In-Memory Columnar Databases - Hyper (November 2012)

Arto Kärki, University of Helsinki, Helsinki, Finland, arto.karki@tieto.com

Abstract— **Relational database systems are today the most common database systems. The business requirements are today so demanding that there has been need for two different technical approaches: OLTP (OnLine Transaction Processing) and OLAP (OnLine Analytic Processing). OLTP consists of tuples that are arranged as rows which are stored in blokcs or tables. Indexing allows fast retrieval of single tuple, but as the number of rows increases the slower the retrieval comes. OLAP is organized often in star schemas with fact and dimension tables.**

The increasing main memory capacities has lead to in-memory databases of relational type. However the tests for row storage in-memory databases were not showing any significant advantages over leading relational type databases. Column oriented OLAP has already been used for years, but increased main memory capacities has made column oriented more interesting solution. This report introduces some examples of in-memory database with column oriented OLAP.

***Index Terms*— Hyper, columnar databases, in-memory, OLAP.**

I. Introduction

THE relational database systems are today mostly separated in two different technical solutions because of the increased number of tuples and for performance issues. OLTP is designed for fast row inserts, updates and selections. OLAP instead is designed for long lasting queries. An OLAP database is populated with rows from OLTP database. This means complicated data extracting, transferring and loading mechanism. This mechanism is usually slow and can be made only once a day submitting batch jobs during night time. Thus that is the challenge, how one could avoid this overhead and merge OLTP and OLAP back in the same database system and manage to preserve the fast OLTP transactions and as well as achieve up-to-date data for OLAP queries.

This seminar report highlights some in-memory column database approaches. All examples describe solutions for one system OLTP and OLAP. In section 2 is given an overview of Hasso Plattner's article [6] A Common Database Approach for OLTP and OLAP Using an In-Memory Column Database. Section 3 presents Hyper database system article [4], Hyper: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. Section 4 contains compression overview of Hyper database systems article [2], Compacting Transactional Data in Hybrid OLTP&OLAP Databases.

II. A common database approach

This section is based on Hasso Plattner's work [6].
Today's computers provide enormous amout of computing power. Blades with 8 CPUs and 16 cores per CPU gives us 128 computing units. To optimize the use of this computing power we have to understand memory hierarchies, cache sizes, and how to enable parallel processing within one program. We can achieve memory savings with column store using the more efficient vertical compression along columns. According to Hasso Plattners analyses of real systems with customer data, most applications in enterprise computing are actually based on set processing and not direct tuple access. So, this lead to substantial benefit when using column store data arrangement. The main difficulty with parallel processing is how the programs can be divided into equal-sized pieces, which then can be processed in parallel without much synchronization. The scan operation can be split easily into equal parts and distributed to multiple cores. All calculations on tuple level will automatically be parallized, since they are completely independent of each other.
The introduction of indices is not necessary because the scanning speed is so enormous, especially if parallel processing across multiple cores is active. On current CPUs, we can expect to process 1 MB per ms and with parallel processing on 16 cores more than 10 MB per ms. With this kind of speed, we can scan 2.5 million tuples for qualification 1 ms. And therefore there is no need for primary key index for most of the tables anymore but we can use the full column scan instead. Column storage is so well suited for modern CPUs that the full scope of the relational algebra can be used without shortcomings of performance. It is important to note that every attribute now represents a potential index. The hard disk is used only for transaction logging and snapshots for fast recovery. In fact, disk has become yesterday's tape.

Next claim: column storage is suited for update-intensive applications. It is said that column store databases are expensive to update. Despite the fact that typical enterprise

systems are not really update-intensive, by using insert-only and by not maintaining totals, we can even reduce these updates. Since there are less updates, there are less locking issues and the tables can be more easily distributed (partitioned) horinzontally across separate computing units (blades) with a shared nothing approach. Having basically eliminated updates, we now need to consider only inserts and reads. With the insert approach we can simplify database structure and can reduce the amount the database objects. The insert-only approach and calculation algorithms executed on the fly replace all indices, materialized views, and change history.

Consequences of the insert-only approach. The insert-only approach has consequences on how locking is handled both on the application- and database level. The application-level locks are implemented using an in-memory data structure. With the insert-only approach the update of tuples by the application could be eliminated with the exception of binary status variables. Each inserted tuple carries timestamp of its creation and another timestamp for update. Only the latest version of a tuple carries no update timestamp and is therefore easily identifiable. Since multiple queries can coincide with inserts and updates, extreme care has to be taken to avoid too much locking on table-, column- or dictionary level. Since the inserts are realized as an append to the delta store no exclusive lock should be necessary. As a general rule the database system should perform each task with maximum speed, even occupying all resources (e.g. CPU cores) in order to reduce the potential for collisions and increasing management overhead.

Column storage is superior to row storage with regards to memory consumption. Under the assumption to build a combined system for OLTP and OLAP data has to be organized for set processing, fast inserts, maximum (read) currency and low impact of reorganization. In column store, the compression via conversion of attribute values and the complete elimination of columns with null values only is very efficient but can be improved: all characters blank, all characters zero, and decimal floating point zero as null values. Comparing the memory requirements of column and row storage of a table, the difference in compression rate is obvious. Various analyses of existing customer data show a typical compression rate of 20 for column store and a compression rate of 2 for (write-optimized) row storage on disk. A column storage allows us to eliminate all materialized views (aggregates) and calculate them algorithmically on demand.

What happens to typical dataentry transactions. The database update is reduced to a mere insert. No indices need to be maintained and for journal entries, such as customer orders, stock movements etc., no update of aggregates takes place. As a result, the throughput of transactional data entry will improve.

The impact on application development. Applications should use relational algebra and the extended SQL-features to delegate as much of the logic to the database level and the stored procedures. The strict use of minimal projections is recommended. The high performance of the database makes caching of data on the application level largely superfluous. The option to partition tables in multiple dimensions helps to achieve minimum response times even larger tables.

Plattner's ongoing research efforts are concentrated on creating a benchmark for combined OLTP and OLAP systems, which is derived from real customer systems and data. They expect that the impact on management of companies will be huge, probably like the impact of Internet search engines on all of us.

III. HYPER: HYBRID OLAP & OLTP

This section describes HyPer architecture and is based on paper by Kemper, Neumann, Funke, Leis and Mühe [4].

The two workloads of online transaction processing (OLTP) and online analytical processing (OLAP) present different challenges for database architectures. Hyper is a efficient hybrid system, that can handle both OLTP and OLAP simultaneously by using hardware-assited replication mechanism to maintain consistent snapshots of the transactional data.

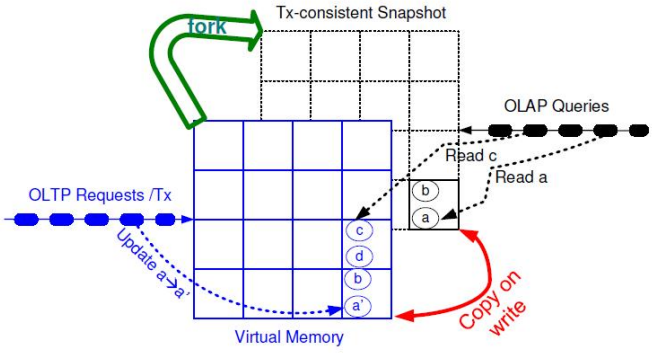

Fig. 1. Virtual Memory Snapshots to separate OLTP & OLAP.

Hyper's performance is due to the following design choices: in-memory data management, SQL table definitions are transformed into simple vector-based virtual memory representations – which constitutes a column-oriented physical storage scheme, OLAP processing is separated from the mission-critical OLTP transaction processing by fork-ing virtual memory snapshots (see Figure 1), transactions and

queries are specified in SQL and are efficiently compiled into LLVM assembly code [5], parallelism is this serial execution model is achieved by logically partitioning the database and admitting multiple partition-constrained transactions in parallel, use of hash indexes for exact match (see Figure 2 and 3).

```
select   *
from     R1,R3,
         (select   R2.z,count(*)
         from      R2
         where     R2.y=3
         group by R2.z) R2
where    R1.x=7 and R1.a=R3.b and R2.z=R3.c
```

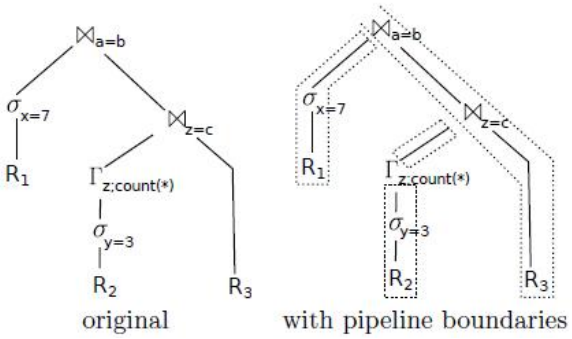

Fig 2. Example Query and Execution Plan.

initialize memory of $\bowtie_{a=b}$, $\bowtie_{c=z}$, and Γ_z
for each tuple t in R_1
  if $t.x = 7$
    materialize t in hash table of $\bowtie_{a=b}$
for each tuple t in R_2
  if $t.y = 3$
    aggregate t in hash table of Γ_z
for each tuple t in Γ_z
  materialize t in hash table of $\bowtie_{z=c}$
for each tuple t_3 in R_3
  for each match t_2 in $\bowtie_{z=c}[t_3.c]$
    for each match t_1 in $\bowtie_{a=b}[t_3.b]$
      output $t_1 \circ t_2 \circ t_3$

Fig 3. Compiled query for Fig. 2.

Hyper relies on the ability to efficiently create a transaction-consistent snapshot of the database. There are also different mechanisms for snapshot creation which do not diminish OLTP performance. These techniques are hardware page shadowing approach also used by Hyper, tuple shadowing which is used by SolidDB and a variant of the so called ZigZag approach as evaluated by Cao [1]. These three techniques can be subdivided by the method they use to achieve a consistent snapshot while still allowing high throughput OLAP transactions on the data. The hardware page shadowing approach uses a hardware supported copy on write mechanism to create a snapshot. In contrast to that, tuple shadowing as well as the twin object approach use software mechanisms to keep a consistent snapshot of the data intact while modifications are stored separately.

Database compaction: hot/cold clustering and compression. While some of the existing work addresses the problem of updates in compressed databases, none of the techniques developed for OLAP systems can be easily adapted for OLTP-style workloads. To avoid hurting transactional throughput, OLTP engines often refrain from compressing their data and thus waste memory space. The lack of compression becomes even more impeding, when the database system is capable of running OLAP-style queries on the transactional data, like the HyPer system. In this scenario, compression can not only reduce memory consumption significantly due to columnar storage, but also promises faster query execution.

Approaches that maintain two separate data stores, an uncompressed store for freshly inserted data and a compressed store for older data, require costly merge phases that require exclusive locking of tables when moving data to the compressed store. They also tend to complicate and slow down query and transaction processing. HyPer approach to compression in hybrid OLTP&OLAP column stores is based on the observation that while OLTP workloads frequently modify the dataset, they often follow the working set assumption: only a small subset of the data is accessed and an even smaller subset of this working set is being modified. HyPer uses a lightweight monitoring component to observe accesses to the dataset and identify opportunities to reorganize data such that it is clustered into hot and cold parts. After clustering, the database system compresses cold chunks to reduce memory consumption and streamline queries.

In-memory technology has facilitated a reunion of OLTP and OLAP systems by separating the two disparate workloads via snapshotting. So, after all, a “one size fits for all” system appears possible. In the comparison of snapshot techniques it was demonstrated the benefits of hardware-assisted shadow paging over software approaches. The hot/cold clustering stores frequently accessed tuples together on regular memory pages while cold, immutable tuples can reside on huge pages. This leads to the advantageous combination of page table size (and thus snapshot creation costs) and replication overhead. In addition, it allows to compress the majority of the database without causing OLTP throughput declines.

IV. HYPER: COMPACTING TRANSACTIONAL DATA

This section on Compacting Transactional Data is based on paper by Funke, Kemper and Neumann [2]. It shortly describes design of HyPer's data compression.

Despite memory sizes of several Terabytes in a single commodity server, RAM is still a precious resource: Since free memory can be used for intermediate results in query processing, the amount of memory determines query performance to a large extent. Modern in-memory database systems with high-performance transaction processing capabilities face a dilemma: On the one hand, memory is a scarce resource and these systems would therefore benefit from compressing their data. On the other hand their fast and lean transaction models penalize additional processing severely which often prevents them from compressing data in favor of transaction throughput. As a result of this dilemma, OLTP engines often refrain from compressing their data and thus waste memory space. The lack of a compact data representation becomes even more impeding, when the database system is capable of running OLAP-style queries on the transactional data, like the HyPer system.

Some related work has been done as Héman et al. proposed Positional Delta Trees [3]. They allow for updates in ordered, compressed relations and yet maintain good scan performance. Binning et al. propose ordered-dictionary compression that can be bulk-updated efficiently. Both techniques are not designed for OLTP-style updates, but rather for updates in data warehouses. Oracle 11g has on OLTP Compression feature. This feature seems to be applicable only in pure OLTP workloads without analytical queries. Approaches that maintain two separate data stores, an uncompressed "delta" store for freshly inserted data and a compressed "main"-store for older data, require costly merge phases that periodically insert new data into the main store in a bulk operation.

HyPer data representation combines horizontal partitioning and columnar storage: A relation is represented as a hierarchy of partitions, chunks and vectors (see figure 4). Hot/cold clustering aims at partitioning the data into frequently accessed data items and those that are accessed rarely (or not at all). This allows for physical optimizations depending on the access characteristics of data (see figure 5). HyPer measures the "temperature" of data on virtual memory page granularity. Since HyPer stores attributes column-wise, this allows HyPer to maintain a separate temperature value for each attribute of a chunk, i.e. for each vector. Both read and write accesses to the vectors are monitored by the Access Observer component using a lightweight, hardware-assisted approach. It distinguishes four states a vector can have: hot, cooling, cold or frozen.

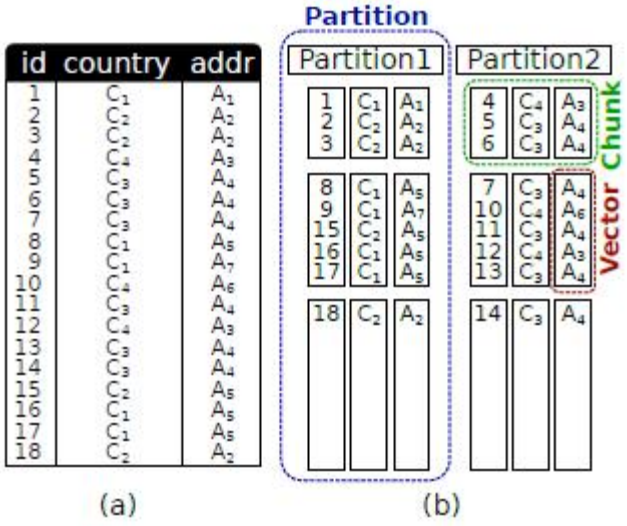

Fig. 4. (a) Example relation. (b) Physical representation of example relation (without compression).

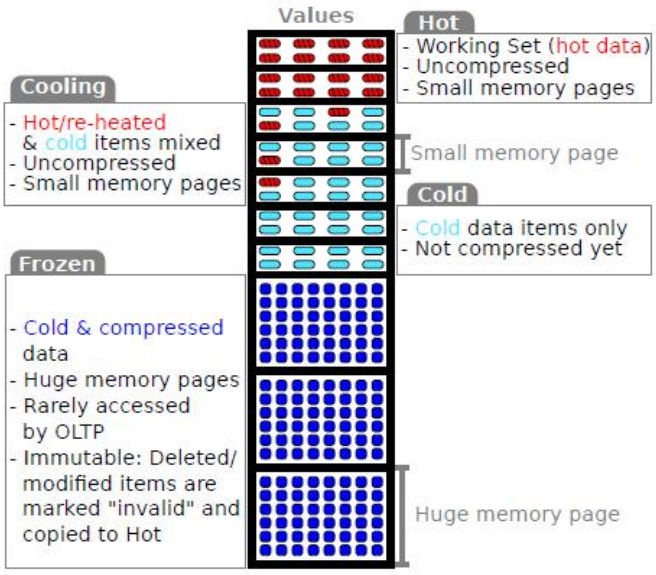

Fig. 5. Hot/cold clustering for compaction.

Cold chunks of the data can be "frozen", i.e. converted into a compact, OLAP-friendly representation as they are likely to be almost exclusively accessed by analytical queries in the future. They are compressed, stored on huge pages (2MB per page on x86) for frozen data has multiple advantages over the use of regular pages (4kB on x86). First:scanning huge pages is faster than scanning regular pages. Second: the translation lookaside buffer (TLB) has separate sections for huge and normal pages on most platforms and so the two separate workloads do not compete for the TLB. And third: huge pages speed up snapshotting via faster copying of the process's page.

HyPer do not propose new compression algorithms, but use well-known algorithms. HyPer's main goal is to impact transaction processing as little as possible and in the same time speed-up query execution. The small part of data is frequently accessed – and freshly inserted data in particular – is left uncompressed and thus efficiently accessible for transactions. For cold chunks, HyPer proposes to use dictionary compression and run-length encoding, which was found to be beneficial for column stores.

V. Conclusion

HyPer's In-memory columnar database system development seems to give us very promising results: world record transaction processing throughput and best-of-breed OLAP query response times and better OLTP throughput than that of dedicated OLTP engines. Furthermore there is no need for resource consuming ETL processes. If these goals are met and they can be achieved with moderate enough costs, there will be enterprise systems for larger companies. This kind of technique may well be successfully competing with the today's technology in the near future.

References

[1] T. Cao, M. Salles, B. Sowell, Y. Yue, J. Gehrke, A. Demers, and W. White. Fast Checkpoint Recovery Algorithms for Frequently Consistent Applications. In SIGMOD, 2011.

[2] Florian Funke, Alfons Kemper, Thomas Neumann: Compacting Transactional Data in Hybrid OLTP & OLAP Databases. PVLDB 5(11):1424-1435(2012).

[3] S. H´eman, M. Zukowski, N. J. Nes, L. Sidirourgos,and P. A. Boncz. Positional Update Handling in Column Stores. In SIGMOD, pages 543–554, 2010.

[4] Alfons Kemper, Thomas Neumann, Florian Funke, Viktor Leis, Henrik Mühe: HyPer:Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. IEEE Data Eng. Bull. (DEBU)35(1):46-51(2012).

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

[6] Hasso Plattner: A common database approach for OLTP and OLAP using an in-memory column database. SIGMOD 2009:1-2.