Put All Eggs in One Basket: an OLTP and OLAP Database Approach for Traceability Data

Veneta Dobreva
Technische Universität München
Boltzmannstr. 3
D-85748 Garching, Germany
dobreva@in.tum.de

Martina-Cezara Albutiu
Technische Universität München
Boltzmannstr. 3
D-85748 Garching, Germany
albutiu@in.tum.de

supervised by Alfons Kemper (kemper@in.tum.de)
and Thomas Neumann (neumann@in.tum.de)

ABSTRACT

Accurate tracking and tracing of moving objects is an emerging trend in vertical industries like retail, logistics, and manufacturing. In order to monitor objects in business processes, more and more companies are deploying upcoming technologies like Radio Frequency Identification (RFID). Therefore, modern databases have to be able to cope with the challenges originating from the specifics of traceability data: efficient incremental update as well as efficient transactional and analytic ad-hoc querying and efficient storage of the data. Another requirement of business intelligence applications is to provide "real world awareness" [7] by using the latest information in the descision-making process. We therefore present an approach for efficient storing and managing of traceability data (on the example of RFID data), where the OLAP and OLTP components reside in one database and which meets the defined challenges. We discuss and analyze the experimental results and lessons learned and take them as a basis for our future research direction.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design

1. INTRODUCTION

Tracking and tracing of moving objects is a key aspect of many vertical industries like retail, logistics, and manufacturing. In order to monitor objects in business processes, more and more companies are deploying maturing technologies like Radio Frequency Identification (RFID) [4]. It allows automated recording of information about an object movement even without line of sight. In contrast to bar codes [11], RFID tags can be assigned to individual objects (instead of object groups). The Real World Awareness described by Claus Heinrich in [7] defines the process of extracting realtime information in order to gain a better insight into different kinds of business processes. Object movement and related traceability data are valuable business information, which gives companies a greater visibility into their supply chain and a better understanding of their production processes. Modern database systems are therefore willing to cope with the challenges that traceability data imposes. These are: efficient incremental update, efficient analytical ad-hoc querying, and efficient storage. Furthermore, real-time business intelligence applications are not only interested in "old" data for their decision-making processes, but need to involve the latest information as well. Hasso Plattner [12] discusses in his work the need for a common database approach for OLTP and OLAP since this "could make both components more valuable to their users" [12]. Former database approaches for handling traceability data [6, 8, 9] (in particular on the example of RFID data) try to fulfill one or more of the requirements mentioned above. However, they either ignore the OLTP part of the data and focus on the OLAP data, or have a hybrid approach, where OLTP and OLAP reside in different systems. We present a database approach for managing traceability data, which meets the challenges listed above, and merges the OLTP and OLAP components, so they reside in one system. Since traceability applications need to refer to the up-to-date information in order to conduct a good analysis for the decision process, our approach provides an "up-to-the-minute" currency.

The rest of the paper is organized as follows: In Section 2 we present the characteristics of traceability data. In Section 3 we give an overview of related work and discuss to what extent these approaches are applicable to our scenario. In Section 4 we present our approach for efficient data staging and query processing for traceability data. We evaluate this approach and report our results in Section 5. Finally, we summarize our experiences and conclude in Section 6.

2. DATA CHARACTERISTICS

We describe the specifics of traceability data in detail, in order to motivate why modern database systems have to cope with additional challenges when managing this data.

Why is storage and management of traceability data challenging?

In contrast to traditional warehouses, where updates occur

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research (IDAR 2010), June 11, 2010, Indianapolis, USA.
Copyright 2010 ACM 978-1-4503-0191-6/10/06... $10.00

EVENT		
oid	**sid**	**ts**
□	s_1	t_1
■	s_1	t_1
△	s_2	t_1
...	...	...

Figure 1: Example data for the naive data model.

only at predicted time intervals, e.g., at night or during the weekends, traceability information has to be updated "on demand". As soon as new events arrive, the data staging process is triggered. Accessing the most current information is important not only for OLTP applications that are often interested in locating an object by searching for its latest position, but it is also an upcoming requirement of decision processes based on business intelligence applications executing OLAP queries. Analytical processing of business data is often applied in tactical problem solving where decisions are made on a daily basis. Therefore, we need an efficient incremental update (*data staging*) for traceability data as well as acceptable response times for both OLTP and OLAP queries. A typical query in a traceability scenario determines the path of an object (*pedigree query* [1]). Another group of common queries for this scenario are the *contamination queries*. They reconstruct only a part of the object's history, which is of interest for a particular use case or application, e.g., they determine which products were stored together with product X in the stock. If these products are incompatible an alert should be produced by the application.

Another challenge is the vast amout of data that the sensors produce continually. For a medium-sized enterprise this would be about 18 million events per day. (The estimation relies on the assumption that events are produced within 10 hours each day and on data published by BMW [2] from which we derive an average event generation frequency of 500 events/second.) The data has to be filtered and aggregated in order to be reasonably stored and managed. If we stored each event in one table, this table would "explode" and we would not be able to retrieve information efficiently. Figure 1 shows the data schema of the naive approach which is referenced in different works on RFID. This approach simply stores all data (consisting of the object's identifier *oid*, the sensor *sid* which reported the event and the timestamp *ts* when the event was generated) in one huge table. As we will demonstrate in Section 5 the naive approach has a poor scalability and the query response times for most of the traceability queries are not acceptable.

In the context of traceability (e.g., RFID data), the challenge of false or missing event data is often considered. However, this issue is beyond our scope. In this work we assume that the generated events are correct and complete.

What are the challenges to a database design posed by traceability data?
The challenges posed by traceability data form the criteria (metric 1 to 6) by which we measure the applicability of different approaches. **Metric 1:** As already mentioned, supporting incremental updates is essential for traceability applications. In order to realize the real world awareness, efficient data staging mechanisms must be guaranteed. **Metric 2:** Dynamic insertion of new sensors has to be provided, due to the fact that the sensor landscape can change over time. **Metric 3:** In most traceability scenarios objects move in groups and split into smaller groups. This tree-like object movement has to be mapped on the data model. **Metric 4:** More complex scenarios demand more complex object movements like the graph-like object movement. That kind of movement can be seen in a post office, where parcels that come from a lot of different small post offices are gathered in one central post office. **Metric 5:** Consider the post scenario again. If a mail is returned to its sender, a cycle occurs in our movement graph. Therefore we need storage solutions that can deal with cyclic object movements. **Metric 6:** Some objects may "stay" at a location and not move any further. This implies that paths of different length occur in the movement tree or graph.

3. RELATED WORK

In this section we discuss the current state of the art for storing and managing RFID traceability data in relational databases.

The data model of Gonzalez et al. [6] aggregates and compresses the path data of objects based on the observation that objects move in clusters. This approach also motivates that the movement of clusters in, e. g., the pharma-scenario, can be visualized as a tree: products move in large groups and split into smaller groups as they move from the manufacturer to different pharmacies (fulfills metric 3). Even if this data model was not optimized for a graph-like movement, it is still possible to realize it by storing data redundantly (metric 4). Metric 2 and 6 are implemented as well. The approach of Gonzalez et al. is a typical warehouse approach based on the idea of materializing the hierarchical relations between the objects' paths. Because of these hierarchical relations it is not possible to implement cycles in a movement graph, so metric 5 is not supported. The database design enables a data staging process, i.e. metric 1 is fulfilled. However, this database design is not suitable for managing big amounts of data efficiently due to the hierarchical identifier that has to be processed for the queries and has to be stored as a VARCHAR in the database. Furthermore, this approach is not optimized for OLTP queries, it focuses on OLAP queries.

The approach of Krompass et. al. [8] stores the complete path of an object, taking advantage of the cluster feature of the data. This means that objects that move together share the same path. It provides path information redundantly, though in order to improve performance for some traceability queries. The database design provides one table for storing the last location of an object and one table that materializes the path for a cluster of objects. This model is optimized for a tree-like object movement, however it can also be applied to a graph-like movement and even to a cyclic graph movement (supporting metrics 3, 4 and 5). Efficient incremental update is provided, which is required by metric 1. The database design is flexible, so we can insert new nodes (metric 2) and handle object's paths of different length (metric 6). The approach is a hybrid model, which has two components: an OLTP and an OLAP part. The cache com-

ponent contains the most recent events in order to answer OLTP queries. Aged events are propagated to the warehouse component where OLAP queries are running. The difference to our approach is that we aim to have OLTP and OLAP in one database, in order to fulfill the requirements of real-life business intelligence applications' analysis.

The database design of Lee and Chung [9] uses the mathematical characteristics of prime numbers for encoding and decoding an object's path. This results in a very compact representation of the path information. Using a path encoding scheme, the object's path (a sequence of sensors) is materialized. To encode a path the authors assign a prime number to each location and build the product of all locations which occur in the object's path. This product of prime numbers can be uniquely defactorized using the Fundamental Theorem of Arithmetic [5], i.e. we can decode the locations participating in a path. In order to determine the ordering between the locations the Chinese Remainder Theorem CRT [3] is applied. The encoding scheme can handle a tree-like and a graph-like object movement (supports metrics 3 and 4), but has the drawback that it cannot handle cycles in the object movement (metric 5), since CRT could not be applied in this case. Storing paths of different length (metric 6) is supported, though. Further, the authors use a so called region numbering scheme, which is described in detail in [9], to construct a time tree. It gives information about the locations visited by an object at a particular time and is represented as a table in the database design. The region numbering scheme represents a limitation of this approach, though, since it is not possible to incrementally update the database design. This means that metric 1 is not supported, as well as metric 2, which requires the dynamic insertion of new nodes. For instance if objects move at different times along the same locations, a new time tree has to be built every time. This means that data staging cannot be performed and the data model can only be applied if the object movement is known in advance, which is a major drawback for traceability applications. Due to these disadvantages this approach is not suitable for storing traceability data of moving objects.

Based on the classification presented in Section 2, we compare the described approaches concerning their applicability for the traceability scenario. The approach of Krompass et. al. [8] fulfills all the checkpoints. For more complex movements however, redundant information has to be stored. The database design of Gonzalez et al. [6] has the same problem. It is also not suitable for handling cycles in the object movement, because of the hierarchical identifier. In addition to the drawbacks of the above two approaches, the data model of Lee and Chung [9] does not support an incremental update and dynamic insertion of new sensors. Our approach that is presented in Section 4 is designed to meet the requirements that traceability data poses.

4. AN OLTP AND OLAP APPROACH FOR TRACEABILITY DATA

We aim at creating a database schema that fulfills the requirements of traceability applications described in 2. Our approach is based on a new path encoding using a Bloom filter which enables the materialization of the object's history and also functions as an index. In order to provide an efficient data staging we pursue an append-only approach, i.e., there exist only inserts and no updates, and consolidate the database regularly.

OLTP

oid	rdr	ts	bloom
□	s_4	t_n	101
■	s_4	t_n	101
◯	s_3	t_1	010
...	...	...	...

Region1

oid	rdr	ts
□	s_1	t_1
■	s_1	t_1
△	s_2	t_2
...	...	...

Region2

oid	rdr	ts
◯	s_3	t_1
◇	s_3	t_1
▲	s_4	t_2
...	...	...

Region3

oid	rdr	ts
□	s_4	t_n
■	s_4	t_n
△	s_5	t_n
...	...	...

Figure 2: Bloom filter approach

4.1 Data Model Description

Figure 2 shows our database design consisting of the OLTP table where the most current data is kept and the REGION tables where (historical) path information is stored. A region is a geographical unit that comprises the sensors located in it and may represent a country, a city, or a single factory, depending on the use case. In the OLTP table the last occurrence of an object (idendified by its *oid*), the sensor that scanned it (*rdr*) and the timestamp (*ts*) when the object passed the sensor are stored. Using this generic data representation, metrics 2, 3, 4, 5, and 6 can easily be supported. The Bloom filter (more specifically, the positions r_i where a 1 occurs) in the OLTP table points to the regions in which an object was scanned, i.e., the corresponding *REGION* tables that hold the information about those (potentially outdated) read operations. Each *REGION* table has the schema of the naive approach and stores events produced by sensors from the respective region. We employ a Bloom filter with a size equal to the number of regions which is reasonable for a medium-size business. Thus, we do not have to consider false positives.

Our approach efficiently answers both OLTP and OLAP queries. As the name indicates, the *OLTP* table serves OLTP requests which require up-to-date information. A typical OLTP request in a traceability scenario is to determine the last position of an item. OLAP queries, e.g. asking for all readers an item has passed, can be answered by joining the *OLTP* table and the *REGION* tables the Bloom filter points to. Queries examining a data flow in only one region read data only from the corresponding region table.

4.2 Data Staging

An efficient incremental update (metric 1) is one of the biggest challenges when designing a data model for traceability data. In order to achieve this, we do not insert each single event, but process a batch of events. The more events a batch contains the higher the throughput. However, a database supporting OLTP has to contain the most current

Algorithm 1: Algorithm `processBatch`

input : A set of SQL statements S

```
1  create a temporary table T;
2  forall the statements s ∈ S do
3  |  rewrite s as insert i and append i to file F;
4  end
5  BULK INSERT data from F to T;
6  forall the epc values e ∈ T do
7  |  forall the tuples d: d.epc = e.epc ∧ d.timestamp <
   |  e.timestamp;                          /* outdated tuples */
8  |  do
9  |  |  write d to deathlist file D;
10 |  end
11 end
12 U ← T ⋈ OLTP;                          /* tuples to be updated */
13 forall the tuples u ∈ U do
14 |  write u to deathlist file D;
15 |  forall the tuples t ∈ T with t.epc = u.epc do
16 |  |  write t to oltp file O with updated bloom value;
17 |  |  write t to corresponding region file R_i;
18 |  end
19 end
20 forall the tuples i ∈ T ∧ i ∉ U;        /* tuples to be inserted */
21 do
22 |  forall the tuples t ∈ T with t.epc = u.epc do
23 |  |  write t to oltp file O with bloom value pointing to one
   |  |  region;
24 |  |  write t to corresponding region file R_i;
25 |  end
26 end
27 forall the files R_i do
28 |  BULK INSERT data from R_i to corresponding region table;
29 end
30 BULK INSERT data from O to OLTP table;
31 BULK INSERT data from D to Deathlist;
```

data, so there is a trade-off between batch size and data update latency. We consider a batch size of 5000 events/second to be a reasonable trade-off for our scenario. Since updates of indexed data are more expensive than inserts we further replace the updates in the *OLTP* table by inserts. We use two auxiliary tables in order to make use of the efficient batch and insert processing of database systems. The temporary *TEMP* table holds each batch before it is processed. The *Deathlist* table contains outdated events.

In the implementation of batch processing described in Algorithm 1 we exploit the database's efficient BULK INSERT and join computation. In lines 1 to 5 the temporary table T is created in the database, all tuples to be processed are written to a file F and the data contained in F is loaded to T. In lines 6 to 11 all tuples in the processed batch which represent already outdated data are identified and written to the deathlist file D. If multiple subsequent reads of the same item are found in the same batch, only the last read (with the most current timestamp) is valid OLTP data, the rest is historical data. In line 12, T is joined with the OLTP table, thereby determining the logical updates within the current batch. All tuples within the join result have to be treated as updates (lines 13 to 19), while the rest of the batch tuples represent "real" inserts (lines 20 to 26). For all tuples to be updated, the current tuple is written to the deathlist file (as the new tuple now is the most current one). Finally, the *OLTP* table, the *REGION* tables, and the *Deathlist* are loaded.

As described above, OLTP queries have to be answered using only the most current data, which is obtained by computing

Algorithm 2: Algorithm `consolidate`

```
1  O ← OLTP − Deathlist;
2  forall the tuples o ∈ O do
3  |  write o to file F;
4  end
5  drop OLTP table;
6  drop Deathlist table;
7  create OLTP table;
8  BULK INSERT data from F to OLTP;
9  create Deathlist;
```

the set difference between the *OLTP* table and the *Deathlist*. In order to keep the overhead as small as possible and to avoid very large tables, we consolidate the *OLTP* table and the *Deathlist* from time to time (i.e., after a certain number of batch inserts). The consolidation procedure (which equals a delete from the *OLTP* table) is described in Algorithm 2.

5. BENCHMARKS

We present some experiments comparing the naive approach and the Bloom filter approach implementations on a commercial row-store database. The results show that our approach succeeds in handling a continuous event stream as expected in a medium-size business and even outperforms the naive approach in query processing.

5.1 Traceability Data and Queries

Traceability events are triples of the form (epc, rdr, ts), where *epc* is the EPC-code of an item, *rdr* is the sensor that read the EPC-code, and *ts* is a timestamp denoting the time when the sensor read the EPC code. As there is no publicly available data or experimental environment for traceability data, we use an event generator to simulate the movement of items through different regions by creating event tuples. Thereby, 20% of the events represent new objects, and 80% are positional updates of these objects. The average data generation frequency relevant to our scenario is 500 events/second (see Section 2). Therefore, we conduct our experiments with this event frequency (generated by two client threads) if not stated otherwise.

Figure 3 lists the queries we employed in our benchmark and their semantic. Queries 1 to 4 are OLTP-style queries that may be submitted to the system for almost every object that is tracked. Queries 5 through 11 are OLAP-style queries, processing big amounts of data and usually long-running. OLAP queries are typically submitted for report generation or decision making and occur less often than OLTP queries.

5.2 Experiments

We report benchmark results for experiments conducted on the row-store database implementing the Bloom filter approach (denoted "Row-store" in the figures) and the naive approach ("Naive").

The database runs on a 64bit-Red Hat Enterprise Linux server with two Intel Xeon 3.16GHz CPUs, 8GB main memory, and 8 SAS disks associated with RAID level 5.

5.2.1 Data Staging

We first examined only the data staging procedure of our approach without any queries being processed in parallel. As

qid	Query
q_1	Last location of an object
q_2	The pedigree (complete path) of an object
q_3	The number of objects scanned by a certain sensor
q_4	A list of objects scanned by a sensor within a time interval
q_5	A list of objects, which were scanned by sensors s1 and s2 (no order)
q_6	A list of objects, which were scanned by sensors s1 and s2 in this order
q_8	A list of objects that were at sensor s, together with object x within a certain time interval
q_{10}	Listing the number of all objects scanned by all the readers in 10 regions, ordered by region, reader, and a time interval of a second
q_{11}	Listing the number of all objects which were scanned by the sensors s1, s2, and s3 in this order aggregated per second

Figure 3: Queries for Traceability Scenario.

we motivated above, a suitable system has to be able to handle an average data arrival frequency of 500 events/second. We thus ran benchmarks with this fixed event generation frequency and found out that the different data models and database systems are able to cope with the arriving events in the data staging process. However, as there might be peaks in event generation, the system must be able to handle event frequencies greater than the expected ones, that is why we analyze the upper limit the database designs can handle. The naive approach has a very high insert throughput (15466 events/second), since the events do not need to be transformed in any way, but are directly inserted into the database. The Bloom filter implementation has a throughput of only 2240 events/second due to the overhead of Bloom filter processing during data staging. The conclusion would be to fall back on the naive appraoch in periods of very high loads. However, the query response times will show that the naive approach does not support efficient query processing and is therefore not an appropriate long-term solution.

5.2.2 *Data Staging and Query Workload*

We also analyzed whether the specified frequency can be kept while executing a mixed workload consisting of OLTP and OLAP queries. Thereby, the workload is designed as follows: two insert clients continuously insert events during one hour, thus generating a total of 1.8 million events. The query clients start submitting queries after the benchmark has been running for 5 minutes, so that approximately 150000 events are preloaded before the first query arrives at the database. Each query type is handled by one query client. Depending on the query type, a think time of 1 respectively 60 seconds is set up for OLTP and OLAP queries. The clients submit one query, retrieve the result and wait for the think time before submitting the next query.

Figure 4 shows the average response times of the OLTP queries. The row-store Bloom filter approach has better response times for all OLTP queries except $q2$. For this query, the Bloom filter approach is a factor 2 slower than the naive approach. This is due to the Bloom filter processing which requires a two-step communication of the application and the database for determining the relevant regions and querying the corresponding tables. Determining the last position of an object ($q1$) has nearly the same response time for both approaches, since both times we sort the tuples by ts and select the most current event. The contamination queries $q3$ and $q4$ are much more efficient for the Bloom filter approach, because the data is segmented in smaller tables, compared to the big table of the naive approach.

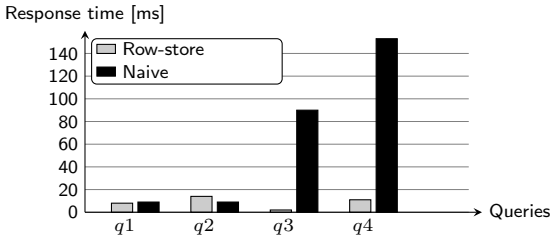

Figure 4: Mixed Workload: OLTP Queries

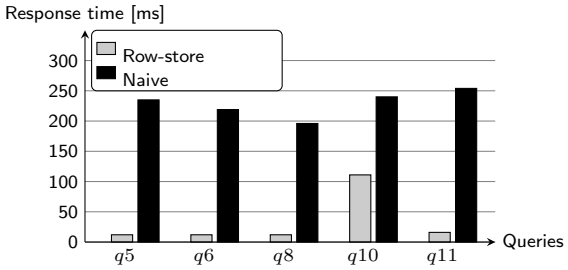

Figure 5: Mixed Workload: OLAP Queries

The response times of the OLAP queries are presented in Figure 5. All shown queries except query q_{10} are executed a factor of 20 slower on the naive approach schema. This is due to the much higher amount of data the naive approach has to process for each query. Query q_{10} takes half the time on the row-store database schema. The query processes a union over 10 subquery results.

6. CONCLUSIONS AND FUTURE WORK

RFID is becoming a widespread adopted technology for seamlessly tracing products, possibly across a global supply chain. It provides manufacturers with up-to-date information about their whole network. However, this also poses the challenge to efficiently store and manage big amounts of data which traceability technology produce. The Bloom filter approach we developed is derived from the naive approach and optimized for efficiently handling both OLTP and OLAP queries while providing a sufficient data staging performance. It makes use of a new path encoding technique (Bloom filter), which organizes the data hierarchically and provides effecient data retrieval. In our first experiments we compared the Bloom filter approach to the naive approach often referenced in RFID works. We further examined our approach implemented on a commercial column-store database systems. However, the results indicate that the column-store database is not suitable for combining efficient data staging and fast OLTP and OLAP processing as it could handle the insert as well as the query workload separately but failed when both workloads were executed concurrently. The

lessons learned can be summarized as follows:

Benefit of Bloom filter: There is a trade-off between the event processing throughput and the query processing. When comparing the Bloom filter approach to the naive approach, the Bloom filter approach achieves a lower event processing throughput because of the overhead incurred by the Bloom filter processing. However, this decrease in data staging performance is negligible in our scenario as we are considering medium-sized businesses with event processing requirements falling into our results. Further, when it comes to query processing, the Bloom filter approach clearly outperformes the naive approach. An important point concerning the Bloom filter processing is that the database systems we employed do not support the direct extraction of the right regions out of the Bloom filter, so that query 2 is processed in two steps which is a drawback for its performance.
Benefit of splitting: In particular when regarding traceability scenarios where the majority of queries request information about a certain object or sensor, the Bloom filter approach shows to have performance advantages in terms of shorter query response times. Those can be attributed to the following factor: the splitting of information is beneficial for queries interested only in a particular segment of the hierarchically structured data as only a fraction of the data (e.g., one *REGION*) table is scanned.

Our results show, that for some scenarios our Bloom filter approach implemented on a row-store database is feasible. However, the achieved throughput in data staging is only sufficient for medium-sized businesses. Further, we cannot fully exploit the advantages of the Bloom filter data structure as this feature is not implemented in the database systems themselves and thus requires a two-step communication of application and database system: the Bloom filter is extracted from the database, the corresponding regions are determined inside the application, and a new query accessing the relevant *REGION* tables is submitted to the database. Implementing this inside the database would improve both data staging (where Bloom filters have to be re-computed for updates) and query processing performance (especially the pedigree query). Thus, regarding traceability data, we agree with Stonebreaker that the "one size fits all" era in database design comes to an end [13].

Therefore, we intend to develop a new RISC style database system focussing on the characteristics of traceability data (see Section 2), which is based on the RDF-3X engine for scalable management of RDF data by Neumann et al. [10]. This database system is optimized for efficiently handling OLTP queries with a high selectivity on huge amounts of RDF data. The data model ressembles the naive approach storing all information in one huge table. By making use of exhaustive indexes for all permutations of the table columns, RDF-3X supports very fast merge joins and range selections. As the indexes are highly compressed, the additional space requirements are negligible. Further, the system provides good support for efficient updates by means of a staging architecture. Overall, the RDF-3X engine fulfills most of the requirements that we presented in Section 2. In order to adapt the system to our requirements we have to focus on the following changes: compared to a traceability scenario, RDF data is relatively static, i.e. updates do not occur as often as this is the case in our RFID example. Thus, index building and maintenance have to be revised for our use case. Further, OLAP queries processing great parts of the database cannot exploit the good selectivity performance provided by the indexes. We need to examine to what extent this will affect our query performance. Besides, the RDF-3X system uses data dictionary compression which may be suitable for our scenario as well, e.g., for the sensors in our scenario (as there exists a limited number of them). For the EPC-codes and timestamps new values are continuously produced which provides different challenges for compression. Our ongoing work aims therefore at adopting the RDF-3X system for traceability data and creating a dedicated RSIC-style DBMS for efficient traceability data management.

7. REFERENCES

[1] R. Agrawal, A. Cheung, K. Kailing, and S. Schonauer. Towards Traceability across Sovereign, Distributed RFID Databases. In *10th Intl. Database Engineering and Applications Symposium (IDEAS)*, 2006.

[2] BMW. Quarterly Report to 30 September 2009. `http://www.bmwgroup.com`. accessed February 19, 2010.

[3] Chinese remainder theorem. `http://en.wikipedia.org/wiki/Chinese_remainder_theorem`, 2008.

[4] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. Wiley Publishing, 2003.

[5] Fundamental theorem of arithmetic. `http://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic`, 2008.

[6] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In *Proc. of the 22nd Intl. Conf. on Data Engineering (ICDE)*, 2006.

[7] C. Heinrich. *RFID and Beyond: Growing Your Business Through Real World Awareness*. Wiley Publishing, 2005.

[8] S. Krompass, S. Aulbach, and A. Kemper. Data Staging for OLAP- and OLTP-Applications on RFID Data. In *Datenbanksysteme in Business, Technologie und Web (BTW)*, 2007.

[9] C.-H. Lee and C.-W. Chung. Efficient Storage Scheme and Query Processing for Supply Chain Management using RFID. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2008.

[10] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *The VLDB Journal*, 19(1):91–113, 2010.

[11] T. Pavlidis, J. Swartz, and Y. P. Wang. Fundamentals of bar code information theory. *Computer*, 23:74–86, 1990.

[12] H. Plattner. A common database approach for oltp and olap using an in-memory column database. In *SIGMOD '09: Proc. of the 35th SIGMOD Intl. Conf. on Management of data*, pages 1–2, New York, NY, USA, 2009. ACM.

[13] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era. In *VLDB '07: Proc. of the 33rd Intl. Conf. on Very large data bases*, pages 1150–1160. VLDB Endowment, 2007.