 \bullet t_2 \in Q_1 \bowtie_{jc} Q_2 \Leftrightarrow t \in Q_1 \bowtie Q_2 \checkmark$

  (Verbundbed. $jc(t_1, t_2)$ (nat. Verbund) bezieht sich nur auf Datenattributen, da durch Konstruktion stets gilt $\text{evAttr}_1 \cap \text{evAttr}_2 = \emptyset$)

  (ii) $\forall t \in Q_1 \bowtie Q_2 : \bigvee_{\substack{\hat{t}\in R_1^{ev}\bowtie R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa} \times \Phi_2^{pa}, \hat{t}) =$
  $\bigvee_{\substack{t_1\in R_1^{ev}, t_2\in R_2^{ev},\\ (t_1\bullet t_2)[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa} \times \Phi_2^{pa}, t_1 \bullet t_2) =$
  $\bigvee_{\substack{t_1\in R_1^{ev}, t_2\in R_2^{ev},\\ (t_1\bullet t_2)[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa}, t_1) \wedge \text{gen}(\Phi_2^{pa}, t_2) =$
  $\bigvee_{\substack{t_1\in R_1^{ev},\\ t_1[\text{datAttr}_1]=t}} \text{gen}(\Phi_1^{pa}, t_1) \wedge \bigvee_{\substack{t_2\in R_2^{ev},\\ t_2[\text{datAttr}_2]=t}} \text{gen}(\Phi_2^{pa}, t_2) \overset{IA}{=} \phi_{t_1} \wedge \phi_{t_2} \checkmark$

  (da $(t_1 \bullet t_2)[\text{datAttr}] = t \Rightarrow jc(t_1 \bullet t_2) = \text{true}$; konkatenierte Muster (erzeugt durch $\times$) drücken Konjunktion aus und Φ_i^{pa} enthält nur Basisereignismuster von t_i)

- **Operator: $\mathbf{Q \equiv Q_1 \cup Q_2}$**, d.h. zu zeigen

  (i) $t \in \pi_{\text{datAttr}}(R_1^{ev} \bowtie^{fo} R_2^{ev}) \overset{?}{\Leftrightarrow} t \in Q_1 \cup Q_2$,
  (ii) $\forall t \in Q_1 \cup Q_2 : \bigvee_{\substack{\hat{t}\in R_1^{ev}\bowtie^{fo} R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa} \cup \Phi_2^{pa}, \hat{t}) \overset{?}{\equiv} \phi_{t_1} \vee \phi_{t_2}$

  Beweis:

  (i) $t \in \pi_{\text{datAttr}}(R_1^{ev} \bowtie^{fo} R_2^{ev}) \Rightarrow t \in \pi_{\text{datAttr}_1}(R_1^{ev}) \vee t \in \pi_{\text{datAttr}_2}(R_2^{ev}) \overset{IA}{\Leftrightarrow} t \in Q_1 \vee t_2 \in Q_2 \Leftrightarrow t \in Q_1 \cup Q_2$

  (wegen der Def. eines vollen äußeren Verbundes und $\text{datAttr} = \text{datAttr}_1 = \text{datAttr}_2$)

  (ii) $\forall t \in Q_1 \cup Q_2 : \bigvee_{\substack{\hat{t}\in R_1^{ev}\bowtie^{fo} R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa} \cup \Phi_2^{pa}, \hat{t}) =$
  $\bigvee_{\substack{\hat{t}\in R_1^{ev}\vee\hat{t}\in R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \big(\text{gen}(\Phi_1^{pa}, \hat{t}) \vee \text{gen}(\Phi_2^{pa}, \hat{t})\big) =$
  $\bigvee_{\substack{\hat{t}\in R_1^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_1^{pa}, \hat{t}) \vee \bigvee_{\substack{\hat{t}\in R_2^{ev},\\ \hat{t}[\text{datAttr}]=t}} \text{gen}(\Phi_2^{pa}, \hat{t}) \overset{IA}{=} \phi_{t_1} \vee \phi_{t_2} \checkmark$

  (wegen der Def. eines vollen äußeren Verbundes und da Φ^{pa} eine disjunktiv verknüpfte Klauselkombination beschreibt)

B. REFERENCES

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[2] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowl. and Data Eng.*, 4:487–502, October 1992.

[3] R. Fink, D. Olteanu, and S. Rath. Providing support for full relational algebra in probabilistic databases. In *ICDE*, pages 315–326, 2011.

[4] N. Fuhr and T. Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. IS*, 15(1):32–66, 1997.

[5] F. Henze, H. Lehmann, and W. Langer. CISAR - A Modular Database System as a Basis for Analysis and Documentation of Spatial Information. In *CAA*, pages 228–233, 2007.

[6] R. M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[7] C. Koch. MayBMS: A System for Managing Large Uncertain and Probabilistic Databases. In *Managing and Mining Uncertain Data, ch. 6.* Springer-Verlag, 2008.

[8] S. Lehrack, S. Saretz, and I. Schmitt. QSQL2: Query Language Support for Logic-Based Similarity Conditions on Probabilistic Databases. In *RCIS*, 2012 (to appear).

[9] S. Lehrack and I. Schmitt. QSQL: Incorporating Logic-Based Retrieval Conditions into SQL. In *DASFAA*, pages 429–443, 2010.

[10] S. Lehrack and I. Schmitt. A Probabilistic Interpretation for a Geometric Similarity Measure. In *ECSQARU*, pages 749–760, 2011.

[11] S. Lehrack and I. Schmitt. A Unifying Probability Measure for Logic-Based Similarity Conditions on Uncertain Relational Data. In *NTSS*, pages 14–19, 2011.

[12] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

[13] D. Olteanu and H. Wen. Ranking Query Answers in Probabilistic Databases: Complexity and Efficient Algorithms. In *ICDE*, 2012 (to appear).

[14] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.

[15] C. Ré and D. Suciu. Managing Probabilistic Data with MystiQ: The Can-Do, the Could-Do, and the Can't-Do. In *SUM*, pages 5–18, 2008.

[16] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, page 7, 2006.

[17] F. Schaefer and A. Schulze. OpenInfRA – Storing and retrieving information in a heterogenous documentation system. In *CAA*, 2012 (to appear).

[18] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[19] J. Widom. Trio: A system for handling, uncertainty, and lineage. In *Managing and Mining Uncertain Data*, pages 113–148. Springer, 2008.

Flexible Indexierung für Ähnlichkeitssuche mit logikbasierten Multi-Feature-Anfragen

Marcel Zierenberg
Brandenburgische Technische Universität Cottbus
Institut für Informatik, Informations- und Medientechnik
Lehrstuhl Datenbank- und Informationssysteme
Postfach 10 13 44, 03013 Cottbus, Deutschland
zieremar@tu-cottbus.de

KURZFASSUNG

Ähnlichkeitssuche beschäftigt sich mit dem Auffinden ähnlicher Objekte zu einem vorgegebenen Anfrageobjekt. Die logische Kombination verschiedener Features des Anfrageobjekts erhöht dabei die Ausdruckskraft von Anfragen und führt zu besseren Anfrageergebnissen. Um eine effiziente Suche zu ermöglichen ist eine Indexierung der Datenbankobjekte nötig. Neben einer möglichst hohen Sucheffizienz spielt die Flexibilität des Indexierungsverfahrens eine entscheidende Rolle. Das in dieser Arbeit vorgestellte Indexierungsverfahren ermöglicht eine effiziente Verarbeitung von Multi-Feature-Anfragen mit beliebigen logischen Kombinationen und Gewichtungen anhand eines einzigen Index. Die Verwendung metrischer Indexierungsmethoden garantiert die Anwendbarkeit des Konzepts für ein großes Spektrum von Features und Distanzfunktionen.

Kategorien und Themenbeschreibung

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing — Indexing methods

Allgemeine Begriffe

Algorithms, Performance

Schlüsselwörter

similarity search, retrieval, nearest neighbor search, metric indexing, complex query

1. EINLEITUNG

Ähnlichkeitssuche [13] verfolgt das Ziel, in einer Menge von Objekten genau die Objekte zu finden, die einem Anfrageobjekt am ähnlichsten sind. Die (Un-)Ähnlichkeit der Objekte wird mithilfe von *Distanz-* oder *Ähnlichkeitsmaßen*[1] anhand der aus den Objekten extrahierten *Features* bestimmt. Bei einem Feature handelt es sich dabei um eine Menge von Werten, die bestimmte Eigenschaften eines Objekts charakterisieren. Die Nutzung von Features, welche die gewünschte Semantik geeignet beschreiben, ist dabei entscheidend für eine effektive Ähnlichkeitssuche.

Die Verwendung mehrerer Features und einer geeigneten Kombination dieser, erhöht die Ausdruckskraft von Anfragen und führt somit zu besseren Anfrageergebnissen [20]. *Multi-Feature-Anfragen* nutzen daher eine Vielzahl von Features für die Anfrageformulierung.

Logikbasierte Anfragemodelle, wie die *Commuting Quantum Query Language* (*CQQL* [14]) oder die *Fuzzy-Logik* [18], erlauben die Kombination mehrerer Features mithilfe boolescher Junktoren. Neben der verbesserten Ausdruckskraft von Anfragen wird hierdurch eine Verbindung von (unscharfen) Ähnlichkeitsbedingungen und (scharfen) relationalen Datenbankbedingungen ermöglicht [13]. Eine *Gewichtung* der einzelnen Anfrageatome erlaubt weiterhin eine dynamische Anpassung der Anfragen. Das Lernen dieser Gewichte anhand von Nutzerpräferenzen [19] ermöglicht eine schrittweise Verfeinerung von Anfragen in Form von *Relevance Feedback*.

Der naive Ansatz zur Umsetzung einer Ähnlichkeitssuche ist der *lineare Scan* der Datenbank, bei dem alle Distanzen zwischen Anfrageobjekt und Datenbankobjekten ermittelt werden. Für Multi-Feature-Anfragen bedeutet dies, dass für jedes einzelne Feature die Distanz zwischen Anfrage- und jedem Datenbankobjekt ermittelt wird. Anschließend werden diese *Teildistanzen* für jedes Objekt zu einer *Gesamtdistanz* aggregiert. Da die Evaluierung von Distanzfunktionen mitunter hohe CPU- und das Lesen der zugehörigen Features hohe I/O-Kosten verursachen, ist der lineare Scan für große Datenbanken mit einer Vielzahl von Objekten und Features nicht praktikabel. Stattdessen sollten *Indexierungsverfahren* genutzt werden, um eine effizientere Suche zu realisieren. Sie ermöglichen einen frühzeitigen Ausschluss von Objekten, die nicht zur Ergebnismenge gehören können, und verringern somit die Anzahl der nötigen Distanzberechnungen und I/O-Operationen.

Eine besondere Anforderung an die logikbasierte Indexierung stellt die *Flexibilität* dar. Die logische Kombination der Anfrage und auch die Anfragegewichte können sich im Rahmen von Relevance Feedback dynamisch verändern, dennoch sollte nur ein einziger Index zur Verarbeitung beliebiger Anfragen benötigt werden. Weiterhin sollte das Indexierungsverfahren unabhängig von Art und Struktur der verwendeten Features und Distanzfunktionen sein.

Hauptbeitrag dieser Arbeit ist die Entwicklung eines effizienten Indexierungsverfahrens zur Verarbeitung logikbasierter Multi-Feature-Anfragen am Beispiel von CQQL. Dazu werden spezifische Anforderungen an die logikbasierte Indexierung definiert und anhand dieser ein Indexierungsverfahren entworfen, das eine effizi-

[1] Im Folgenden gehen wir von Distanzmaßen aus.

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

Tabelle 1: Umwandlung boolescher Ausdrücke in arithmetische Formeln in CQQL

Ausdruck	arithmetische Formel
$\neg a$	$1 - a$
$a \wedge b$	$a * b$
$a \vee b$	$a + b - a * b$
$(c \wedge a) \vee (\neg c \wedge b)$	$a + b$

ente und gleichzeitig flexible Verarbeitung beliebiger logikbasierter Multi-Feature-Anfragen und eine dynamische Gewichtung dieser ermöglicht.

Die Arbeit ist wie folgt aufgebaut. Abschnitt 2 definiert die grundlegenden Begriffe und die Notationen dieser Arbeit. In Abschnitt 3 wird ein theoretisches Anwendungsbeispiel aus dem Bereich der Bildähnlichkeitssuche mithilfe logikbasierter Multi-Feature-Anfragen präsentiert. Auf die speziellen Anforderungen an Indexierungsverfahren für derartige Anfragen wird in Abschnitt 4 detailliert eingegangen. Abschnitt 5 beschäftigt sich mit dem Stand der Technik zur Indexierung. Abschnitt 6 zeigt das Konzept eines Indexierungsverfahrens und Kriterien zur Indexauswahl. Abschließend wird in Abschnitt 7 eine Zusammenfassung der Arbeit gegeben.

2. GRUNDLAGEN

Der folgende Abschnitt definiert die grundlegenden Begriffe und die Notationen, die in dieser Arbeit verwendet werden.

2.1 Nächste-Nachbarn-Suche

Ähnlichkeitsanfragen anhand von Distanzmaßen werden auch als *k-Nächste-Nachbarn-Suche* (*kNN*) bezeichnet und geben die k Objekte aus einer Datenbank zurück, deren Distanz zum Anfrageobjekt am geringsten ist.

Eine Ähnlichkeitsanfrage $\mathrm{kNN}(q)$ besteht aus einem Anfrageobjekt q aus dem Universum $\mathbb{U}$ und bezieht sich auf eine Datenbank $D = \{o_1, o_2, \ldots, o_n\} \subseteq \mathbb{U}$ von Objekten o_i. Die Distanz (Unähnlichkeit) von Objekten wird mithilfe einer Distanzfunktion $\delta : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}_{\geq 0}$ anhand der aus den Objekten extrahierten Features q' und o_i' bestimmt. Das Ergebnis der Anfrage $\mathrm{kNN}(q)$ ist dann eine (nichtdeterministische) Menge $K \subseteq D$, für die gilt: $|K| = k$ und $\forall o_i \in K, o_j \in D \setminus K : \delta(q, o_i) \leq \delta(q, o_j)$.

Bei einer kNN-Anfrage bestehend aus m Features werden die Features eines Objekts mit $q' = (q^1, q^2, \ldots, q^m)$ beziehungsweise $o_i' = (o_i^1, o_i^2, \ldots, o_i^m)$ bezeichnet. Jedem Feature wird eine eigene Distanzfunktion δ^j zugeordnet. Die Teildistanzen d_i^j aller Features werden durch eine *Aggregationsfunktion* $\mathrm{agg} : \mathbb{R}_{\geq 0}^m \mapsto \mathbb{R}_{\geq 0}$ zu einer Gesamtdistanz d_i^{agg} vereinigt. Die k nächsten Nachbarn werden dann anhand dieser Gesamtdistanz bestimmt.

2.2 CQQL

Für die Evaluation einer auf CQQL basierenden Anfrage werden boolesche Ausdrücke nach ihrer DNF-Normalisierung anhand von festgelegten Transformationsregeln in arithmetische Formeln umgewandelt [14]. Tabelle 1 zeigt die in CQQL verwendeten Transformationsregeln. Ebenso ist eine Umsetzung der Gewichtung in CQQL direkt innerhalb der booleschen Logik möglich. Dazu werden Gewichte anhand der in Tabelle 2 gezeigten Transformationsregeln in die Logik eingebettet.

2.3 Logikbasierte kNN

Im Folgenden werden die in Abschnitt 2.2 beschriebenen arithmetischen Formeln als Aggregationsfunktionen betrachtet, deren Definitions- und Wertebereich auf Ähnlichkeitswerte im Intervall [0,1] beschränkt ist: $\mathrm{agg} : [0,1]^m \mapsto [0,1]$. Hierbei steht ein Ähnlichkeitswert von 1 für die maximale Ähnlichkeit (Identität), während ein Wert von 0 die größtmögliche Unähnlichkeit darstellt.

Tabelle 2: Einbettung von Gewichten in die Logik

Ausdruck	Einbettung
$a \wedge_{\theta_1,\theta_2} b$	$(a \vee \neg\theta_1) \wedge (b \vee \neg\theta_2)$
$a \vee_{\theta_1,\theta_2} b$	$(a \wedge \theta_1) \vee (b \wedge \theta_2)$

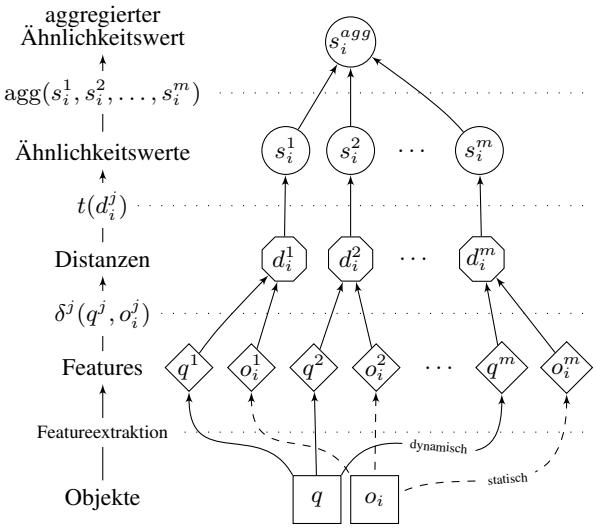


Abbildung 1: Ablauf einer logikbasierten Multi-Feature-Anfrage

Um eine Nächste-Nachbarn-Suche anhand logikbasierter Anfragen zu realisieren, müssen alle Teildistanzen d_i^j vor ihrer Aggregation in Ähnlichkeitswerte s_i^j umgewandelt werden. Die nächsten Nachbarn sind nach der Aggregation dieser Ähnlichkeitswerte dann genau die Objekte, deren aggregierte Ähnlichkeitswerte s_i^{agg} bezüglich dem Anfrageobjekt am größten sind. Abbildung 1 verdeutlicht den Suchablauf.

Beispiele für geeignete Funktionen zur Umwandlung von Distanzen in Ähnlichkeitswerte sind $t(x) = e^{-x}$ oder $t(x) = 1 - \frac{x}{\delta_{max}}$, wobei δ_{max} der maximale Distanzwert der genutzen Distanzfunktion darstellt [13].

2.4 Monotonie

Eine Aggregationsfunktion ist *monoton steigend im i-ten Argument*[2], wenn gilt:

$$x^i \leq y^i \implies \mathrm{agg}(x^1, \ldots, x^i, \ldots, x^m) \leq \mathrm{agg}(x^1, \ldots, y^i, \ldots, x^m) \quad (1)$$

Das bedeutet, dass wenn alle Parameter außer x^i festgelegt sind und x^i vergrößert wird, kann das Ergebnis der Aggregationfunktion nicht kleiner werden. Eine Aggregationsfunktion ist *global monoton steigend*, wenn sie in allen m Argumenten monoton steigend ist. Ein Beispiel ist $\mathrm{agg}(x^1, x^2) = x^1 + x^2$.

Eine Aggregationsfunktion ist *lokal monoton*, wenn sie in einem Teil ihrer m Argumente monoton steigend und in allen anderen Argumenten monoton fallend ist. Ein Beispiel ist $\mathrm{agg}(x^1, x^2) = x^1 - x^2$.

[2] *Monoton fallend im i-ten Argument* ist analog anhand des $\geq$-Operators definiert.

3. ANWENDUNGSBEISPIEL

Das folgende Anwendungsbeispiel illustriert eine logikbasierte Ähnlichkeitsanfrage anhand mehrerer Features.

Gegeben sei eine Datenbank mit Bildern verschiedener Pilze und Informationen über deren Art und Giftigkeit. Ziel sei es nun anhand eines Anfragebildes eines gesammelten Pilzes, die Art des Pilzes zu ermitteln und so zu bestimmen, ob der Pilz essbar ist. Die aus den Bildern extrahierten Features umfassen dabei aus den Pixeldaten erzeugte Farb- ($color$) und Formfeatures ($shape$) sowie aus den Metadaten stammende Informationen, wie das Datum der Bildaufnahme ($date$) oder die GPS-Koordinaten des Aufnahmeortes (gps). Zur Ermittlung der (Un-)Ähnlichkeit von Objekten werden jeweils für das Feature geeignete Distanzfunktionen genutzt, beispielsweise die *Earth Mover's Distanzfunktion* [11] für Farbsignaturen oder verschiedene *Minkowski-Distanzfunktion* (L_p*-Distanzfunktion*).

Eine Anfrage, bestehend aus nur einem Feature, genügt nicht, um eine korrekte Zuordnung der Pilzarten vorzunehmen. So kann zum Beispiel allein anhand der Farbfeatures eines Pilzes nicht immer ein Rückschluss auf dessen Art gezogen werden, wenn sich etwa die Form stark von der des Anfragebildes unterscheidet. In diesem Fall ist eine UND-Verknüpfung der Features nötig: $shape_{\approx} \wedge color_{\approx}$. Für den Fall, dass der Pilz des Anfragebildes eine für seine Art sehr untypische Form aufweist ($\neg shape_{\approx}$) und daher anhand dieser nicht zugeordnet werden kann, soll stattdessen allein anhand der GPS-Koordinaten des Bildes ($gps_{\approx}$) auf dessen Art geschlossen werden. Als zusätzliche relationale (scharfe) Bedingung sollen nur Pilze betrachtet werden, die im selben Monat gesammelt wurden, wie der Pilz des Anfragebildes ($date_{=}$). Eine logische Verknüpfung der Features eines Anfragebildes sieht dann wie folgt aus, wobei θ_1, θ_2 für Parameter stehen, die eine unterschiedliche Gewichtung der Teilbedingungen ermöglichen.

$$date_{=} \wedge ((shape_{\approx} \wedge color_{\approx}) \vee_{\theta_1,\theta_2} (\neg shape_{\approx} \wedge gps_{\approx})) \quad (2)$$

Nach einer DNF-Normalisierung und Transformation anhand der in Abschnitt 2.2 beschriebenen Transformationsregeln, ergibt sich aus dem booleschen Ausdruck (2) folgende arithmetische Formel:

$$\begin{aligned}&(\theta_1 * date_{=} * shape_{\approx} * color_{\approx}) + \\ &(\theta_2 * date_{=} * (1 - shape_{\approx}) * gps_{\approx})\end{aligned} \quad (3)$$

4. ANFORDERUNGEN

In [13] werden allgemeine Anforderungen an Indexierungsverfahren im Bereich des Multimedia Retrievals definiert. Auf Grundlage dieser werden im Folgenden die spezifischen Anforderungen für Indexierungsverfahren zur effizienten Verarbeitung von logikbasierten Multi-Feature-Anfragen vorgestellt.

Flexibilität: Multi-Feature-Anfragen setzen sich aus unterschiedlichen Features und Distanzfunktionen zusammen, das Indexierungsverfahren muss daher unabhängig von Art und Struktur der Features sein und ein breites Spektrum an Distanzfunktionen unterstützen. Die Anzahl der unterschiedlichen zur Verfügung stehenden Features ist potentiell hoch. Das Indexierungsverfahren muss daher mit einer großen Menge von Features umgehen können, auch wenn nur eine Teilmenge dieser für eine Anfrage genutzt wird. Je nach Anfrage können sich die genutzten Features unterscheiden. Ebenso sind unterschiedliche logische Kombinationen und unterschiedliche Gewichtungen der selben Features möglich. Das Indexierungsverfahren darf daher nicht nur auf eine logische Kombination zugeschnitten werden, sondern muss mit beliebigen logischen Kombinationen und Gewichtungen umgehen können.

Sucheffizienz: Die Anzahl der nötigen Berechnungen der Distanzfunktion und die Anzahl von I/O-Operationen (Seitenzugriffe) dienen als Effizienzmaß für Indexierungsverfahren. Die Grundlage für die Bewertung der Sucheffizienz bildet der Vergleich mit dem Suchaufwand des linearen Scans der Datenbank. Ein effizientes Indexierungsverfahren sollte diesen linearen Aufwand stets unterbieten. Es existieren zwar Verfahren, welche eine sehr hohe Sucheffizienz bieten, dabei aber einen nicht realisierbar hohen Speicherverbrauch verursachen (vgl. [16]). Ein geeignetes Indexierungsverfahren sollte daher einen möglichst geringen Speicherverbrauch bei gleichzeitig möglichst hoher Sucheffizienz aufweisen.

Skalierbarkeit: Ein inhärentes Problem der Ähnlichkeitssuche ist der *Fluch der hohen Dimensionen* (*FdhD* [13, 5]). Dieser bewirkt, dass die Performanz von Indexierungsverfahren mit steigender (intrinsischer) Dimensionalität[3] eines Features abnimmt. Indexierungsverfahren können den Suchaufwand des linearen Scans dann nicht mehr signifikant unterbieten oder übersteigen ihn sogar. Analog zur Erhöhung der Dimensionsanzahl bei einem Feature lässt sich der FdhD auch bei Multi-Feature-Anfragen beobachten. Die Kombination von Features bewirkt hier ebenfalls eine Erhöhung der (intrinsischen) Dimensionalität. Ein geeignetes Indexierungsverfahren für Multi-Feature-Anfragen muss daher Möglichkeiten bieten, mit dem FdhD umzugehen und möglichst ohne Effizienzeinbußen skalierbar bezüglich der (intrinsischen) Dimensionalität einzelner Features und der Kombination mehrerer Features sein.

5. STAND DER TECHNIK

Der folgende Abschnitt geht auf den Stand der Technik auf dem Gebiet der effizienten Verarbeitung von Multi-Feature-Anfragen ein. Die existierenden Verfahren werden kurz vorgestellt und ausschnittsweise hinsichtlich der in Abschnitt 4 definierten Anforderungen bewertet. Wir beschränken uns dabei auf Verfahren für die *exakte Suche* der nächsten Nachbarn und gehen nicht auf Spezialfälle wie die *approximative Suche* oder zusätzliche Anfragearten wie *getNext* (*Ranking-Anfrage*) ein.

5.1 Combiner-Algorithmen

Combiner-Algorithmen [8] kombinieren die Ergebnisse mehrerer Ähnlichkeitsanfragen zu einem aggregierten Ergebnis. Für Multi-Feature-Anfragen existiert dazu je Feature eine nach Distanz[4] zum Anfrageobjekt sortierte Liste der Datenbankobjekte.

Combiner-Algorithmen sind keine Indexierungsverfahren, sondern arbeiten auf einer darüber liegenden Ebene. Sie legen nicht fest, wie die sortierten Listen bereitgestellt werden. Um eine effiziente Suche zu ermöglichen, sollte der Zugriff über Indexierungsverfahren mit Unterstützung von getNext-Anfragen umgesetzt werden.

Combiner-Algorithmen erlauben eine dynamische Auswahl der verwendeten Features sowie unterschiedliche Aggregationsfunktionen und Gewichtungen. Aufgrund der Forderung nach globaler Monotonie kommen sie jedoch nicht für alle logikbasierten Multi-Feature-Anfragen in Frage. Formel 3 ist beispielsweise nicht global monoton steigend, da eine Erhöhung des Ähnlichkeitswerts $shape_{\approx}$ nicht in jedem Fall zu einer Erhöhung des aggregierten Ähnlichkeitswerts führt. Ein weiterer Nachteil ist die geforderte Bereitstellung der sortierten Listen, da Indexstrukturen mit effizienten getNext-Anfragen nicht immer zur Verfügung stehen und sich durch die getrennte Verwaltung jedes einzelnen Index ein Mehraufwand ergibt.

[3]Die Dimensionalität eines Features ergibt sich aus der Anzahl seiner Featurewerte. Die intrisische Dimensionalität lässt sich beispielsweise anhand der paarweisen Distanzen zwischen den Featurewerten der Datenbankobjekte abschätzen und ist dann definiert als $\rho = \mu^2/2 * \sigma^2$ [1].

[4]Combiner-Algorithmen sind ohne größere Anpassungen auch für Ähnlichkeitswerte nutzbar.

5.2 Räumliche Indexierung

Räumliche Indexierungsverfahren gehen davon aus, dass die Features in Form von Vektoren vorliegen und die euklidsche Distanzfunktion (L_2-Distanzfunktion) zur Berechnung der Unähnlichkeit zwischen den Featurevektoren verwendet wird. Da diese Beschränkung im Widerspruch zur Flexibilität steht, scheidet die Verwendung räumlicher Indexierungsverfahren aus. Dennoch soll im Folgenden auf sie eingegangen werden, da ihre Konzepte teilweise übertragbar sind.

Hierarchische Verfahren, wie zum Beispiel der *R-Baum* [9], beschreiben Mengen von Objekten durch geometrische Regionen (*Cluster*). Aufgrund des Fluchs der hohen Dimensionen sinkt die Sucheffizienz dieser Verfahren jedoch bereits ab einer Dimensionsanzahl der Featurevektoren von 10-20 unter die des linearen Scans [13]. Im Folgenden stellen wir daher lediglich den nicht-hierarchischen Ansatz der VA-Datei vor.

5.2.1 VA-Datei

Die *Vektor-Approximations-Datei* (*VA-Datei*) [17] ist ein nicht-hierarchisches Verfahren und akzeptiert den FdhD in dem Sinne, dass sie statt Cluster zu bilden, direkt einen linearen Scan der Datenbank durchführt. Sie setzt dazu einen *Filter-Refinement*-Ansatz auf Basis kompakter Bitsignaturen ein. Ziel dieses Ansatzes ist es in der Filterphase, anhand der durch Bitsignaturen approximierten Objekte, Distanzgrenzen zu ermitteln und mithilfe dieser möglichst viele Objekte von der weiteren Suche auszuschließen. Die exakte Distanz muss in der Verfeinerungsphase dann lediglich für die Objekte berechnet werden, die in der Filterphase nicht ausgeschlossen werden konnten.

Die Sucheffizienz des Verfahrens ergibt sich daraus, dass die Bitsignaturen in der Filterphase sequentiell aus *Signaturdateien* gelesen werden und durch den Ausschluss von Objekten die Anzahl teurer, wahlfreier Zugriffe in der Verfeinerungsphase verringert wird. Für Multi-Feature-Anfragen ist eine Anpassung der VA-Datei nötig.

5.2.2 GeVAS

GeVAS [2] ist eine Erweiterung der VA-Datei für Multi-Feature-Anfragen und erlaubt eine dynamische Auswahl der in der Anfrage verwendeten Features aus einer großen Menge vorhandener Features. Für jedes Feature wird dazu eine separate VA-Datei erzeugt, wobei die Reihenfolge der Objekte in allen Signaturdateien gleich ist. Bei einer Multi-Feature-Anfrage werden nun nur die Signaturdateien der Features parallel abgearbeitet, die tatsächlich in der Anfrage eingesetzt werden. Für jedes einzelne Feature eines Objekts werden Distanzgrenzen ermittelt und dynamisch zu aggregierten Distanzgrenzen zusammengefasst. Der Ausschluss von Objekten geschieht in der Filterphase anhand dieser aggregierten Distanzgrenzen. Voraussetzung für die korrekte Aggregation von Distanzgrenzen ist, dass die Aggregationsfunktion global monoton steigend ist. Diese Forderung lässt sich jedoch so abschwächen, dass beliebige, logikbasierte Multi-Feature-Anfragen ermöglicht werden (siehe Abschnitt 6.1).

Liegen die einzelnen VA-Dateien auf der gleichen Festplatte (HDD) ergeben sich für GeVAS Effizienzprobleme beim Lesen der Daten vom Sekundärspeicher, da statt jede VA-Datei einzeln sequentiell zu lesen, der Lesekopf der Festplatte zwischen den verschiedenen VA-Dateien hin und her springen muss [13].

5.3 Metrische Indexierung

Metrische Indexierungsverfahren stellen keine Anforderungen an die Art der Featuredaten. Im Gegensatz zu räumlichen Indexierungsverfahren erlauben sie daher auch die Indexierung von Features bei denen es sich nicht um Vektoren handelt (zum Beispiel textuelle Daten) oder die nicht die euklidsche Distanzfunktion verwenden. Sie erfordern lediglich das Vorliegen einer Metrik.

Eine *Metrik* δ ist eine Distanzfunktion, für die folgenden Eigenschaften für alle $o_i, o_j, o_k \in \mathbb{U}$ gelten: $\delta(o_i, o_j) > 0$ für $o_i \neq o_j$ (Positivität), $\delta(o_i, o_i) = 0$ (Selbstidentität), $\delta(o_i, o_j) = \delta(o_j, o_i)$ (Symmetrie) und $\delta(o_i, o_k) \leq \delta(o_i, o_j) + \delta(o_j, o_k)$ (Dreiecksungleichung).

Der Ausschluss von Objekten wird mithilfe der Dreiecksungleichung erreicht, die es ermöglicht, untere und obere Grenzen bezüglich der Distanz von Anfrageobjekt und Datenbankobjekten zu bestimmen. Die Grenzen können effizient anhand von vorberechneten Distanzen zu einem oder mehreren Referenzobjekten[5] ermittelt werden. Die untere und die obere Grenze für die Distanz $\delta(q, o_i)$ und ein Referenzobjekt p sind wie folgt definiert:

$$\underbrace{|\delta(q,p) - \delta(p,o_i)|}_{\delta_{lb}(q,o_i)} \leq \delta(q,o_i) \leq \underbrace{\delta(q,p) + \delta(p,o_i)}_{\delta_{ub}(q,o_i)} \quad (4)$$

Analog zu räumlichen Indexierungsverfahren lässt sich zwischen hierarchischen (*M-Baum* [7]) und nicht-hierarchischen (*AESA* [16]) metrischen Indexierungsverfahren unterscheiden. Eine umfassende Übersicht metrischer Indexierungsverfahren bietet zum Beispiel das Lehrbuch von Samet [12].

Der Großteil der existierenden metrischen Indexierungsverfahren ist auf die Indexierung anhand einer einzigen Distanzfunktion ausgelegt. Die Forderung nach der Unterstützung beliebiger logischer Kombinationen kann daher nicht erfüllt werden. *Multi-Metrische Indexierungsverfahren* [4] ermöglichen die Indexierung auf Grundlage einer dynamischen Kombination mehrerer Distanzfunktionen zu einer Aggregationsfunktion. Sie unterstützen dadurch mit einem einzigen Index unterschiedliche Gewichtungen der gleichen Aggregationsfunktion. Da es sich bei der Aggregationsfunktion jedoch um eine Metrik handeln muss, ist diese auf die gewichtete Summe metrischer Distanzfunktion beschränkt. Für logikbasierte Multi-Feature-Anfragen sind diese Verfahren daher nur eingeschränkt anwendbar.

5.3.1 M^2-Baum

Der *M^2-Baum* [6] ist eine Erweiterung des M-Baums für Multi-Feature-Anfragen. Statt die Distanzen bezüglich aller Features bereits bei der Indexierung zu aggregierten Distanzen zusammenzufassen, werden die Distanzgrenzen bei der Anfrage dynamisch für jedes einzelne Feature abgeschätzt. Diese Grenzen werden anschließend in Ähnlichkeitswerte umgewandelt und zu aggregierten Ähnlichkeitsgrenzen kombiniert (vergleiche aggregierte Distanzgrenzen bei GeVAS). Der Vorteil bei diesem Vorgehen ist, dass die metrischen Eigenschaften in diesem Fall nicht für die Aggregationsfunktion gelten müssen, sondern nur für jede zugrundeliegende Distanzfunktionen.

Der M^2-Baum verlangt die lokale Monotonie der Aggregationsfunktion. Die arithmetische Formel 3 erfüllt jedoch auch diese Eigenschaft nicht. Analog zu GeVAS lässt sich die Monotonie-Eigenschaft aber auch hier so abschwächen, dass beliebige, logikbasierte Multi-Feature-Anfragen möglich werden (siehe Abschnitt 6.1). Der M^2-Baum erlaubt somit beliebige, logische Kombinationen und eine dynamische Auswahl der tatsächlich genutzten Features. Da es sich beim M^2-Baum um ein hierarchisches Verfahren handelt, nimmt die Sucheffizienz jedoch aufgrund des Fluchs der hohen Dimensionen mit steigender intrinsischer Dimensionalität der Features stärker ab, als bei nicht-hierarchischen Verfahren [13].

[5] Für Referenzobjekte existieren unterschiedliche Benennungen in der Literatur, wie zum Beispiel *Routing-*, *Focus-*, *Vantage-* oder *Pivot*-Objekt, die das gleiche Konzept widerspiegeln.

6. KONZEPT

Dieser Abschnitt beschreibt das Konzept eines Indexierungsverfahrens zur effizienten Verarbeitung logikbasierter Multi-Feature-Anfragen. Dazu wird zuerst auf die Berechnung aggregierter Distanzgrenzen eingegangen. Die Übertragung des GeVAS-Ansatzes auf den metrischen Raum stellt den Kern des Konzepts dar. Wir wählen GeVAS aufgrund seiner Flexibilität im Bezug auf Featureanzahl und -auswahl sowie aufgrund seiner besseren Skalierbarkeit im Bezug auf die Dimensionalität als hierarchische Verfahren. Zusätzliche Anpassungen, wie die direkte Berechnung exakter Distanzen für eine Teilmenge der Features, dienen dazu, die Sucheffizienz des entworfenen Verfahrens bei steigender Featureanzahl zu verbessern.

6.1 Aggregierte Distanzgrenzen

Die korrekte Berechnung aggregierter Distanzgrenzen[6] hängt von der Monotonie der Aggregationsfunktion ab. Zur Berechnung der oberen Distanzgrenze einer global monoton steigenden Aggregationsfunktion müssen die oberen Grenzen aller Teildistanzen in die Aggregationsfunktion eingesetzt werden [2].

Für die obere Distanzgrenze lokal monotoner Aggregationsfunktionen kommt es auf die Monotonie der einzelnen Argumente an. Bei monoton steigenden Argumenten muss die obere Distanzgrenze und bei monoton fallenden Argumenten die untere Distanzgrenze eingesetzt werden [6].

Die arithmetische Formel (3) ist nicht lokal monoton. Jedoch liegt eine Monotonie vor, für die wir den Begriff *fixe Monotonie* einführen. Hierbei hängt die Monotonie eines Arguments der Aggregationsfunktion von den Wertebelegungen der anderen Argumente ab. Eine fix monotone Funktion kann also in einem Argument für bestimmte Wertebelegungen monoton fallend und für alle andere Wertebelegungen monoton steigend sein. In Formel (3) ergibt sich die fixe Monotonie daraus, dass das Argument $shape_{\approx}$ in einem Teil der Formel negiert und in einem anderen Teil nicht-negiert auftritt.

Die aggregierte obere Distanzgrenze für fix monotone Funktionen ergibt sich durch das Einsetzen aller möglichen Kombinationen von oberen und unteren Grenzen in die Aggregationsfunktion und einer Auswahl des maximalen Ergebnisses.

$$C = \{d_{lb}^1, d_{ub}^1\} \times \cdots \times \{d_{lb}^m, d_{ub}^m\} \tag{5}$$

$$d_{ub}^{agg} = \max_{c \in C} \mathrm{agg}(c) \tag{6}$$

Wie zuvor erwähnt, lässt sich die Berechnung der aggregierten Grenzen in GeVAS und M^2-Baum entsprechend anpassen.

Es kann gezeigt werden, dass alle mithilfe von CQQL erzeugten arithmetischen Formeln eine der beschriebenen Monotonien erfüllen. Die Monotonie kann dabei allein anhand der Syntax der Anfrage bestimmt werden. Auf die Darstellung der Beweise dieser Eigenschaften wird an dieser Stelle aus Platzgründen verzichtet.

6.2 Metrisches Filter-Refinement

Im Gegensatz zu GeVAS werden die Signaturen beim *metrischen Filter-Refinement* nicht anhand der Featuredaten der Datenbankobjekte ermittelt, sondern ergeben sich aus den Distanzen der Datenbankobjekte zu Referenzobjekten. Jede Signaturdatei besteht daher aus den Bitsignaturen der Distanzen jedes Datenbankobjekts zu einer Menge von Referenzobjekten. Die Reihenfolge der Datenbankobjekte ist dabei in allen Signaturdateien gleich. Die kNN-Suche verläuft analog zu GeVAS, wobei der Ausschluss von Objekten jedoch anhand aggregierter Ähnlichkeitsgrenzen stattfindet.

[6]Die Beschreibungen lassen sich analog auf Ähnlichkeitsgrenzen anwenden.

Für die Auswahl geeigneter Referenzobjekte stehen verschiedene Verfahren zur Verfügung, darunter die zufällige Auswahl oder die inkrementelle Auswahl entfernter Objekte [3]. Um möglichst enge Grenzen zu garantieren, nutzt jedes Datenbankobjekt eine dynamisch ausgewählte Teilmenge seiner nächsten Referenzobjekte.

Bei der Indexerzeugung werden für einen repräsentativen Ausschnitt der Datenbank die paarweisen Distanzen je Feature bestimmt. Ein *Equi-Height-Histogramm* [10] dieser Distanzen wird genutzt um Distanzintervalle für jedes Feature zu berechnen. Diese Distanzintervalle dienen dann der Quantisierung der Distanzen zwischen Referenzobjekten und Datenbankobjekten. Statt also zur Indexierung exakte Distanzen zu speichern, werden die exakten Distanzen durch nummerierte Distanzintervalle repräsentiert (vgl. [4]). Die kompakte Darstellung dieser Nummern durch Bitsignaturen verringert den Speicherverbrauch und erhöht gleichzeitig die Sucheffizienz in der Filterphase, da weniger Daten von der Festplatte gelesen werden müssen. Der Approximationsfehler der Distanzgrenzen steigt jedoch durch die Verwendung von Distanzintervallen. Die Festlegung der Anzahl an Bits pro Signatur entspricht daher der Steuerung der Genauigkeit der Distanzintervalle.

6.3 Lesefenster

Dem in Abschnitt 5.2.2 erläuterten GeVAS-Problem der sinkenden Sucheffizienz bei der Ablage aller Signaturedateien auf einer Festplatte begegnen wir mit der Einführung eines *Lesefensters*. Statt für jedes Objekt parallel auf alle genutzten Signaturdateien zuzugreifen, wird jede Signaturdatei einzeln in Größe des Lesefensters ausgelesen. Der Vorteil dabei ist, dass längere sequentielle Lesephasen entstehen und weniger Sprünge zwischen den Signaturdateien stattfinden. Allerdings müssen die gelesenen Daten jeweils solange im Hauptspeicher gehalten werden, bis alle genutzten Signaturdateien in Größe des Lesefensters abgearbeitet wurden.

6.4 Begrenzter Hauptspeicher

Ein Problem des Filter-Refinements ist, dass alle Objekte, die in der Filterphase nicht ausgeschlossen werden können, bis zur Verfeinerungsphase in einer nach unterer Distanzgrenze sortierten Kandidatenliste gehalten werden müssen. Für große Datenbanken kann es jedoch vorkommen, dass der vorhandene Hauptspeicher dazu nicht ausreicht. Ein weiteres Lesefenster ermöglicht daher die Einhaltung von festen Grenzen für die Hauptspeichernutzung.

Nach einem in [15] vorgeschlagenen Prinzip werden Filter- und Verfeinerungsphase verschränkt. Die Filterphase wird gestoppt, sobald eine festgelegte Speichergrenze erreicht ist. In der nun folgenden Verfeinerungsphase wird die Kandidatenliste abgearbeitet, bis k vorläufige nächste Nachbarn ermittelt wurden. Damit diese k Objekte nicht verloren gehen, werden sie abschließend in die zuvor geleerte Kandidatenliste eingefügt, bevor die Filterphase wieder an der abgebrochenen Stelle fortgesetzt wird. Der Effizienznachteil dieses Vorgehens ist, dass das sequentielle Lesen der Filterphase regelmäßig unterbrochen wird und durch einen wahlfreien Zugriff wieder fortgesetzt werden muss.

6.5 Steigende Featureanzahl

Steigt die Anzahl der in der Anfrage genutzten Features, steigt der Approximationsfehler bei der Abschätzung der aggregierten Ähnlichkeitsgrenzen, da alle Teildistanzen in Form von Distanzgrenzen eingehen. Für Anfragen mit vielen Features bedeutet dies, dass sich die Anzahl der Objekte, die anhand dieser Grenzen ausgeschlossen werden können, verringert und die Sucheffizienz des Verfahrens sinkt.

Um diesem Problem zu begegnen, werden bei steigender Featureanzahl, statt Distanzgrenzen für jedes einzelne Feature zu nutzen, nur noch für eine Teilmenge der verwendeten Features Di-

stanzgrenzen abgeschätzt. Für alle anderen Features werden direkt die exakten Distanzen berechnet. Aus der exakten Berechnung von Teildistanzen ergibt sich ein Mehraufwand gegenüber der Nutzung von Distanzgrenzen. Dieser Mehraufwand kann jedoch ausgeglichen werden, wenn durch diese exakte Berechnung genügend zusätzliche Objekte ausgeschlossen werden können.

Dieses Prinzip lässt sich in der Filterphase anwenden um mehr Objekte auszuschließen. Analog zum sequentiellen Lesen der Signaturdateien müssen die Features in diesem Fall ebenfalls sequentiell gelesen werden, um die Sucheffizienz in der Filterphase zu garantieren.

Das gleiche Prinzip kann in der Verfeinerungsphase genutzt werden, um die Suche früher und mit weniger exakten Distanzberechnungen abbrechen zu können. Statt für ein Objekt in der Verfeinerungsphase direkt alle Teildistanzen auf einmal exakt zu berechnen und zu aggregieren, wird schrittweise vorgegangen. Jedes Mal, wenn ein Objekt in der Verfeinerungsphase am Anfang der Kandidatenliste steht, für das noch nicht alle Teildistanzen berechnet wurden, wird eine Teildistanz berechnet und das Objekt anhand der neuen (genaueren) aggregierten Ähnlichkeitsgrenze wieder in die Kandidatenliste einsortiert. Ein Vorteil ergibt sich dann, wenn der Ausschluss eines Objekts von nur wenigen Teildistanzen abhängt. Bei einer geeigneten Reihenfolge der berechneten Teildistanzen müssen dann nur diese diskriminierenden Distanzen exakt berechnet werden, die Berechnung aller weiteren Teildistanzen kann gespart werden.

Die Auswahl der Features, für die exakte Distanzen berechnet werden, erfolgt statisch (zur Indexierungszeit) oder dynamisch (zur Anfragezeit) nach unterschiedlichen Kriterien, wie der (intrinsischen) Dimensionalität der Features oder der Berechnungsdauer der zugehörigen Distanzfunktion.

6.6 Indexauswahl

Die Sucheffizienz des beschriebenen Verfahrens hängt besonders von der Anzahl der verwendeten Features und der daraus resultierenden (intrinsischen) Dimensionalität ab. Hierarchische Verfahren können bei niedriger (intrinsischer) Dimensionalität, aufgrund der hohen Lokalität der Daten, größere Mengen von Objekten auf einmal ausschließen und dadurch effizienter als nicht-hierarchische Verfahren sein. Ein Vergleich des entworfenen Konzepts mit dem angepassten M^2-Baum dient daher zur Bestimmung der Schnittpunkte bezüglich der Sucheffizienz beider Ansätze. Eine Selektion des effizienteren Index kann dann entweder bereits bei der Indexerzeugung oder erst zur Anfragezeit anhand der in der Anfrage genutzten Features und ihrer (intrinsischen) Dimensionalität stattfinden. Überschreitet die (intrinsische) Dimensionalität eine bestimmte Grenze, ist unter Umständen ein Rückfall auf den linearen Scan sinnvoll.

7. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Konzept zur flexiblen Indexierung für Ähnlichkeitssuche mit logikbasierten Multi-Feature-Anfragen vorgestellt. Dazu wurden die spezifischen Anforderungen an geeignete Indexierungsverfahren definiert und der Stand der Technik im Bezug auf diese Anforderungen analysiert. Das entwickelte Konzept zur Indexierung basiert auf einer Übertragung und Anpassung des GeVAS-Ansatzes auf den metrischen Raum. Der Index ist dadurch unabhängig von der genutzten logischen Kombination, der Art und Struktur der verwendeten Features und unterstützt eine Vielzahl unterschiedlicher Features und Distanzfunktionen zur Berechnung der Unähnlichkeit.

Als zukünftige Arbeiten verbleiben Teile der Implementierung, die Bestimmung der optimalen Indexparameter und die Evaluation des Konzeptes anhand von synthetischen und realen Daten. Die Unterstützung von Multi-Objekt-Anfragen sowie von Distanzfunktionen die keine Metriken sind, stellen weitere Herausforderungen dar.

Literatur

[1] Stanislav Barton u. a. “Estimating the indexability of multimedia descriptors for similarity searching”. In: *Adaptivity, Personalization and Fusion of Heterogeneous Information*. RIAO ’10. 2010, S. 84–87.

[2] Klemens Böhm u. a. “Fast Evaluation Techniques for Complex Similarity Queries”. In: *Proceedings of the 27th International Conference on Very Large Data Bases*. VLDB ’01. 2001, S. 211–220.

[3] Benjamin Bustos, Gonzalo Navarro und Edgar Chávez. “Pivot selection techniques for proximity searching in metric spaces”. In: *Pattern Recogn. Lett.* 24 (14 2003), S. 2357–2366.

[4] Benjamin Bustos und Tomáš Skopal. “Dynamic similarity search in multi-metric spaces”. In: *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*. MIR ’06. 2006, S. 137–146.

[5] Edgar Chávez u. a. “Searching in metric spaces”. In: *ACM Comput. Surv.* 33 (3 2001), S. 273–321.

[6] Paolo Ciaccia und Marco Patella. “The M^2-tree: Processing Complex Multi-Feature Queries with Just One Index”. In: *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*. 2000.

[7] Paolo Ciaccia, Marco Patella und Pavel Zezula. “M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”. In: *VLDB’97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. Hrsg. von Matthias Jarke u. a. 1997, S. 426–435.

[8] Ronald Fagin, Amnon Lotem und Moni Naor. “Optimal aggregation algorithms for middleware”. In: *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS ’01. 2001, S. 102–113.

[9] Antonin 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. 1984, S. 47–57.

[10] Yannis Ioannidis. “The history of histograms (abridged)”. In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB ’03. 2003, S. 19–30.

[11] Yossi Rubner, Carlo Tomasi und Leonidas J. Guibas. “The Earth Mover’s Distance as a Metric for Image Retrieval”. In: *Int. J. Comput. Vision* 40 (2 2000), S. 99–121.

[12] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. 2005.

[13] Ingo Schmitt. *Ähnlichkeitssuche in Multimedia-Datenbanken - Retrieval, Suchalgorithmen und Anfragebehandlung*. 2005.

[14] Ingo Schmitt. “QQL: A DB&IR Query Language”. In: *The VLDB Journal* 17 (1 2008), S. 39–56.

[15] Ingo Schmitt und Sören Balko. “Filter ranking in high-dimensional space”. In: *Data Knowl. Eng.* 56 (3 2006), S. 245–286.

[16] Enrique Vidal. “An algorithm for finding nearest neighbours in (approximately) constant average time”. In: *Pattern Recognition Letters* 4.3 (1986), S. 145 –157.

[17] Roger Weber, Hans-Jörg Schek und Stephen Blott. “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces”. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB ’98. 1998, S. 194–205.

[18] Lofti A. Zadeh. “Fuzzy Logic”. In: *Computer* 21 (1988), S. 83–93.

[19] David Zellhöfer und Ingo Schmitt. “A preference-based approach for interactive weight learning: learning weights within a logic-based query language”. In: *Distributed and Parallel Databases* 27 (1 2010), S. 31–51.

[20] David Zellhöfer und Ingo Schmitt. “Approaching Multimedia Retrieval from a Polyrepresentative Perspective”. In: *Adaptive Multimedia Retrieval. Context, Exploration, and Fusion*. Hrsg. von Marcin Detyniecki u. a. Bd. 6817. Lecture Notes in Computer Science. 2011, S. 46–60.

Challenges in Finding an Appropriate Multi-Dimensional Index Structure with Respect to Specific Use Cases

Alexander Grebhahn
Institute of Technical and Business Information Systems
University of Magdeburg
grebhahn@st.ovgu.de

David Broneske
Institute of Technical and Business Information Systems
University of Magdeburg
dbronesk@st.ovgu.de

Martin Schäler
Institute of Technical and Business Information Systems
University of Magdeburg
schaeler@ovgu.de

Reimar Schröter
Institute of Technical and Business Information Systems
University of Magdeburg
rschroet@st.ovgu.de

Veit Köppen
Center for Digital Engineering
University of Magdeburg
vkoeppen@ovgu.de

Gunter Saake
Institute of Technical and Business Information Systems
University of Magdeburg
saake@ovgu.de

ABSTRACT

In recent years, index structures for managing multi-dimensional data became increasingly important. Due to heterogeneous systems and specific use cases, it is a complex challenge to find an appropriate index structure for specific problems, such as finding similar fingerprints or micro traces in a database. One aspect that should be considered in general is the dimensionality and the related *curse of dimensionality*.

However, dimensionality of data is just one component that have to be considered. To address the challenges of finding the appropriate index, we motivate the necessity of a *framework* to evaluate indexes for specific use cases. Furthermore, we discuss core components of a framework that supports users in finding the most appropriate index structure for their use case.

Keywords

index structures, evaluation, multi-dimensional data

1. INTRODUCTION

In the last years, data storage and management in computer-aided systems became more advanced, because of an increasing amount of unstructured data being stored. For example, in multimedia databases images or videos are stored and analyzed to find similar data items. A special use case is the Digi-Dak Database Project[1], where multi-dimensional feature vectors of fingerprints and micro traces are stored in a database. To manage these data items, methods are required to handle unstructured data in an appropriate way. It is possible to extract feature vectors from an item to manage the data in a compressed and meaningful way. For managing these feature vectors, multi-dimensional index structures can be used. In general, the question arises, which index structure supports managing data best. Throughout this paper, index structure performance describes suitability with respect to a specific use case. However, we analyze core aspects that have to be considered, if trying to answer this for a specific use case. In order to achieve a reconstructible and valid comparison, we present the idea of a framework that allows the comparison of different index structures in a homogeneous test environment.

This paper is organized as follows: In Section 2, we give a short overview of basic components that have to be considered for evaluating the performance of index structures. Within Section 3, we give an overview of additional challenges, which have to be handled by using index structures in a specific use case. Finally, in Section 4, we present core components that a framework needs for quantitatively evaluation of multi-dimensional index structures with respect to different use cases.

[1]https://omen.cs.uni-magdeburg.de/digi-dak

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

2. BASIC CHALLENGES

Querying multi-dimensional data in an efficient way is a complex challenge. Within the last decades, new index structures are proposed and existing once are improved to solve this challenge. Regarding a specific use case, it is not suitable to consider an index structure in isolation. Additionally data properties, used query types, and underlying distance metrics have to be taken into account. In this section, we give a short overview of these four basic challenges.

2.1 Data Properties

Characteristics of data cause main challenges of querying data within a database system. For instance, data dimensionality has to be considered, because existing index structures are generally effected by the curse of dimensionality [7, 23]. As a result, index structures, that are suitable for a small number of dimensions are not necessarily suitable for a larger amount of dimensions. An additional important property is the data distribution, because some index structures are more practicable for clustered data than others.

Furthermore, value domain and the type of the data has to be considered.

2.2 Query Types

Based on the work of Böhm et al. [8], query types can be categorized into two groups: ϵ-similarity queries and Nearest-Neighbor-similarity (NN-similarity) queries. The former describes a query, resulting in a set of data points being situated in a defined ϵ-distance to the query point, whereas the latter results in a data point being the nearest item to the query point. Describing these two groups, the ϵ-similarity and NN-similarity has to be defined.

Definition: ϵ-similarity Query.

Two data points p_1 and p_2 are ϵ-similar if and only if $d(p_1, p_2) \leq \epsilon$. The function d defines a similarity measure for two points. In literature, similarity measures are often replaced by distance metrics, which we review in Section 2.3. For finding all points in the data base being ϵ-similar, an ϵ-similarity query is executed. A special case of the ϵ-similarity is represented for $\epsilon = 0$, because this implies two identical points and an exact match is executed [8].

Definition: NN-similarity Query.

The data point p_1 is NN-similar to p_2 with respect to a data base of points DB if and only if $\forall p \in DB, p \neq p_1 : d(p_2, p_1) \leq d(p_2, p)$. For NN-similarity queries, all points in a database are retrieved that are NN-similar to the query point. An extension to the NN-similarity query is presented, when instead of a nearest neighbor, k nearest neighbors have to be retrieved. In this paper, we call the resulting query k-NN query.

Apart from the mentioned similarity range query, window queries are common queries and often called range queries in literature [24]. These window queries are defined by intervals for every queried dimension.

2.3 Distance Metrics

To execute similarity queries, we require a function computing the similarity of two data items. To this end, similarity for equal points is 1 whereas the maximum dissimilarity is expressed by 0. Equivalent information is delivered from distance metrics, whereupon two data items are more similar, the smaller their distance is.

The most common distance metrics are Minkowsky class metrics, also called L_p distance metrics. The distance of two data items x and y is computed by:

$$L_p(x, y) = \Big(\sum_{i=1}^{d} (x_i - y_i)^p \Big)^{1/p}.$$

By choosing different values for p, different representatives of this class are produced. For $p = 2$, the Euclidean distance metric is generated, which dominates common database systems according to Bugatti et al. [9].

Beneath these distance metrics, there are many other metrics, such as Canberra [9] or Dynamical-Partial [16] distance function. In contrast to Minkowsky distance functions, Dynamical-Partial distance metric d_m uses only the m smallest distances for the computation of the distance of data items [16]. As a result, in some specific use cases, it can be a great benefit using the Dynamical-Partial distance metric, because the influence of particular dimensions can deteriorate the distance of data items.

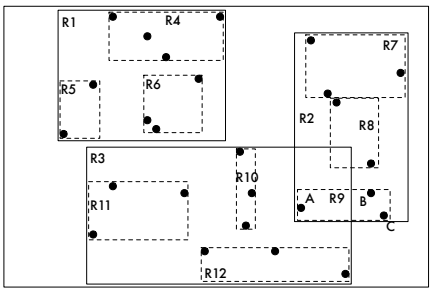


Figure 1: R-Tree with overlapping MBRs.

2.4 Index Structures

Since we aim at providing a comprehensive set of indexes, we want to consider different types of index structures. Thus, we use the classification of Weber et al. [23] to address a broad variety of different approaches. Thus, index structures are classified by partitioning of the data space. Index structures that partition the whole space are called space partitioning methods, whereas data partitioning methods partition the necessary space according to the location of data points [8, 23]. Consequently, there are regions that are not taken into account by performing a query on data partitioning methods.

Alternatively, Andoni and Indyk [2] classify index structures by query results. There are exact index structures that guarantee to retrieve the exact result of a query. Although, this behavior is usually preferred, there are approximation-based index structures, guaranteeing to retrieve points that are similar to the correct result of a query. For instance for k-NN queries, approximation-based index structures provide k near neighbors to the query point instead of all exact nearest neighbors. Hereby, the quality of the retrieved results, called precision, can differ significantly, because approximate index structures aim at improving the query performance by decreasing the precision. Nevertheless, an approximation-based index should hold a threshold, because resulting data would not be useful. In the following sections, we present some representatives of index structures. First, exact index structures, such as R-Tree [11], Pyramid Technique [5], and VA-File [22] are introduced. Subsequently, p-stable Locality Sensitive Hashing [12] as an approximation-based index structure is presented.

2.4.1 Exact Index Structures

Giving an overview of existing index structures, we introduce promising exact index structures in this section. Furthermore, the difference between space partitioning and data partitioning methods is stated by presenting at least one index structure for each category.

R-Tree.

One of the most important multi-dimensional index structures is the R-Tree [11], introduced by Guttmann in 1984. Since this time, many new index structures are proposed based on the ideas used in the R-Tree. For instance R^+-Tree [21], R^*-Tree [4], X-Tree [6], A-Tree [19], and SR-Tree [13]. Beside these structures, there are many more index structures which are not mentioned here. For further informations, see Samet [20], giving a comprehensive overview of existing index structures.

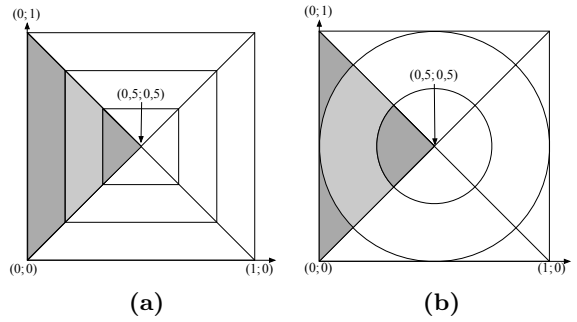


Figure 2: Space partition of a 2-d space by Berchtold et al. [5] (a) and Lee and Kim [15] (b).

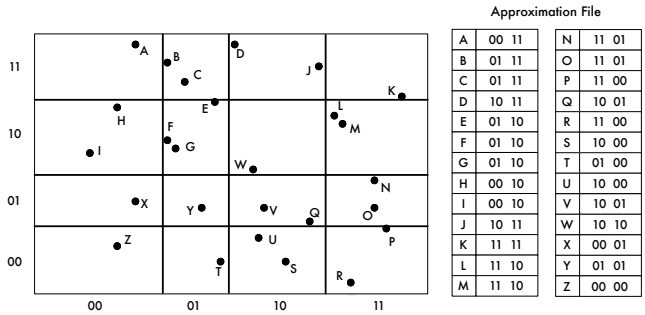


Figure 3: Partitioning of the VA-File.

However, the basic idea of these index structures is to administrate points hierarchically in a tree. The R-Tree partitions the data space using minimum bounding rectangles (MBR). A minimum bounding rectangle can be described by two points, being the end of the diagonal of the rectangle. Stepwise, the space is partitioned by MBRs, so that the superordinate MBR encloses all of its subordinate MBRs, as we visualize in Figure 1.

With increasing dimensionality, R-Trees face the challenge of overlapping MBRs. A query rectangle, situated in a region, where two or more MBRs overlap (like the MBR R2 and R3 in Figure 1), forces the R-Tree to follow up two or more different routes in the tree. Thus, the query performance decreases [6]. To overcome this disadvantage other index structures that we mentioned before, are developed.

Pyramid Technique.

An example for an exact space partitioning index structure is the Pyramid Technique, which was introduced by Berchtold et al. [5]. The Pyramid Technique divides an n-dimensional space into $2d$ pyramids [5]. A d dimensional normalized point x is inserted into a pyramid according to the dimension j_{max} with its maximum distance to the center of the data space. Thus, the pyramid number p_i is computed as follows:

$$i = \begin{cases} j_{max} & \text{if } x_{j_{max}} < 0,5 \\ (j_{max} + d) & \text{if } x_{j_{max}} \geq 0,5 \end{cases}$$

Second, for managing the space enclosed by a pyramid, the pyramids are divided in pyramid slices. According to the query types supported by the index structure, the partition of pyramids can be done in different ways. In Figure 2, we present two different possible methods for partitioning a pyramid, for a two dimensional normalized space.

In particular, the partition of Figure 2 (a) is proposed by Berchtold et al. [5] to support range queries. The other partition, shown in Figure 2 (b), is used by the approach of Lee and Kim [15] to support k-NN queries. It is possible to use the partitioning from Berchtold et al. for k-NN queries as well, but not in an efficient way. Anyway, a point is inserted into the slice depending on its distance to the center of the space. To sum up, for supporting different query types in an efficient way, different pyramid partitions are required.

VA-File.

In 1997, Weber and Blott [22] introduce the VA-File to overcome the curse of dimensionality. The VA-File is an improved sequential scan, because Weber et al. noticed a degeneration of most index structures to a sequential scan, if the dimensionality of data points exceeds a certain limit [23]. Hence, the authors propose to accelerate the sequential scan by using vector approximation.

The VA-File divides each dimension of the space into 2^b equally filled cells, where b is an user defined amount of bits per dimension. Each cell is labeled with a unique bit string, being the concatenation of the corresponding bit strings for every dimension. For every point, the bit string of the cell is stored, which the point is inserted into. Thus, the VA-File uses two lists: an approximation file that stores the bit string of the cells for every point and a vector file with the vector data for each point. An exemplary space partitioning and the corresponding approximation file can be seen in Figure 3.

Generally, the query algorithm of the VA-File traverses the whole approximation file to collect suitable candidates for the query result at first. After that, exact comparisons between the vector data of the candidates and the query are performed.

The approximation technique of the VA-File helps to reduce hard-disk accesses, because small bit strings can be kept in main memory. Even if the whole approximation file does not fit into the main memory, the sequential examination of the approximations reduces disk access costs compared to random accesses to many data items [22]. Another advantage is, in contrast to the Pyramid Technique, the availability of different algorithms to efficiently support all query types being executable on a sequential scan without adaption of the space partitioning of the VA-File.

2.4.2 *Approximation-based Index Structures*

Typical representatives for an approximation-based index structure are based on hash schemes. Apart from common hashing algorithms, scattering inserted data points over the amount of buckets is not applicable for similarity queries. Consequently, there is a need for hash functions, causing collisions when hashing locally near situated points. This challenge is handled by Locality Sensitive Hashing (LSH). The aim of LSH is to map the key to a one dimensional hash value. Thus, all comparisons are made on the hash value instead of a high dimensional key. Supporting nearest neighbor queries, LSH uses $(P1, P2, r, cr)$-sensitive functions h to compute the hash value. These functions h have to fulfill the following constraints [12]:

For every dataset in a d-dimensional space $p, q \in \mathcal{R}^d$:

1. $if\ ||p-q|| \leq r,\ then\ Pr[h(p) = h(q)] > P1$
2. $if\ ||p-q|| \geq cr,\ then\ Pr[h(p) = h(q)] < P2$

The first constraint demands that the probability for two points to be hashed into the same bucket has to be larger than $P1$ if their distance is smaller than r. Whereas, if their distance is bigger than cr, the probability should be smaller than $P2$. In order to be an useful locality sensitive function, $P1$ should be much bigger than $P2$.

Improving the precision of the index structure, usually several hash tables with different hash functions are used. Consequently, the need for $(P1, P2, r, cr)$-sensitive functions is obvious. A promising family of hash functions is used in p-stable LSH.

p-stable LSH.

The approach of p-stable LSH is based on p-stable distributions. A distribution $\mathcal{D}$ is p-stable for $p \geq 0$ if for any n the real numbers $v_1, ..., v_n$ and i.i.d. random variables $X_1, ..., X_n$ with distribution $\mathcal{D}$, the following constraint is fulfilled:

$$\sum_{i=1}^{n}(v_i X_i) \sim \Big(\sum_{i=1}^{n}(\|v_i\|^p)\Big)^{1/p} X,$$

$\sim$ means the operands have the same distribution and X is a random variable from the distribution $\mathcal{D}$ [18].

Using d random variables from $\mathcal{D}$ to form a d-dimensional Vector $\vec{a}$, the scalar of vector $\vec{a}$ and the data point $\vec{v}$ result in a random variable with distribution $\Big(\sum_{i=1}^{d}(\|v_i\|^p)\Big)^{1/p} X$ [12]. Several of these scalar products with different vectors can be used to estimate $\|\vec{v}\|_p$ (the L_p distance metric). The corresponding distributions are:

- The Cauchy Distribution $\mathcal{D}_C(0,1)$, defined by the density function $c(x) = \frac{1}{(\pi(1+x^2))}$ is 1-stable and can be used to estimate the Manhattan distance metric.
- The Gaussian (Normal) Distribution $\mathcal{D}_G(0,1)$, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is 2-stable and can be used to estimate the Euclidean distance metric.

Instead of estimating a distance metric, the scalar product with vectors from p-stable distributions can be used to compute hash values of the data points, because the scalar product maps the vectors to a one dimensional space. Furthermore, the result of the scalar product has the same distribution as the L_p distance metric, which guarantees the $(P1, P2, r, cr)$-sensitiveness.

3. ADVANCED CHALLENGES

After giving a short introduction to general challenges of indexing multi-dimensional data, in this section we provide existing challenges of evaluating the performance of index structures for a specific use case. For giving an overview of possible challenges when evaluating index structures, we group the challenges into three groups.

3.1 Parameter of the Index Structures

Some index structures have specific parameters for tuning their performance. Thus, when evaluating the performance of index structures, these parameter have to be considered as well. For index structures given in Section 2.4, these parameter are: the minimum and the maximum number of points within a MBR for the R-Tree, the number of slices a pyramid is divided in, for the Pyramid Technique, and the length of the bit vector for the VA-File. The parameters of the approximation-based index structure p-stable LSH presented in Section 2.4 are number of hash functions and width of hash buckets.

Thus, we have to assume, that these parameters have an impact on the performance of index structures. Therefore, it is necessary to analyze suitable parameter values when trying to identify an appropriate index structure for a given use case. However, there are some problems considering an appropriate value of some parameters. For example, the vectors used for p-stable LSH are randomly chosen from p-stable distributions. As a result of this random component it is possible that the performance and precision results of the same index structure created with different seeds of the random component can differ very much, within the same use case. This is problematic when trying to quantitatively evaluate the index structure.

3.2 Workload and used Queries

Although, two different applications can deal with the same data, they can have a different workload. For that reason, they can differ in requirements of index structures. The workload of an application depends on the used query types. Yet, it is obvious, not to use the Pyramid Technique presented from Berchtold et al. [5] for performing a k-NN query, but the version presented by Lee and Kim [15], because it is optimized for this query type.

For defining the workload of a database system we use a definition inspired by Ahmad et al. [1]. As a result, the workload is defined by the percentage of the query types used and the amount of concurrent requests performed.

3.3 DBMS Environment

In addition to use cases and workload, the test environment has an impact on the performance of each index structure. As already mentioned, the VA-File is optimized for database systems, storing data items on a disk and not in main memory. Evaluations of VA-File and sequential scan, result in different conclusions according to an evaluation with an in-memory database or a database storing items on the disk. Consequently, it is necessary to consider the underlying storage management of the database system as well.

Beside the storage management of a database, the amount of main memory and the CPU performance are other impact factors to the performance.

4. TOWARDS A FRAMEWORK

Since we aim at providing a comprehensive library of use cases and suitable indexes, we motivate a framework to give users the possibility to evaluate own use cases with different index structures. In this paper, we summarize key aspects of a framework that supports four groups. In Figure 4, we give an overview of these four groups.

4.1 Extensibility

First, the framework has to be extensible w.r.t. four key aspect, we present in Section 2. In other words, for an user, it has to be possible to implement, integrate, and evaluate own index structures. Furthermore, it has to be possible to extend the framework and existing index structures with additional distance metrics and also other query types. Finally, it has to be possible to integrate existing data in the framework and to create data with specific properties like a

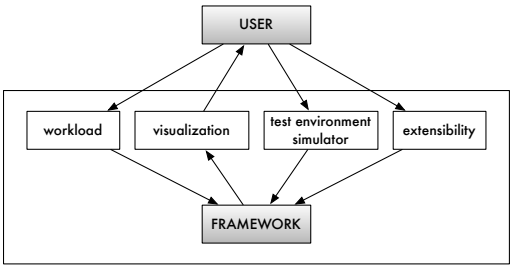


Figure 4: Structure of the Framework.

specific data distribution. Thus, creating data distributions is not trivial, an interface has to be created for importing existing data sets and communicating with systems like R, see for instance [14].

4.2 Adaptability to Different Workload

In real world applications, the workload differs quite much. On the one hand, there are use cases that use only read transactions. On the other hand, the workload can consist of read and write transactions. Thus, the performance results of workloads can differ very much. Hence, an interface is needed for importing workloads from existing systems. A further requirement, is to support standardized benchmarks, e.g. the TPC-H Benchmark[2].

Beside the queries used, the desired precision of the query results have to be defined by the user. Thus, if approximate results are allowed, the user has to define the accuracy of the results. Nevertheless, the precision depends on the data properties, the given distance metric, the used queries and the parameters of the index structure as well.

4.3 Test Environment Simulator

Existing index structures are created with respect to different optimization criteria. As already mentioned, the VA-File is optimized for reducing disk accesses. Consequently, another criteria our framework has to consider is the environment the tests are located in. Thus, within the framework a parameter has to exist, for setting whether the test is for an in-memory database or if a disk access is needed for accessing the data. Due to an assumption, that the access time of data differs very much considering Hard-Disk-Drives and Solid-State-Disks, the framework should have a component for virtualizing the disk access. With this component it is possible to perform tests on one system while simulating an access delay of another system. In addition to this storage device simulator, a simulator for all hardware components is required to give an useful hint about the best performing index structure.

Additionally, within the framework a parameter has to exist for defining the values of some index structure parameters such as the maximum number of data items of leaves of the R-Tree.

4.4 Visualization

In case the index structure has to struggle with a specific data distribution or query type, it can be useful to visualize the space partitioning of the index structure. With this visualization, further hypothesis can be drawn on the benefits or pit-falls of the chosen index structure. For instance, the user is able to follow the split of MBRs in the R-Tree and can easily identify overlapping regions while the tree is being constructed. Another aspect, being worth to visualize, is a statistic on query performance. These statistics help to analyze the performance of different index structures for a given workload or an index structure under different workloads. Apart form the query performance, other interesting values may be worth visualizing. The time spent on constructing the index structure is important for systems with many delete and update queries, because a reconstruction of the index is sometimes necessary when a certain threshold of changed data is reached. Furthermore, when using an approximate index structure, the precision of executed queries and the overall precision of the index structure is worth visualizing, because it has an impact on the suitability of an index structure for a special use case.

4.5 Working with the Framework

Finally, our framework shall help finding the most suitable index structures for a given use case. For this, the expected workload has to be known. These parameters include supported query types, exact or approximate results, data dimensionality and distribution, amount of data, the delay of the data access, and the environment. By finding suitable index structures for the given parameters, there are index structures that do not have to be taken into account, because they do not support certain query types or work approximately although the result is restricted to be exact. After excluding unsuitable index structures, the remaining index structures are evaluated under the given workload. By reviewing the performance results, the user can choose the suitable index structure for her use case.

5. RELATED WORK

In the last decades, many new index structures are created [5, 10, 22]. In addition, existing index structures are improved for supporting new query types [15] or to increase performance [6]. However, within the presented evaluation of these index structures only a small set of existing index structures is considered. For example, within Berchtold et al. [5], the Pyramid Technique is evaluated against X-Tree, Hilbert R-Tree, and sequential scan. Therefore, it is problematic to identify, which is the most appropriate index structure for a given problem. Additionally, different performance evaluations are done in different environments with different data characteristics. So, it is problematic to generalize the results of an evaluation.

For giving a comparison of the performance of multi-dimensional index structures, there already exists some frameworks, like the GiST [3] framework or the MESSIF [3] framework. In contrast to the framework we present here, these frameworks have some additional constraints. For example, the GiST framework only focuses on trees, hence no other multi-dimensional index structures such as the VA-File or the Pyramid Technique are considered, while the MESSIF framework only focuses on metric data. Another framework limiting the available index structures is introduced by Muja et al. [17]. The aim of this framework is to optimize parameters of approximate index structures in order to match the required precision under given data distributions.

[2]http://www.tpc.org/tpch/

[3]http://gist.cs.berkeley.edu/

6. CONCLUSION

In this paper, we provide an overview of existing challenges in finding an appropriate index for multi-dimensional data for a specific use case. First, we explain distance metrics and common query types that have to be considered. Second, the parameters of the index structures can have an impact on the performance of an index structure. Third, for users, it has to be possible to define own workload pattern and the environment, the application is located in.

For supporting these characteristics of real-world use cases we present requirements of a framework we intend to develope. Our framework has to support four key aspects. Namely, it has to be extensible, support different workload patterns, virtualize different use case environments, and contain a visualization component for improving user experiences.

7. ACKNOWLEDGMENTS

The work in this paper has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Programme under Contract No. FKZ:13N10817 and FKZ:13N10818. Additionally we want to thank Sandro Schulze for giving us useful comments.

8. REFERENCES

[1] M. Ahmad, A. Aboulnaga, and S. Babu. Query interactions in database workloads. In *Proc. Int'l. Workshop on Testing Database Systems*, DBTest, pages 11:1–11:6. ACM, 2009.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[3] M. Batko, D. Novak, and P. Zezula. Messif: Metric similarity search implementation framework. In *Proc. Conf. on Digital Libraries (DELOS)*, 2007.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. Int'l. Conf. on Mgmt. of Data (SIGMOD)*, pages 322–331. ACM, 1990.

[5] S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid-Technique: Towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27:142–153, 1998.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high-dimensional data. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 28–39. Morgan Kaufmann Publishers Inc., 1996.

[7] C. Böhm. *Efficiently Indexing High-Dimensional Data Spaces*. PhD thesis, Ludwig-Maximilians-Universität München, 1998.

[8] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33:322–373, 2001.

[9] P. H. Bugatti, A. J. M. Traina, and C. Traina, Jr. Assessing the best integration between distance-function and image-feature to answer similarity queries. In *Proc. ACM Symp. on Applied Computing (SAC)*, pages 1225–1230. ACM, 2008.

[10] E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(9):1647–1658, 2008.

[11] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *SIGMOD'84, Proc. of Annual Meeting*, pages 47–57, 1984.

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. Symp. on Theory of Compu. (STOC)*. ACM, 1998.

[13] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. Int'l. Conf. on Mgmt. of Data (SIGMOD)*, pages 369–380. ACM, 1997.

[14] V. Köppen. *Improving the Quality of Indicator Systems by MoSi – Methodology and Evaluation*. PhD thesis, Freie Universität Berlin, 2008.

[15] D.-H. Lee and H.-J. Kim. An efficient technique for nearest-neighbor query processing on the SPY-TEC. *Trans. on Knowl. and Data Eng. (TKDE)*, 15:1472–1486, 2003.

[16] B. Li, E. Chang, and Y. Wu. Discovery of a perceptual distance function for measuring image similarity. *Multimedia Systems*, 8(6):512–522, Apr. 2003.

[17] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. Int'l. Conf. on Computer Vision Theory and Applications (VISAPP)*, pages 331–340, 2009.

[18] J. P. Nolan. *Stable distributions: Models for heavy tailed data*. Springer-Verlag, 2009.

[19] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 516–526. Morgan Kaufmann Publishers Inc., 2000.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[21] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 507–518. Morgan Kaufmann Publishers Inc., 1987.

[22] R. Weber and S. Blott. An approximation-based data structure for similarity search. Technical Report ESPRIT project, no. 9141, 1997.

[23] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 194–205. Morgan Kaufmann Publishers Inc., 1998.

[24] R. Zhang, B. C. Ooi, and K.-L. Tan. Making the pyramid technique robust to query types and workloads. In *Proc. Int'l. Conf. on Data Engineering (ICDE)*, pages 313–324. IEEE Computer Society, 2004.

Minimal-Invasive Indexintegration

Transparente Datenbankbeschleunigung

Alexander Adam
dimensio informatics GmbH
Brückenstraße 4
09111 Chemnitz
alad@dimensio-informatics.de

Sebastian Leuoth
dimensio informatics GmbH
Brückenstraße 4
09111 Chemnitz
lese@dimensio-informatics.de

Wolfgang Benn
Technische Universität Chemnitz
Straße der Nationen 62
09107 Chemnitz
benn@cs.tu-chemnitz.de

Keywords

Datenbankenerweiterung, Indizierung, Transparent

ZUSAMMENFASSUNG

Aktuelle Datenbanksysteme bieten dem Nutzer eine enorme Funktionsvielfalt [12, 8]. Selbst sehr spezielle Gebiete wie z. B. Geodatentypen [9, 14] werden unterstützt. Abhängig vom verwendeten Datenbanksystem können diese Fähigkeiten durch einen Nutzer noch erweitert werden. Beispiele hierfür wären Funktionen und nutzerdefinierte Datentypen [12, 8]. Alle diese Erweiterungen sollten natürlich nicht die Geschwindigkeit des Datenbanksystems negativ beeinflussen. So gibt es neben normalen Indexen auch Funktionsindexe und Indexe für nutzerdefinierte Datenhaltungen [20, 2]. Die Möglichkeiten, zu indizieren sind dabei, je nach Datenbankhersteller, vorbestimmt und selbst nicht erweiterbar. Allen diesen Techniken ist weiterhin gemein, dass sie direkt am Datenbanksystem ansetzen und teils auch in der Anfragesprache sichtbar sind. Es ist daher nicht einfach möglich, eine Anwendung, die fest vorgegeben ist, mittels solcher Techniken zu beschleunigen. In diesem Papier wollen wir eine Möglichkeit vorstellen, mit der eine solche Beschleunigung auch dann noch möglich ist, wenn weder das Datenbanksystem noch die Anwendung im Zugriff des Nutzers stehen. Verdeutlicht wird dieses am Beispiel der JDBC-Schnittstelle [15].

1. EINLEITUNG

Datenbanksysteme haben in den letzten Jahrzehnten ein breites Anwendungsspektrum erschlossen. Funktionalitäten wurden oft aufgrund der Bedürfnisse der Anwender hinzugefügt [17]. So gibt es heute nutzerdefinierte Datentypen, nutzerdefinierte Datenablagen und selbst elementare Dinge wie Indexe, die durch einen Nutzer in ihrer Struktur bestimmt werden.

Allen gemein ist eine gewisse Abhängigkeit der Implementation vom Datenbankhersteller und einer damit verbundenen schlechten Portabilität der entwickelten Module. Besonders im Bereich der Indizierung um den es in diesem Papier gehen soll, stellen sich aber noch weitere Probleme dar.

Normalerweise kann ein Index auf einer beliebigen Tabelle und deren Spalten definiert werden. Die Anwendung bemerkt nur insoweit etwas von dieser Aktion, als dass Anfragen schneller beantwortet werden sollten.

Ein kleines Beispiel soll dies verdeutlichen. Die folgende Anfrage gibt alle Mitarbeiterinformationen zurück, die zu Mitarbeitern mit mehr als 1000 Euro Gehalt gehören:

Listing 1: SQL-Anfrage mit Bedingung im `WHERE`-Teil

```
SELECT  *
FROM    mitarb
WHERE   mitarb.gehalt > 1000
```

Für die Anwendung, die diese Anfrage an die Datenbank stellt, ist es vollkommen unerheblich, ob sich auf der Gehaltsspalte der Mitarbeitertabelle ein Index befindet oder nicht. Die Anfrage würde, ob nun mit oder ohne Index, immer gleich aussehen.

Wir nehmen nun an, dass die Gehaltsspalte durch einen anderen Typ ersetzt werden soll, Gründe hierfür könnten eine effizientere Speicherung oder bessere Zugriffsmöglichkeiten sein. Wir nehmen weiter an, dass das Datenbanksystem für den neuen Typ den >-Operator nicht mehr anbietet. Es muss nun eine Vergleichsfunktion geschrieben werden, die die Aufgabe des >-Operators übernimmt. Diese tiefgreifende Umstrukturierung scheint nun bis in die Anfrage durch, ist also nicht mehr transparent:

Listing 2: SQL-Anfrage mit Anfrage an eine nutzerdefinierte Funktion im `WHERE`-Teil

```
SELECT  *
FROM    mitarb
WHERE   pruefe( mitarb.gehalt, 1000) = 1
```

Eine Anwendung, die der Nutzer nicht verändern kann, würde also von diesem neuen Typ ausschließlich dann profitieren, wenn der Hersteller diese Möglichkeit vorsieht oder standardmäßig einbaut. Für bestehende Umgebungen ist all dies also keine Option.

Im obigen Fall würde noch die Möglichkeit des automatischen SQL-Umschreibens einen Ausweg bieten [12, 8]. Dabei wird eine materialisierte Sicht angelegt. Anschließend wird der Datenbank mitgeteilt, dass bestimmte Anfragen nicht so gestellt werden sollen, wie sie der Anwender abgesetzt hatte,

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

sondern in veränderter Form auf der materialisierten Sicht abgearbeitet werden. Das Vorgehen muss vom Datenbanksystem aber auch unterstützt werden. Je nach Ausprägung müssen außerdem genaue Übereinstimmungsmerkmale angegeben werden, mit denen das Umschreiben ausgelöst wird. Scheitert das Umschreiben, weil es eine kleine Varianz in der Anfrage gibt, wird das Ergebnis u. U. falsch.

Im Folgenden werden wir eine Möglichkeit aufzeigen, wie es ohne einen Eingriff – weder bei der Anwendung noch bei der Datenbank – möglich ist, einen Index weitestgehend transparent in ein bestehendes System zu integrieren. Dazu wird zunächst untersucht, an welchen Stellen und wie in die Kommunikation von Anwendung und Datenbanksystem eingegriffen werden kann. Anschließend wird auf die Herausforderungen eingegangen, die sich bei dem hier genutzten Ansatz zeigen und wie diese gelöst werden können.

2. INTEGRATIONSPUNKTE

2.1 Überblick

Um mögliche Integrationspunkte zu finden, muss zunächst untersucht werden, wie eine Anwendung mit einem Datenbanksystem kommuniziert. Üblicherweise werden hierfür Datenbanktreiber eingesetzt. Das sind Programmbibliotheken, die die Anfragen in ein dem Datenbanksystem verständliches Protokoll überführen und dieses dann übermitteln. Der Datenbankserver dekodiert das Protokoll und arbeitet die darin enthaltenen Anweisungen ab.

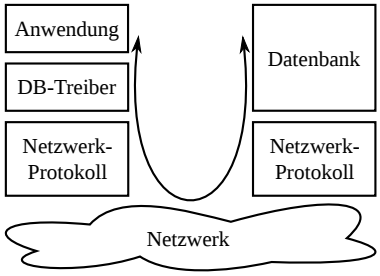


Abbildung 1: Schematische Darstellung des Zusammenspiels zwischen Anwendung und Datenbank. Die Anwendung verwendet ein API für einen Datenbanktreiber, welcher vom Datenbankhersteller zur Verfügung gestellt wird. Dieser Treiber ist dann für die Kommunikation mit dem Datenbanksystem über ein Netzwerk verantwortlich.

Das beschriebene Szenario – abgebildet in Abbildung 1 – offenbart drei Möglichkeiten, eine Integration vorzunehmen:

- den Datenbanktreiber,
- die Kommunikation über das Netzwerk und
- das Datenbanksystem selbst.

Im Folgenden werden wir uns auf den Datenbanktreiber, also die Anwendungsseite, konzentrieren. Die Vorgehensweise bei der Integration in die Kommuniktion ist bereits in [1] beschrieben. Möglichkeiten, ein Datenbanksystem zu erweitern, wurden in den letzten Jahrzehnten vielfach an anderer Stelle beschrieben [4, 20, 2].

2.2 Datenbanktreiber

Der Datenbanktreiber ist die Schnittstelle für einen Anwendungsprogrammierer, um mit dem Datenbanksystem zu kommunizieren. Er stellt Funktionen bereit, um Verbindungen zu verwalten und Datenbankanfragen abzuarbeiten (sei es nun SQL oder irgendeine andere Art der Anfrage) [5, 11]. Es ist wichtig, zu beobachten, dass dabei jeder Abarbeitungsschritt von der Anwendung ausgelöst wird. Alle Ergebnisse einer Anfrage werden nicht einfach als Resultat einer `query()`-Funktion zurückgegeben, sondern müssen aktiv angefordert werden. Eine typische Schrittfolge, die eine Anwendung verwenden könnte, ist in Listing 3 aufgezeigt.

Listing 3: Mögliche, stark reduzierte, Schrittfolge bei der Nutzung einer Datenbankbibliothek, hier JDBC. Es wird zunächst eine Verbindung geöffnet, dann ein Statement mit ungebundenen Variablen vorbereitet und gebunden. Schließlich werden die angefragten Daten abgeholt.

```
Connection conn = DriverManager.getConnection(
                  "jdbc:mysql://localhost/testdb",
                  "username",
                  "password");
Statement ps = conn.prepareStatement(
                  "SELECT * FROM mitarb
                   WHERE gehalt > ?");
ps.setInt(1, 1000);
ResultSet rs = ps.executeQuery();
String name = rs.getString("name");
ps.close();
conn.close();
```

Wie nun stellt sich hier eine Möglichkeit zur Integration dar? Es ist möglich, vor jede Funktion, die eine Anwendung vom originalen Datenbanktreiber aufruft, eine eigene Funktion zu setzen. Die Anwendung ruft nun die eigene Funktion, ohne dies zu bemerken, und die eigene Funktion ruft schließlich die originale. Da nun alle Daten, die von einer Anwendung zum Datenbanksystem gesendet werden, vorher analysiert und verändert werden können, stellt sich so dieser Integrationspunkt dar. Außerdem können eigene Funktionen auf der so abgefangenen Datenbankverbindung „huckepack“ aufgesetzt werden.

Ein erster Gedanke, dies zu realisieren, könnte in die Richtung einer *DLL-Injection* [18] gehen. Das bedeutet das komplette Ersetzen des vom Datenbankhersteller bereitgestellten Treibers durch einen eigenen. Dieser kann, da die Protokolle nicht zwingend offengelegt sein müssen, die Kommunikation mit dem Datenbanksystem nicht selbst übernehmen, sondern ruft den ursprünglichen Datenbanktreiber. Abhängig von der Anzahl der zu implementierenden Funktionen kann dies ein möglicher Weg sein. Eine weit verbreitete solche Schnittstelle, um aus einer Javaanwendung mit einem Datenbanksystem in Verbindung zu treten, ist JDBC. Mit seinen vielen Klassen und mehr als 1000 zugehörigen Methoden [6] wäre es eine sehr langwierige Aufgabe, einen solchen sogenannten Wrapper [7] zu implementieren. In einem unserer Produkte namens *Cardigo* ist dieses aber bereits implementiert und erleichtert so die Aufgabe enorm. Weitere Projekte und Arbeiten zu diesem Thema sind unter anderem auch bei [19, 22, 21] und [13] zu finden.

Cardigo ist ein Rahmenwerk, welches u. a. alle Klassen und Methoden enthält, die das JDBC-API anbietet. Anstatt selbst ein kompletter Treiber zu sein, bietet es lediglich einen Wrapper für einen „richtigen“ Datenbanktreiber. Das Ziel dieses Produktes ist es, einem Anwender die Möglichkeit zu geben, die Datenbankkommunikation – in diesem Falle hauptsächlich Anfragen, Ergebnisse u. s. w. – zu loggen oder gar zu verändern. Mit diesem Werkzeug können wir nun die Integration des Index angehen.

Technisch ist zur Integration von Cardigo nichts weiter nötig, als ein geänderter Verbindungsparameter für die Anwendung. Dieser bewirkt, dass anstelle des originalen JDBC-Treibers nun Cardigo geladen wird. Der Aufbau dieser veränderten Umgebung ist in Abbildung 2 verdeutlicht.

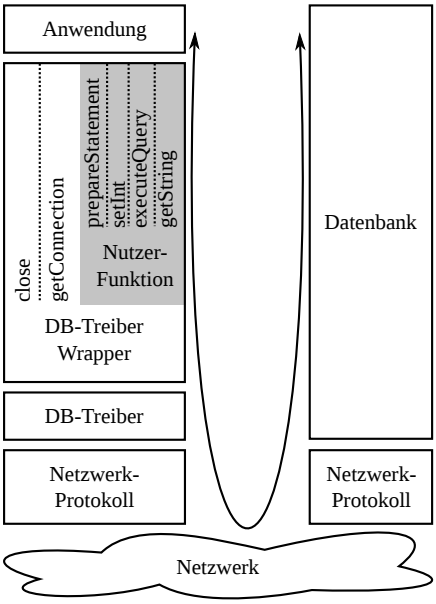


Abbildung 2: Anwendung, die Cardigo durch einen veränderten Kommunikationsparameter nutzt. Der Code, den ein Nutzer schreibt, umfasst typischerweise nicht den gesamten API-Umfang von JDBC, weshalb er hier nicht über die gesamte Breite dargestellt ist. Exemplarisch sind die in Listing 3 verwendeten Methoden dargestellt.

3. INTEGRATION

Bevor wir die Integration des Index weiter betrachten, wollen wir kurz darauf eingehen, wie ein in ein Datenbanksystem integrierter Index arbeitet: Wird eine Anfrage an die Datenbank gestellt und treffen einige der verwendeten Attribute die im Index enthaltenen, so wird der Index verwendet. Dabei wird die Anfrage vom Index bearbeitet und im Ergebnis entsteht einen Liste von Ergebniskandidaten. Diese können in Form von RowIDs [3] oder auch einfach als Primärschlüssel vorliegen. Das Datenbanksystem überprüft dann nur noch die Datensätze, deren Identifikatoren der Index als Ergebnis lieferte. Auf diese Art wird der Aufwand, den das Datenbanksystem beim Laden der Datensätze und ihrer Verifikation hat, erheblich verringert [10].

Im Folgenden wird zunächst die *logische Integration* betrachtet, d. h., wie es überhaupt möglich ist, Ergebnisse eines Index an das Datenbanksystem zu übertragen. Es schließt sich eine Betrachtung an, die die *programmtechnische Integration* betrachtet, da sich hier noch einige weitere Probleme auftun.

3.1 Logische Integration

Die Grundidee der Integration ist es nun, die Anfrage, die eine Anwendung absetzt, so zu verändern, dass sie das Verhalten eines datenbankinternen Index nachahmt. Die Anfrage aus Listing 1 könnte nun, wie in Listing 4 aufgezeigt, um das Primärschlüsselattribut erweitert werden.

Listing 4: SQL-Anfrage, die um die Ergebnisse eines externen Index erweitert wurde

```
SELECT    *
FROM      mitarb
WHERE     gehalt > 2000
      AND id IN (4, 18)
```

Dieses Vorgehen vertraut darauf, dass der Optimierer des Datenbanksystems erkennt, dass es sich bei den Werten in der `IN`-Klausel um Primärschlüssel handelt. Er muss diese möglichst am Anfang der Anfrageverarbeitung einbeziehen, um die Menge der zu untersuchenden Tupel zu minimieren.

Durch Tests haben wir herausgefunden, dass die Anzahl der in `IN`-Listen enthaltenen Elemente eine Obergrenze hat. Durch die Disjunktion mehrerer `IN`-Listen kann diese zu einem gewissen Grad ausgeglichen werden. Somit ist es möglich, sehr lange Anfragen zu erzeugen, Allerdings gibt es auch eine Grenze für die maximale, vom Datenbanksystem zugelassene, Anfragelänge. Bei IBM DB2 und Oracle ist diese Maximallänge bspw. 64kB [8, 16].

Ein weiterer Aspekt, der bei diesen langen Anfragen betrachtet werden muss, ist, dass diese vom Datenbanksystem auch geparst werden. Werden die Anfragen lang, so steigen auch deren Parsezeiten, selbst optimistisch betrachtet ist dieser Zusammenhang linear. Auch systematisch bedingt, ist, dass der Optimierer zu lange und viele `IN`-Listen nicht beachtet, selbst wenn es sich um Primärschlüssel handelt. Mit Hinweisen an den Optimierer kann hier gegengesteuert werden. Zusammenfassend lässt sich aber sagen, dass ab einer gewissen Anfragelänge, der Gewinn durch den Index sich im schlechtesten Falle sogar ins Gegenteil verkehren kann.

Um diese Beschränkungen zu umgehen, können die Ergebnisse des Index auch in eine Tabelle in der Datenbank eingefügt werden. Hier gibt es wieder mehrere Möglichkeiten:

1. Einfügen in eine Tabelle, die angelegt ist
2. Einfügen in eine temporäre Tabelle

In beiden Fällen muss allerdings die Datenbank in der Weise modifiziert werden, dass die Anwendung, in die der Index integriert wurde, das Recht hat, auf diese Tabellen zu schreiben. Die Tabellen müssen prinzipiell eine Spalte für die Anfrage und eine Spalte für die Ergebnisse des Index besitzen.

Werden über eine Sitzung Anfragen nur sequenziell bearbeitet, haben temporäre Tabellen den Vorteil, dass zwischen verschiedenen Datenbanksitzungen nicht mittels der eben beschriebenen zusätzlichen AnfrageID-Spalte in dieser

Tabelle unterschieden werden muss. Das ist darin begründet, dass temporäre Tabellen für jede Sitzung als leere neue Tabellen erscheinen. Die Aktionen einer Sitzung wirken sich nicht auf den Inhalt der temporären Tabelle in einer anderen Sitzung aus.

Ein bisher nicht zur Sprache gekommener Punkt ist das Aktualisieren des Index. Natürlich muss ein datenbankexterner Index über `INSERT`-, `UPDATE`- und `DELETE`-Operationen informiert werden. Der einfachste Weg ist, wenn alle Operationen, die auf der Datenbank laufen, über die gleiche abgefangene Schnittstelle gehen und so direkt gelesen werden können. In der Realität ist dies jedoch unpraktisch, da diese Voraussetzung nicht zu 100% gewährleistet werden kann. Trigger sind eine Variante, wie Veränderungen in der Datenbank nach außen gereicht werden können, diese müssen jedoch integriert werden dürfen. Hier ergeben sich damit auch Grenzen, über die hinaus unser Ansatz nicht angewendet werden kann. Eine andere Möglichkeit sind statische Datenhaltungen, die nur in definierten Zeitabschnitten aktualisiert werden, bspw. einmal pro Monat. Hier kann ein statischer Index genutzt werden, der über eine simple Zeitsteuerung aktualisiert wird.

3.2 Programmtechnische Integration

Oft halten sich Anwendungsprogrammierer an die Beispiele der Autoren der jeweiligen Datenbankschnittstelle. Jedoch erlauben alle APIs auch eine freiere Nutzung, d. h., dass die Schrittfolge der Kommandos nicht festgelegt ist und es verschiedene Gründe geben kann, die so aufgezeigten Standardroutinen zu verwerfen. Listing 3 zeigt zunächst einen Standardweg auf. Für diesen wollen wir nun die Schritte, die ein datenbankexterner Index verfolgen kann, aufzeigen:

- Statement vorbereiten (`conn.prepareStatement(...)`):
  if Anfrage relevant **then**
  Anfrage speichern
  end if
- Variablen binden (`ps.setInt(...)`):
  if Anfrage war relevant **then**
  Bindung speichern
  end if
- Statement ausführen (`ps.executeQuery()`):
  if Anfrage war relevant **then**
  Index anfragen
  Indexergebnisse in DB laden
  Anfrage modifizieren
  Datenbank anfragen
  end if
- Ergebnisse holen (`rs.getString(...)`):
  hier sind keine weiteren Schritte notwendig

Beobachtungen an realen Programmen zeigen, dass das Binden von Variablen teils mehrfach auf die gleichen Variablen angewendet wird. Mit dem eben beschriebenen Verfahren ist dies kein Problem. Auch Operationen, die ein Statement näher beschreiben, sind nach wie vor ausführbar, da alle Bestandteile erhalten bleiben und nur Ergänzungen vorgenommen werden.

4. ERGEBNISSE UND AUSBLICK

Das hier beschriebene System war bereits erfolgreich im Einsatz. Die Latenzen für das reine Abfangen der Datenbankaufrufe bewegen sich im einstelligen Mikrosekundenbereich, also dem, was für einen Funktionsaufruf erwartet werden kann. Hinzu kommt die Zeit für die Indexanfrage. Für einen realen Gewinn muss natürlich die Zeit für die Integration, die Indexanfrage und den Einbau der Ergebnisse in das Statement in Summe geringer sein, als die einer Anfrage ohne den externen Index.

Es zeigte sich auch, dass, wird eine Integration über Tabellen angestrebt, es verschiedene Arten gibt, die Ergebnisse aus der Ergebnistabelle zu entfernen. In 3.1 wurden die verschiedenen Arten von Tabellen hierfür beschrieben. Wenn keine Spalte für die AnfrageID verwendet werden muss, so können die Ergebnisse mit einem `TRUNCATE` entfernt werden. Dieses wird erheblich schneller ausgeführt, als ein `DELETE FROM ... WHERE anfrage_id == <current_id>`.

Unsere weitere Arbeit beschränkt sich nicht nur auf eine Integration in JDBC, die wir hier aufgezeigt haben, sondern geht auch darüberhinaus auf native Datenbanktreiber ein, die dann jedoch herstellerspezifisch sind. Hier müssen andere Mechanismen angewandt werden, eine Integration elegant zu vollziehen, die Prinzipien bleiben jedoch die gleichen. Auch der bereits vorgestellte Ansatz, einen Proxy zu integrieren, der das Protokoll, welches das Datenbanksystem im Netzwerk verwendet, versteht, wurde weitergeführt und zeigte sich bereits im Einsatz als wertvolle Hilfe. Das oben bereits erwähnte Cardigo dient uns dabei als Werkzeugkasten, der alle diese Möglichkeiten vereint.

5. LITERATUR

[1] A. Adam, S. Leuoth, and W. Benn. Nutzung von Proxys zur Ergänzung von Datenbankfunktionen. In W.-T. Balke and C. Lofi, editors, *Grundlagen von Datenbanken*, volume 581 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[2] E. Belden, T. Chorma, D. Das, Y. Hu, S. Kotsovolos, G. Lee, R. Leyderman, S. Mavris, V. Moore, M. Morsi, C. Murray, D. Raphaely, H. Slattery, S. Sundara, and A. Yoaz. *Oracle Database Data Cartridge Developers Guide, 11g Release 2 (11.2)*. Oracle, July 2009.

[3] P. Bruni, F. Bortoletto, R. Kalyanasundaram, G. McGeoch, R. Miller, C. Molaro, Y. Ohmori, and M. Parbs. *DB2 10 for z/OS Performance Topics*. IBM Form Number SG24-7942-00, IBM Redbooks, June 2011.

[4] Desloch et al. PatNr. US 6,338,056 B1 – Relational Database Extender that Supports User-Defined Index Types and User-Defined Search, Apr. 1999.

[5] R. Elmasri and S. B. Navathe. *Grundlagen von Datenbanksystemen (3. Aufl., Bachelorausgabe)*. Pearson Studium, 2009.

[6] M. Fisher, J. Ellis, and J. C. Bruce. *JDBC API Tutorial and Reference*. Pearson Education, 3 edition, 2003.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, January 1995.

[8] IBM. *SQL Reference, Volume 1*. IBM Corporation, Nov. 2009.

[9] IBM Deutschland GmbH. *DB2 Spatial Extender und Geodetic Data Management Feature – Benutzer- und Referenzhandbuch*, July 2006.

[10] T. Lahdenmäki and M. Leach. *Relational database index design and the optimizers: DB2, Oracle, SQL server, et al.* Wiley-Interscience, 2005.

[11] T. Langner and D. Reiberg. *J2EE und JBoss: Grundlagen und Profiwissen ; verteilte Enterprise-Applikationen auf Basis von J2EE, JBoss & Eclipse*. Hanser, 2006.

[12] D. Lorentz and M. B. Roeser. *Oracle Database SQL Language Reference, 11g Release 2 (11.2)*. Oracle, Oct. 2009.

[13] A. Martin and J. Goke. P6spy. `http://sourceforge.net/projects/p6spy/`.

[14] C. Murray. *Oracle Spatial Developers Guide, 11g Release 2 (11.2)*. Oracle, Dec. 2009.

[15] G. Reese. *Database Programming with JDBC and Java, Second Edition*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition, 2000.

[16] B. Rich. *Oracle Database Reference, 11g Release 2 (11.2)*, Sept. 2011.

[17] G. Saake, K.-U. Sattler, and A. Heuer. *Datenbanken: Konzepte und Sprachen, 3. Auflage*. mitp-Verlag, Redline GmbH, 2008.

[18] J. Shewmaker. Analyzing dll injection, 2006. GSM Presentation, Bluenotch.

[19] M. Smedberg. Boilerplate JDBC Wrapper. `http://blog.redfin.com/devblog/2011/03/boilerplate_jdbc_wrapper.html`.

[20] K. Stolze and T. Steinbach. DB2 Index Extensions by example and in detail, IBM Developer works DB2 library. Dec. 2003.

[21] Thedwick, LLC. jdbcgrabber. `http://code.google.com/p/jdbcgrabber/`.

[22] C. Wege. Steps out of Integration Hell - Protocol Interception Wrapper. In A. Rüping, J. Eckstein, and C. Schwanninger, editors, *EuroPLoP*, pages 455–458. UVK - Universitaetsverlag Konstanz, 2001.

Self-Tuning Distribution of DB-Operations on Hybrid CPU/GPU Platforms

Sebastian Breß
Otto-von-Guericke University Magdeburg
bress@iti.cs.uni-magdeburg.de

Siba Mohammad
Otto-von-Guericke University Magdeburg
siba.mohammad@st.ovgu.de

Eike Schallehn
Otto-von-Guericke University Magdeburg
eike@iti.cs.uni-magdeburg.de

ABSTRACT

A current research trend focuses on accelerating database operations with the help of GPUs (Graphics Processing Units). Since GPU algorithms are not necessarily faster than their CPU counterparts, it is important to use them only if they outperform their CPU counterparts. In this paper, we address this problem by constructing a decision model for a framework that is able to distribute database operations response time minimal on CPUs and GPUs. Furthermore, we discuss necessary quality measures for evaluating our model.

1. INTRODUCTION

In the context of database tuning, there are many different approaches for performance optimization. A new opportunity for optimization was introduced with General Purpose Computation on Graphics Processing Units (GPGPU) [12]. This approach allows to speed up applications that are suited for parallel processing with the help of GPUs. Parallel computing architectures like compute unified device architecture (CUDA) [12] make it possible to program a GPU almost as simple as a CPU. This technology opens a new branch in research that focuses on accelerating applications using GPUs.

CPUs are optimized for a low response time, meaning that they execute one task as fast as possible. The GPU is optimized for high throughput, meaning they execute as many tasks as possible in a fixed time. This is accomplished by massively parallel execution of programs by using multi threading. Furthermore, GPUs specialize on compute-intensive tasks, which is typical for graphics applications. Additionally, tasks with much control flow decrease performance on GPUs, but can be handled well by CPUs [12]. Consequently, database operations benefit differently by using the GPU. Aggregations are most suited for GPU usage, whereas selections should be outsourced with care. He et al. observed that selections are 2-3 times slower on the GPU compared with the CPU [6].

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

1.1 New Research Trend

A new research trend focuses on speeding up database operations by performing them on GPUs [4, 6, 13, 14].

He et al. present the concept and implementation of relational joins on GPUs [6, 7]. Pirk et al. develop an approach to accelerate indexed foreign key joins with GPUs [13]. The foreign keys are streamed over the PCIe bus while random lookups are performed on the GPU. Walkowiak et al. discuss the usability of GPUs for databases [14]. They show the applicability on the basis of an n-gram based text search engine. Bakkum et al. develop a concept and implementation of the SQLite command processor on the GPU [4]. The main target of their work is the acceleration of a subset of possible SQL queries. Govindaraju et al. present an approach to accelerate selections and aggregations with the help of GPUs [5]. From the examples, we can conclude that a GPU can be an effective coprocessor for the execution of database operations.

1.2 Problem Statement

We assume that an operation is executed through an algorithm which uses either CPU or GPU. For which operations and data is it efficient to execute database operations on a GPU? A GPU algorithm can be faster or slower as its CPU counterpart. Therefore, the GPU should be used in a meaningful way to achieve maximal performance. That means, only if it is to be expected that the GPU algorithm is faster it should be executed.

We do not know a priori which processing device (CPU or GPU) is faster for which datasets in a given hardware configuration. We have to take different requirements into account to determine the fastest processing device:

- the operation to be executed,
- the size of the dataset to process,
- the processing power of CPU and GPU (number of processor cores, clock rate), and
- the current load condition on CPU and GPU.

The following contributions are discussed in the following:

1. A response time minimal distribution of database operations on CPUs and GPUs can achieve shorter execution times for operations and speedup query executions. Hence, a basic idea how such a model can be constructed is depicted in this paper.
2. For the evaluation of our model, we define appropriate quality measures.

To the best of our knowledge, there is no self-tuning decision model that can distribute database operations response time minimal on CPUs and GPUs.

The remainder of the paper is structured as follows. In Section 2, we present our decision model. Then, we discuss model quality metrics needed for evaluation in Section 3. Section 4 provides background about related work. Finally, we present our conclusion and future work.

2. DECISION MODEL

In this section, we present our decision model. At first, we discuss the basic model. Then, we explain details of the estimation and the decision components.

2.1 Basic Model

We construct a model that is able to choose the response time minimal algorithm from a set of algorithms for processing a dataset D. We store observations of past algorithm executions and use statistical methods to interpolate future execution times. We choose the algorithm with the smallest estimated execution time for processing a dataset.

Definitions: Let O be a database operation and let $AP_O = \{A_1, .., A_m\}$ be an algorithm pool for operation O, i.e., a set of algorithms that is available for the execution of O. We assume that every algorithm has different performance characteristics. Hence, the execution times of different algorithms needed to process a dataset D are likely to vary. A data set D provides characteristic features of the input data. In this paper, we consider only the data size. Other characteristics, e.g., data distribution or data types will be considered in future work.

Let $T_A(D)$ be the execution time of an algorithm to process the dataset D. We do not consider hardware specific parameters like clock rate and number of processor cores to estimate execution times. Instead, we learn the execution behavior of every algorithm. For this purpose, we assign to every algorithm A a learning method L_A and a corresponding approximation function $F_A(D)$. Let $T_{est}(A, D)$=$F_A(D)$ be an estimated execution time of algorithm A for a dataset D. A measured execution time is referred to as $T_{real}(A, D)$. Let a measurement pair (MP) be a tuple (D,$T_{real}(A, D)$). Let MPL_A be a measurement pair list, containing all current measurement pairs of algorithm A.

Statistical Methods: We consider the following statistical methods for the approximation of the execution behavior of all algorithms. The first one is the *least squares method* and the second is *spline interpolation* with cubic splines [3]. We use these approaches because we observe in our experiments minimal overhead and a good accuracy (relative error $< 10\%$) of the estimated execution times. While other methods can learn the execution behavior depending on more than one feature, they often need a large amount of time for updating approximation functions and computing estimations, see Section 4. In the case of the least square method $F_A(D)$ is a polynomial of degree n. In the case of cubic splines, $F_A(D)$ is a spline.

Abstraction: The execution time of algorithms is highly dependent on specific parameters of the given processing hardware. In practice, it is problematic to manage all parameters, so maintenance would become even more costly. Hence, we do not consider hardware specific parameters and let the model learn the execution behavior of algorithms using statistical method L_A and the corresponding approximation function $F_A(D)$.

Model Components: The model is composed of three components. The first is the algorithm pool AP_O which contains all available algorithm of an operation O. The second is the estimation component which computes execution time estimations for every algorithm of an operation. The third part is the decision component which chooses the algorithm that fits best for the specified optimization criteria. Depending on whether the chosen algorithm uses the CPU or the GPU, the corresponding operation is executed on the CPU or the GPU. In this paper, we only consider response time optimization. However, other optimization criteria like throughput can be added in future work. Figure 1 summarizes the model structure.

2.2 Estimation Component

This section describes the functionality of the estimation component. First, we discuss when the approximation functions of the algorithms should be updated. Second, we examine how the model can adapt to load changes.

2.2.1 Updating the Approximation Functions

For an operation O that will be applied on a dataset D the estimation component computes for each available algorithm of O an estimated execution time. For this, we need an approximation function that can be used to compute estimations. Since we learn such functions with statistical methods, we need a number of observations for each algorithm to compute the corresponding approximation functions. After we computed an initial function for each algorithm, our model is used to make decisions. Hence, the model operation can be divided into two phases. The first is the initial training phase. The second is the operational phase.

Since the load condition can change over time, execution times of algorithms are likely to change as well. Hence, the model should provide a mechanism for an adoption to load changes. Therefore, the model continuously collects measurement pairs of all executed algorithms of an operation.

There are two problems to solve:

1. **Re-computation Problem:** find a strategy when to re-compute the approximation function for each algorithm, so that a good trade-off between accuracy and overhead is achieved.

2. **Cleanup Problem:** delete outdated measurements pairs from a measurement pair list because the consideration of measurement pairs from past load conditions is likely to be less beneficial for estimation accuracy. Furthermore, every measurement pair needs storage space and results in higher processing time for the statistical methods.

There are different heuristics that deal with the re-computation of approximation functions. Zhang et al. present possible approaches in [15]. These are:

1. Re-compute the approximation function always after a new measurement pair was added to the measurement pair list.

2. Re-compute the approximation function periodically.

3. Re-compute the approximation function, if the estimation error exceeds a certain threshold.

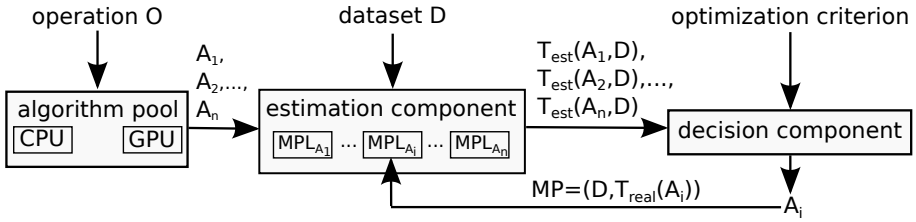


Figure 1: Information flow in the model

As Zhang et al. point out, the most aggressive method is unlikely to achieve good results because the expected overhead for re-computing the approximation function counteracts possible performance gains. Hence, we have to choose between approach 2 and 3. There is no guarantee that one approach causes less overhead than the other.

On one hand, periodic re-computation causes a predictable overhead, but it may re-compute approximation functions when it is not necessary, e.g., the relative estimation error is still beneath an error threshold. On the other hand, an event based approach re-computes the approximation functions only if it is necessary, but has the disadvantage, that the quality of estimations has to decrease until the approximation functions are re-computed. We choose the periodic approach, because of it's predictable overhead. We will consider the event based approach in future work.

Parameters: Now we discuss necessary parameters of the model. Let *RCR* be the re-computation rate of an algorithm. That means that after *RCR* measurement pairs were added to the measurement pair list of an algorithm A, the approximation function $F_A(D)$ of A is re-computed.[1] Hence, the used time measure is the number of added measurement pairs. Let *ITL* be the initial training length, which is the number of measurement pairs that have to be collected for each algorithm, before the model switches into its operational phase.

Now we address the *Cleanup Problem*. We have to limit the number of measurement pairs in the measurement pair list to keep space and computation requirements low. Furthermore, we want to delete old measurement pairs from the list that do not contribute to our current estimation problem sufficiently. These requirements are fulfilled by a ring buffer data structure. Let *RBS* be the ring buffer size in number of measurement pairs. If a new measurement pair is added to a full ring buffer, it overrides the oldest measurement pair. Hence, the usage of a ring buffer solves the *Cleanup Problem*. *RCR* and *RBS* are related because if *RCR* is greater than *RBS* there would be measurement pairs that are never considered for computing the approximation functions. Hence, *RCR* should be smaller than *RBS* .

Statistical Methods: The used statistical methods have to be computationally efficient when computing approximation functions and estimation values. Additionally, they should provide good estimations (relative estimation error <10%). Since the times needed to compute estimations sums up over time, we consider a method computationally efficient if it can compute one estimation value in less than 50μs. Our experiments with the ALGLIB [2] show that this is the case for the least squares and cubic splines.

[1]For each operation one measurement pair is added to the list of the selected algorithm.

Self Tuning Cycle: The model performs the following self tuning cycle during the operational phase:

1. Use approximation functions to compute execution time estimations for all algorithms in the algorithm pool of operation O for the dataset D.
2. Select the algorithm with the minimal estimated response time.
3. Execute the selected algorithm and measure its execution time. Add the new measurement pair to the measurement pair list MPL_A of the executed algorithm A.
4. If the new measurement pair is the *RCR* new pair in the list, then the approximation function of the corresponding algorithm will be re-computed using the assigned statistical method.

2.2.2 Adaption to Load Changes

This section focuses on the adaption to load changes of the model. We start with necessary assumptions, then proceed with the discussion how the model can adapt to new load conditions.

Let A_{CPU} be a CPU algorithm and A_{GPU} a GPU algorithm for the same operation O.

The basic assumptions are:

1. Every algorithm is executed on a regular basis, even in overload conditions. That ensures a continuing supply of new measurement pairs. If this assumption holds then a gradual adaption of the approximation functions to a changed load condition is possible.
2. A change in the load condition has to last a certain amount of time, otherwise the model cannot adapt and delivers more vague execution time estimations. If this assumption does not hold, it is not possible to continuously deliver good estimations.
3. The load condition of CPU and GPU are not related.

As long as the assumptions are fulfilled, the estimated execution time curves[2] approach the real execution time curves if the load changes. Increase and decrease of the load condition on the side of the CPU is symmetric compared to an equivalent change on the side of the GPU. For simplicity, but without the loss of generality, we discuss the load adaption mechanism for the CPU side.

[2]An execution time curve is the graphical representation of all execution times an algorithm needs to process all datasets in a workload.

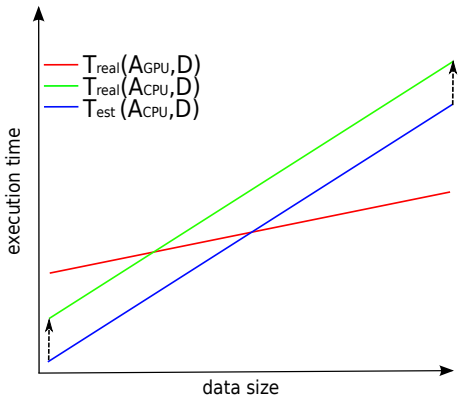


Figure 2: Example for increase load condition on CPU side

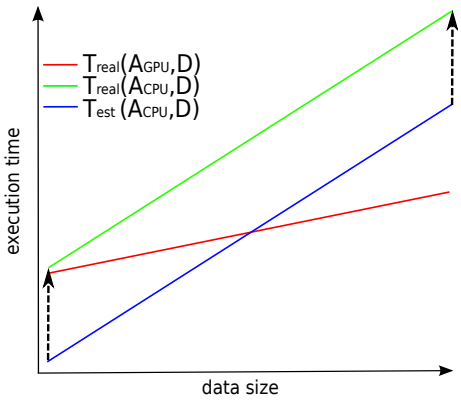


Figure 3: Example for strong increase of load on CPU side

Increasing Load Condition: If the load condition increases on the side of the CPU then the execution times of CPU algorithms increase and the real execution time curves are shifted upwards. In contrast, the estimated execution time curves stay as they are. We illustrate this situation in Figure 2. The model is adapted to the new load condition by collecting new measurement pairs and recomputing the approximation function. This will shift the estimated execution time curve in the direction of the measured execution time curve. Hence, the estimations become more precise. After at maximum *RCR* newly added measurement pairs for one algorithm the estimated execution time curve approaches the real execution time curve. This implies the assumption that a re-computation of approximation functions never has a negative impact on the estimation accuracy.

Decreasing Load Condition: The case of a decreasing load is mostly symmetric to the case of increased load. A decreased load on CPU side causes shorter execution time of CPU algorithms. The consequence is that real execution time curves are shifted downwards, whereas the estimated execution time curves stay as they are.

Limits of the Adaption: There is the possibility that the described load adaption scheme can break, if the load on CPU side increases or decreases too much. We consider the case, where the load increases too much, the other case is symmetric. In Figure 3, we illustrate the former case. Hence, the real execution time curve of A_{CPU} lies above the real execution time curve of A_{GPU}.

If this state continues to reside a certain amount of time, the estimated execution time curves will approach the real execution time curves. If the load condition normalizes again, then only algorithm A_{GPU} is executed, regardless of datasets that can be faster processed by algorithm A_{CPU}. The model is now stuck in an erroneous state since it cannot make response time minimal decisions for operation O anymore.

However, this cannot happen due to the assumption that every algorithm is executed on a regular basis, even in overload conditions. Hence, a continuing supply of new measurement pairs is ensured which allows the adaption of the model to the current load condition.

2.3 Decision Component

In this section, we describe the decision component of our model. Due to limited space, we introduce only one optimization criterion, namely response time, but other criteria like throughput are possible.

2.3.1 Response Time Optimization

If we optimize the operation execution for response time, we want to select the algorithm with the minimal execution time for the dataset D. In choosing the algorithm with the minimal execution time, we distribute the operation response time minimal to CPU and GPU. There is always one decision per operation that considers the size of the dataset that has to be processed.

Definition: Let a workload W be a tuple $W = (DS, O)$, where $DS = D_1, D_2, \cdots, D_n$ is a set of datasets D_i that are to be processed and O the operation to be executed.[3]

Goal: The goal is to choose the fastest algorithm $A_j \in AP_O$ for every dataset $D_i \in DS$ for the execution of operation O.

$$\text{min} = \sum_{D_i \in \text{DS}} T_{real}(A_k, D_i) \text{ with } A_k \in AP_O \quad (1)$$

Usage: The algorithm with the minimal estimated execution time for the dataset D is chosen for execution. The function choose_Algorithm (choose_Alg) chooses an algorithm according to the optimization criterion.

$$\text{choose\_Alg}(D, O) = A_j \text{ with} \quad (2)$$
$$T_{est}(A_j, D) = min(\{T_{est}(A_k, D) | \forall A_k \in AP_O\})$$

It is expected that the accuracy of the estimated execution times has a large impact on the decisions and the model quality.

Practical Use: The execution of the fastest algorithm reduces the response time of the system and results in better performance of a DBS. These response time optimization is necessary to accelerate time critical tasks[4]. Furthermore, it

[3]For simplicity, we only consider one operation per workload. However, a set of operations is more realistic and can be added in future work.

[4]A task is an operation in execution.

is possible to automatically fine-tune algorithm selection on a specific hardware configuration.

3. MODEL QUALITY CRITERIA

In this section, we present four model quality measures, namely average percentage estimation error, hit rate, model quality, and percentage speed increase.

3.1 Average Percentage Estimation Error

The idea of the average percentage estimation error is to compute the estimation error for each executed algorithm and the corresponding estimation value. Then, the absolute percentage estimation error is computed. Finally, the average of all computed percentage estimation errors is computed. This measure is also called relative error and is used, e.g., in [1].

3.2 Hit Rate

The idea of this measure is to compute the ratio of the number of correct decisions and the total number of decisions. A decision is correct, if and only if the model decided for the fastest algorithm. In the ideal case all decisions are correct, so the hit rate is 100%. In the worst case all decisions are wrong. Hence, the hit rate is 0%. The benefit of the hit rate is that it can provide statistical information about how many decisions are wrong. If the hit rate is X then every 1/(1-X) decision is incorrect. However, we cannot quantify the impact of an incorrect decision. This is addressed by the model quality. Note, that algorithms with very similar execution time curves can lead to a bad hit rate, although the overall performance is acceptable.

3.3 Model Quality

The idea of the model quality is to compute the ratio of the resulting execution times that a system using an ideal model and a real model needs to process a workload W. In the ideal case, the real model would be as good as the ideal model. Hence, the model quality is 100%. The worst case model would always select the slowest algorithm and provides a lower bound for the model quality.

Let $T_{DM}(W)$ be the time a system using a decision model (DM) needs to process a workload W. It is computed out of the sum of all algorithm execution times resulting from the algorithm selection of the decision model for the workload added with the overhead caused by DM. Let $T_{Oh}(DM, W)$ be the overhead the decision model DM causes. If a decision model introduces to much overhead, it can eliminate their gain. Hence the overhead has to be considered, which is the sum of the total time needed for computing estimation values and the total time needed to re-compute approximation functions. The time needed for all estimation value computation (EVC) for a workload W is $T_{\mathsf{EVC}}(W)$. The time needed to re-compute the approximation functions of all algorithms is $T_{RC}(W)$. Both measures are highly dependent on DM. Hence, we get the following formulas:

$$T_{Oh}(DM, W) = T_{\mathsf{EVC}}(W) + T_{RC}(W) \tag{3}$$

$$T_{DM}(W) = \sum_{D \in DS} T_{real}(\mathsf{choose\_Alg}_{DM}(D, O), D) + T_{Oh}(DM, W) \tag{4}$$

Let DM_{ideal} be the ideal decision model and let DM_{real} be the real decision model. Then, the model quality MQ is defined as:

$$MQ(W, DM_{real}) = \frac{T_{DM_{ideal}}(W)}{T_{DM_{real}}(W)} \tag{5}$$

This measure describes to which degree the optimization goal described in formula 1 is achieved. Note, that the ideal model is not introducing overhead ($T_{Oh}(DM_{ideal}, W) = 0$).

3.4 Percentage Speed Increase

The idea of the percentage speed increase (PSI) is to quantify the performance gain, if a decision model DM_i is replaced with a decision model DM_j.

$$PSI(DM_i \rightarrow DM_j, W) = \frac{T_{DM_i}(W) - T_{DM_j}(W)}{T_{DM_i}(W)} \tag{6}$$

If $PSI(DM_i \rightarrow DM_j, W)$ is greater than zero, DM_j had a better performance than DM_i and vice versa.

4. RELATED WORK

In this section, we present the related work. We consider analytical models for estimating execution times of GPU algorithms. Furthermore, we discuss learning based approaches to estimate execution times. Finally, we present existing decision models and provide a comparison to our approach.

Analytical Models Hong et al. present an analytical model which estimates execution times of massively parallel programs [8]. The basic idea is to estimate the number of parallel memory requests. The approach uses information like number of running threads and the memory bandwidth. Depending on their case study, relative estimation errors between 5,4% and 13,3% are observed.

Zhang et al. develop a model that should help optimizing GPU programs by arranging qualitative performance analyses [16]. They use a micro benchmark-based approach which needs hardware specific parameters, e.g., the number of processors or the clock rate. The model cannot adapt to load changes and therefore, it is not considered for use in our approach. The observed relative estimation error lies between 5% and 15%.

Learning based Execution Time Estimation Akdere et al. investigate modeling techniques for analytical workloads [1]. They estimate the execution behavior of queries on the basis of different granularities. They presented a modeling approach for the estimation on query level and operation level. The basic idea of their approach is to perform a feature extraction on queries and compute execution time estimations based on them.

Matsunaga et al. present a short overview over machine learning algorithms and their fields of application [11]. The goal is the estimation of the resource usage for an application. One considered resource is the execution time. The developed method PQR2 needs a few milliseconds for the computation of estimations. Since we need for n datasets and m algorithms n·m computations the time for a single computation should be less than 50μs to keep the overhead at an absolute minimum. Hence, the approach of Matsunaga et al. is to be investigated, whether it achieves good results

for a response time minimal operation distribution. This can be done in future work.

Zhang et al. present a model for predicting the costs of complex XML queries [15]. They use the statistical learning method called "transform regression technique". The approach allows a self-tuning query optimizer that can dynamically adapt to load condition changes. The approach of Zhang and our model are similar in basic functionalities. The difference to our approach is that Zhang et al. estimate the execution time of XML queries whereas our approach estimates execution times of single database operations to dynamically distribute them on CPU and GPU. Additionally, we use a different learning method that is optimized for computational efficiency.

Decision Models Kerr et al. present a model that predicts the performance of similar application classes for different processors [10]. The approach allows to choose between CPU and GPU implementation. This choice is made statically in contrast to our work, where an algorithm for an operation execution is chosen dynamically at runtime. Kerr et al. are using only parameters that are statically known before program execution. Hence, it allows no adaption to load changes in contrast to our model that allows load adaption.

Iverson et al. develop an approach that estimates execution times of tasks in the context of distributed systems [9]. The approach, similar to our model, does not require hardware specific information. They use the learning method k-nearest-neighbor, a non-parametric regression method. We use least squares and cubic splines that are parametric regression methods which need less time for computing estimations compared to non parametric regression methods. The goal of Iverson et al. is an optimal selection of nodes in a distributed system, where a task is executed. Unlike this approach, our work has the goal for optimal algorithm selection. It is possible to apply the approach of Iverson et al. on hybrid CPU/GPU platform. However, we consider, the GPU is a coprocessor of the CPU. Hence, a CPU/GPU platform is not a distributed system from our point of view. In a way, our model is less general than the model of Iverson et al.

5. CONCLUSION

In this paper, we addressed a current research problem, namely the optimal distribution of database operations on hybrid CPU/GPU platforms. Furthermore, we develop a self-tuning decision model that is able to distribute database operations on CPUs and GPUs in a response time minimal manner. We discuss the basic structure of the model and provide a qualitative argumentation of how the model works. Additionally, we present suitable model quality measures, which are required to evaluate our model. Our experiments show that our model is almost as good as the ideal model if the model parameters are set appropriately. We omit the evaluation due to limited space. We conclude that distributing database operations on CPUs and GPUs has a large optimization potential. We believe our model is a further step to address this issue.

In ongoing research, we use the defined quality measures to evaluate our model on suitable use cases. A further possible extension is using the model in a more general context, where response-time optimal decisions have to be made, e.g., an optimal index usage. Furthermore, an extension of the model from operations to queries is necessary for better applicability of our model. This leads to the problem of hybrid query plan optimization.

6. REFERENCES

[1] M. Akdere and U. Çetintemel. Learning-based query performance modeling and prediction. IEEE, 2012.

[2] ALGLIB Project. ALGLIB. `http://www.alglib.net/`, 2012. [Online; accessed 05-January-2012].

[3] P. R. Anthony Ralston. *A first course in numerical analysis*. dover publications, second edition, 2001.

[4] P. Bakkum and K. Skadron. Accelerating sql database operations on a gpu with cuda. GPGPU '10, pages 94–103, New York, NY, USA, 2010. ACM.

[5] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast computation of database operations using graphics processors. SIGMOD '04, pages 215–226, New York, NY, USA, 2004. ACM.

[6] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational query coprocessing on graphics processors. *ACM Trans. Database Syst.*, 34:21:1–21:39, December 2009.

[7] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander. Relational joins on graphics processors. SIGMOD '08, pages 511–524, New York, NY, USA, 2008. ACM.

[8] S. Hong and H. Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. *ACM SIGARCH Computer Architecture News*, 37(3):152, 2009.

[9] M. A. Iverson, F. Ozguner, and G. J. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. HPDC '96, pages 263–270, Washington, DC, USA, 1996. IEEE Computer Society.

[10] A. Kerr, G. Diamos, and S. Yalamanchili. Modeling gpu-cpu workloads and systems. GPGPU '10, pages 31–42, New York, NY, USA, 2010. ACM.

[11] A. Matsunaga and J. A. B. Fortes. On the use of machine learning to predict the time and resources consumed by applications. *CCGRID*, pages 495–504, 2010.

[12] NVIDIA. NVIDIA CUDA C Programming Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`, 2012. pages 1–5, Version 4.0, [Online; accessed 1-February-2012].

[13] H. Pirk, S. Manegold, and M. Kersten. Accelerating foreign-key joins using asymmetric memory channels. ADMS '11, pages 585–597. VLDB Endowment, 2011.

[14] S. Walkowiak, K. Wawruch, M. Nowotka, L. Ligowski, and W. Rudnicki. Exploring utilisation of gpu for database applications. *Procedia Computer Science*, 1(1):505–513, 2010.

[15] N. Zhang, P. J. Haas, V. Josifovski, G. M. Lohman, and C. Zhang. Statistical learning techniques for costing xml queries. VLDB '05, pages 289–300. VLDB Endowment, 2005.

[16] Y. Zhang and J. D. Owens. A quantitative performance analysis model for gpu architectures. *Computer Engineering*, pages 382–393, 2011.