Data Structures for Mixed Workloads in In-Memory Databases

Jens Krueger, Martin Grund, Martin Boissier, Alexander Zeier, Hasso Plattner
Hasso Plattner Institute for IT Systems Engineering
University of Potsdam
Potsdam, Germany
{*firstname.lastname*}*@hpi.uni-potsdam.de*

***Abstract*—Traditionally, enterprise data management is divided into separate systems. Online Transaction Processing (OLTP) systems are focused on the day to day business by being optimized for retrieving and modifying complete entities. Online Analytical Processing (OLAP) systems initiate queries on specific attributes as these applications are optimized to support decision making based on the information gathered from many instances. In parallel both hardware and database applications are subject to steady improvements. For example, today's size of main memory in combination with the column oriented organization of data offer completely new possibilities such as real time analytical ad hoc queries on transactional data. Especially latest development in the area of main memory database systems raises the question whether those databases are capable of handling both OLAP and OLTP workloads in one system. This Paper discusses requirements for main memory database systems managing both workloads and analyses using appropriate data structures.**

I. Introduction

Today's enterprise systems are segmented into the so-called Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). With growing data volumes of enterprise applications, relational database management systems were not capable of providing sufficient performance for analytical questions anymore. The more data the OLTP system has to handle, the more analytical applications as Business Intelligence solutions were moved into dedicated enterprise data warehouses and data marts.

This separation into different components continued due to the increasing demand for data timeliness. Where traditional ETL (Extract, Transform and Load) processes ran in batch mode once a day just a few years ago, data older than 30 minutes is not acceptable for most modern companies today. This trend towards right and real time analytics throve the rise of new market segments as software systems for closed-loop techniques, Business Action Management and Business Performance Management [6], [9]. This resulted in new components as Operation Data Stores or Data Staging Integrators while complicating the composition of business systems at the same time.

That increasing obliteration has several reasons. Beside traditional applications as Financials, Customer Relationship Management or Supply Chain Management new data sources had to be integrated into existing data warehouse environments as online stores, or marketing campaigns. Due to the heterogeneity of these sources many companies implemented data staging and integration systems like Operational Data Stores as intermediate steps between transactional systems and analytical components as enterprise data warehouses. These intermediate systems consolidate data from different sources and usually act as data cleansing and integration applications, too. New technologies as closed-loop analytics point to the major problem of this trend: to handle the increasing demand for data timeliness and provide new functions into current systems, new systems have to be integrated since existing systems do usually not provide the performance and scalability required. Hence, the components of enterprise environments are actually increasingly linked as processes getting more sophisticated but they are not growing together and thus increasingly complicate data flowing.

This paper applies recent database research to the field of main memory database systems for enterprise applications. The focus hereby lies on mixed workloads integrated in one common single system as described in [14]. It is evaluated whether main memory databases can act as the primary persistence for enterprise applications including both transactional and analytical workloads. Using a common database model for enterprise systems can significantly improve maintainability and reduce complexity, while providing real time data access and completely new possibilities. Therefore main memory optimized data structures are examined and evaluated in the context of enterprise systems using real customer data.

The remainder of the paper is structured as follows: Section II explains the motivation for research on new data structures and what might be accomplished with the latest hardware. Section III introduces main memory databases in general, shows the research done in this field and discusses main memory optimized data structures. On which findings on enterprise customer data we base our research is shown in Section IV. The Section V includes findings and calculations based on real customer statistics and evaluates the relevance of these findings mean for main memory enterprise databases while Section VI presents related research. The paper closes with a brief conclusion in Section VII.

II. Motivation

A major trend in analytical enterprise systems, especially Business Intelligence solutions, is right and real time computing. For years, vendors have been working on solutions to integrate transactional data into analytical systems as enterprise data warehouses (EDW) and data marts. A major problem of the real time trend were iteratively developed solutions tackling one special problem, often with a lack of scalability.

An example are Operational Data Stores (ODS). Especially in the beginning of the last decade the demand for real time analytics on latest transactional data was increasing. Since most EDWs are not capable of handling the steady stream of transactional data, ODS were integrated in between OLTP and OLAP systems holding only current-valued data. However, new developments in the hardware and database sector encouraged the integration of real time data into EDWs. Due to the fact of growing requirements for timeliness an intermediate step as ODS was providing unacceptable additional delay without providing any new advantages. Furthermore, ODSs usually integrate data only in one-way fashion leaving out new technologies such as closed-loop analytics, which require bidirectional data integration back into the operational systems.

Another major advantage of a database running mixed workloads is the inherent "real time data access". Defining real time in the context of EDWs and Business Intelligence is quite vague and differs from vendor to vendor. In [1] real time is defined as follows.

> The answer is absolutely the most up-to-date information physically possible in terms of both update and access.

Current commercially available systems with redundant transactional data and data integration systems can obviously not fulfill this definition. The techniques discussed in this paper focus on "real real time", thus ensured transactional safety of always having the exact results of the current state of the data when the query was executed. In the following "real real time" is only named real time, even though this might not be congruent with the term real time used by commercial solutions.

To overcome these aforementioned drawbacks this paper discusses new database techniques, contemplating whether it is possible to use a common database model for analytical and transactional workloads enabling a less sophisticated environment while providing real time data access and enabling new functionalities and business processes.

A. Hardware Development

Hardware development of the last years still underlies the trend of an increasing gap between CPU speed and memory speed. CPU performance has improved by 60% each year for over two decades while memory access latency only decreased by less than 10% per year [16]. This development is transferring the bottleneck from hard disk - which was the prime limiting factor when most RDBMS were developmed in the last century - to main memory on modern computer systems [4], [8]. This is even greater if pure main memory data processing is considered. Consequently, new algorithms and data structures tackling the memory gap are needed in order to efficiently use todays hardware.

Commercially available server blades with up to 64 cores and 1 terra byte main memory with steadily falling prices enabled enormous enhancements, especially in the area of main memory column oriented databases, which are currently primarily used for Business Intelligence purposes. Besides performance improvements the growing main memory sizes combined with light-weight compression techniques allow to store complete transactional enterprise data in main memory while providing sufficient query performance in parallel [17].

B. A Common Database Model for Transactional and Analytical Workloads

A very recent trend in the area of analytical databases is optimized write performance. Latest analytical databases need to be able to handle incoming transactional data streams in right or even real time. Looking at those tough time constraints and achievements made in Business Intelligence environments in recent years, this raises the question what these analytical solutions using current hardware development could accomplish. In [17] Plattner proposes the idea of a common database approach using a column oriented main memory database. Hereby all transactional business data is kept in main memory and using the column oriented design all analytical questions are directly answered on the transactional data.

Having only one single source of truth and inherently real time access to this data - even when asking analytical questions - reveals completely new possibilities. Techniques as closed-loop analytics, operational reporting or event-driven business processes are supported out of the box. Furthermore upcoming trends as event-driven process optimization become possible.

III. In-Memory Databases

In-memory (or main memory) database systems are databases which answer all queries directly from main memory instead of reading data from hard disks as traditional RDBMS do. In-memory databases have been researched for over two decades now [3]. Despite the common assumption they are primarily afflicted by CPU bottlenecks as Manegold et al. showed in [15] that the CPU processing time can be directly mapped to the number of cache misses. Hence the new bottleneck was still mainly I/O caused while it moved from hard disk to main memory access [2]. Consequently, existing I/O optimized algorithms have to be modified in order to

acknowledge the memory hierarchies without considering the disk access. The so-called "myth of random access" says that even though access times are constant on main memory, the contiguousness and data locality of read data is a significant performance factor making cache consciousness a prime key performance factor in main memory systems.

Cache memories are subdivided into fixed-sized logical cache lines, which are atomic units for accessing data. Hence, caches are filled on per-cache-line basis rather than on per byte basis. In order to load a value from main memory it is necessary to iterate over all intermediate caches subsequently. Consequently, with stepping down in the memory hierarchy the CPU cycles needed to access a certain value increases. For instances, accessing the main memory consumes 80 times the amount of CPU cycles than an access to the CPU near L1 cache consumes. This factor highlights the necessity of data locality in pure main memory persistent data management. These so called "cache misses" are the goals of optimization in in-memory based databases. As in disk-based systems the proper data management has a huge impact on the processing performance regardless of the in-memory processing speeds up application alone by leaving out the disk access. Furthermore, the cache line based pattern has the effect of reading too much data, if data is accessed in an non-optimal way. For example, especially in analytical workloads, which is largely attribute-focused rather than entity-focused, the column-wise data representation is useful in a sense that only a small number of attributes in a table might be of interest for a particular query. This allows for a model where only the required columns have to be read while the rest of the table can be ignored. Due to this fact, the read cache line only contains the data needed to process the request exhibiting a more cache-friendly I/O pattern. In case of a row-oriented data storage the cache line would be polluted with other attributes of the tuple depending on both the width of the tuple and the size of the cache line. Attributes of a certain tuple can even be on separate cache lines leading to another cache miss while accessing two attributes adjoining each other. Also, this affect can be applied when comparing disk-based row- and column-stores but on page level. Another advantage of sequential memory access is achieved by pre-fetching strategies of modern CPU's. This CPU-inherent technique allows the pre-fetching of another cache line when reading and processing a cache line, i.e. it speeds up the access to the proceeding cache line.

A. Main Memory Data Structures

As mentioned before, database systems organize tables either in a row- or column-wise fashion. Considering a vector-based storage in-memory system the row oriented variant stores each attribute of a tuple contiguously while the column oriented representation stores the values of each column contiguously in memory, which can be seen as

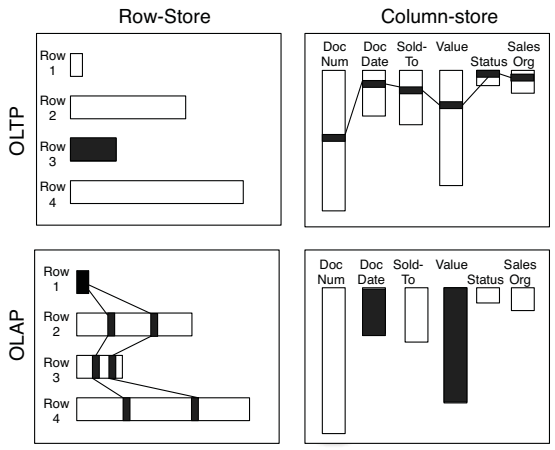

Figure 1. Row vs. column storage with different access patterns

a full vertical partitioning. Therefore the data locality of both designs differs completely with regards to the applied workload. For example, column-stores can efficiently process analytical queries, which usually compute many rows on a small number of attributes. In contrast, row-stores provide better data locality for complete tuple accesses and modifications by low selectivity queries. Figure 1 illustrates the two access patterns OLTP and OLAP with regards to the used storage technique while the black box illustrates the access data for fulfilling the request. As depicted using the appropriate storage organization for a specific workloads leads to an optimal read pattern. For example pure OLTP-style queries perform best in the row-store as the requested row 3 can be access sequentially while in the column-store the reconstruction of the tuple introduces as many cache misses as attributes on the relation. Vice versa in the OLAP use case that benefits from a column-wise data representation.

IV. Real Customer Data Findings

To evaluate main memory data structures not only based on theoretical assumptions, all calculations and predictions in this paper are founded on statistics gathered on a real enterprise customer system. Therefore five years of Financial (FI) and Sales & Distribution (S&D) data have been examined. The data was gathered and analyzed via SQL logs and direct database access.

A. Workload Characteristics

Analyzing the database workload of the given transactional customer system the focus lied on the analytical loads produced in the transactional system. This includes OLAP like queries, usually e.g. aggregations over many table rows.

Besides the actual rate of data modifications was measured. Against common assumptions `SELECT` is the dominating command even on transactional systems as shown in

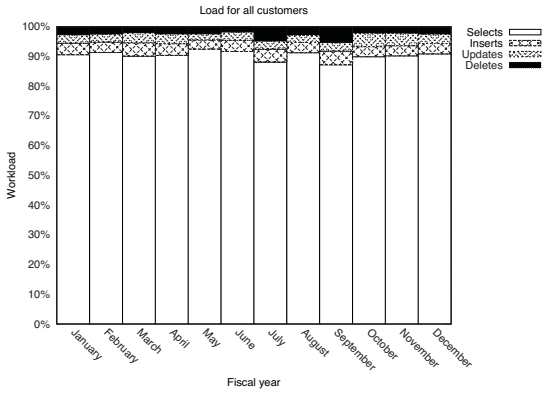

Figure 2. Distribution of SQL Commands on a Transactional System

Table I
DISTRIBUTION OF SELECTION QUERIES

Executions	*Total*	*Distinct Queries*
26986346725	79269	Total selections
949005688	3810	Selections with aggregations
26037341037	75459	Selections without aggregation
3,52%	4,81%	Percentage of aggregations

figure 2. Only a tenth of all queries is actually modifying data. This is especially important when considering the usage of column-stores, since these usually provide a far worse write performance compared to row-stores.

Even though column-stores can profit from the fact already mentioned, that transactional workloads are read dominated, they provide a minor read performance on small result sets or very low selectivities. Table I shows the distribution of selecting queries. Only less than 5 per cent are aggregating select queries. However, this is based on the fact of the inability of todays row-based data management to efficiently handle aggregation on-the-fly. If tables, which store materialized aggregates would be removed due to the on-the-fly aggregation capability of the new storage the workload would show much more aggregating queries.

B. Enterprise Specific Attribute Selection

The analyzed transactional tables have an average width of 200 attributes, which is due to the fact of complex enterprise application that cannot be neglected. Hence, todays data management systems have to handle this amount of columns efficiently even in mixed workload environments with attribute focused queries leaving out most of the columns.

To evaluate main memory data structures it is important, especially for column-stores, to know the exact usage of attributes. Since the row wise data locality of columnar storages is worse than on row-stores, each selected attribute usually produces a cache miss. Therefore narrow selections and modifications of rows are preferable for column-stores. To measure column-stores and typical enterprise selections

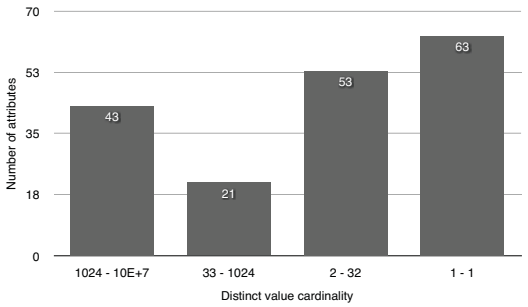

Figure 3. Transactional Table Grouped by Cardinality of Distinct Values per Column

the customer's SQL logs were analyzed. We found that an average `SELECT` uses approximately 20% of all fields. Thus of a 200 attributes wide table 40 attributes in average are requested with a standard deviation of 35, which is actually not very surprising seeing the number of columns having a distinct value cardinality of one as shown in figure 3.

Furthermore the contiguousness of selected attributes has been analyzed. While column-stores suffer from wide selections, the impact of a high attribute selectivity using row-stores in contrast is barely measurable, since often complete rows are read into the cache anyway. For the calculations made in V-A a static value of 20% contiguous attributes selected in each query was used (see Average Selection Width in V-A-1).

Another important aspect concerning main memory data structures are properties effecting compression. Figure 3 shows the cardinality of distinct values per column. Especially column oriented storages profit heavily from columns holding a small number of distinct values since even most trivial compression approaches provide compression rates up to 99% while still providing good query performance. Furthermore, this analysis on characteristics of data leads to the conclusion that only columns with a high cardinality of distinct values are of interest for analytical queries since only these can provide insights.

V. EVALUATION

This Section puts main memory data structures into the context of enterprise systems. First cache miss calculations are shown where hereby the focus is not simply lying on cache miss predictions, but on cache miss comparison between row and column-stores measured in the base of real customer data and real workloads. Assuming a complete main memory database solution might not be realizable right now it is defined which enterprise solutions profit more or less from main memory databases. This Section closes with a brief outlook to current hybrid data structures research.

A. Cache Miss Measurements

Based on the finding explained in Section IV-B the cache misses for typical enterprise workloads have been measured. To find the breakeven point, where a column-store outperforms a row-store depending on the size of the read rows, a set of select queries is calculated based on data explained in Section IV. Hereby a simplified cache miss model was used, which models only one level of cache.

Since an average transactional table has 200 attributes with an average size of 10 byte, we expect a table row to have a size of 2 kilobyte each. Based on analyses on SQL traces (see Section IV-B) the Average Selection Width ASW, which is the average number of selected attributes, is calculated with a sample set of the following selection widths.

$$ASW = (5, 10, 10, 40, 40, 40, 40, 40, 80, 200) \quad (1)$$

The number of resulting rows of each select query is r. The probability of a cache miss per attribute per row is M_o with o as the type of storage form. Based on the cache size and the data statistics shown in Section IV-A we assume M_{row} to be 0.001 (not exactly zero, since row might be spanning over two cache lines). Even though attribute access in a column-store almost translates to one cache miss per attribute we set M_{column} to be 0.8 since Section IV-A revealed that many columns only consist of very few distinct values and thus can be very well compressed resulting in possibly multiple columns per cache line, even on huge data sets. Furthermore we assume no read row to be contiguous to any other read line, simply because of the huge data sets enterprise systems store. Hence the probability R_o of producing a cache miss getting values of a column is 1 for row-stores and 0.15 for column-stores (numbers derived from ASW and calculated B-tree size of data shown in figure 3).

$$CMiss_o(r) = \left\lceil \frac{\sum_{k=1}^{\#ASW} ASW_k \times M_o + R_o \times r}{\#ASW} \right\rceil \quad (2)$$

The results of formula 2 can be seen in figure 4. Column-stores outperform row-stores beginning at an average result set of approximately 350 rows. Even though it was shown that aggregating queries are not dominating in transactional workloads (see table I), this paper assumes that enabling analytical queries on transactional data at any time and on any data will significantly increase the analytical workload on transactional systems. Since only a minority of queries reads more than 350 rows one has to decide which tradeoffs to accept choosing a storage design. Depending on the write performance of column-stores this decision usually tends towards column-stores since they offer analytical functions and are steadily improving their write performance. Row

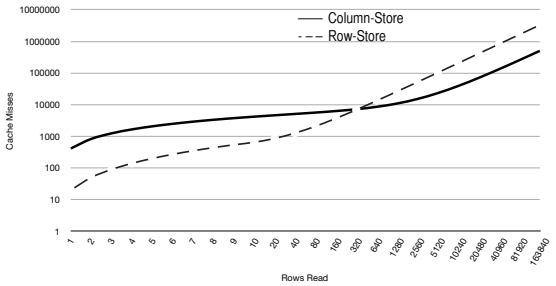

Figure 4. Chart measuring cache misses based on the average ESAS and the row selectivity

stored tables are most suitable for tables with a heavy transactional workload and tables which are usually never read using aggregations or using primary keys. A typical example here are tables storing customer data as address, contact person etc. or configuration tables.

Concluding one can say that given a known workload and a exact knowledge about ones data it is possible to find metrics defining whether a column oriented or row oriented design is more suitable for a certain table.

VI. Related Work

This work focuses handling mixed workloads by evaluating row- and column-wise main memory data structures. Comparable publications to this storage technique are Fractured Mirrors [18] and HyPer [13]. Fractured Mirrors is optimized for hard disk efficiency but describes a hybrid model using the Decomposition Storage Model (DSM) and the N-ary Storage Model (NSM). Fractured Mirrors writes data both to column and row oriented schemas in parallel, while answering selects using the most appropriate table form. HyPer uses a virtual memory snapshot mechanism for fast data modification and duplication. Both implementations improve read queries since they can process these with regards to certain properties by leveraging the advantages of one or the other storage concept. Similar approaches that serve both workloads via redundant data storages suffer in general from additional costs and complexity on writes since some sort of two-phase commit protocol has to be in place if storage of both needs to be consistent. While basically tending in the same direction by providing different data structures for different purposes a major aim of this work is the strict avoidance of any redundancy but profit inherently from advantages such as easier transaction handling, less complexity and better maintainability.

Besides intense research on column-stores there has been comparably little research work on row oriented databases in main memory. The most prominent works are H-Store and P*Time [7], [12], which are both optimized for OLTP workloads. Besides these transactional main memory row-stores mainly performance enhancing cache layers have been

researched as in [19] and [5] where the hard disk still serves as the primary persistence.

As shown in the previous Sections it is possible to define which storage paradigm is more efficient based on measurable metrics. Following this rationale grouping of frequently accessed columns should provide the optimal solution for a specific workload and has been researched over the last decade with different approaches. Besides [18] with having column oriented as well as row oriented tables in parallel other approaches use a more loose table paradigm and combine column and row orientation. Two variants here are e.g. Data Morphing [11] and HYRISE [10].

VII. Conclusion

This work studied data structures for main memory databases and illustrated their performance implications based on real customer data while considering data and workload characteristics derived from realistic enterprise applications.

Taking the hardware development into account especially columnar storages are very promising, since their drawback - the write performance - will lose its impact on the total performance with the current development for increasing real time data integration. Besides, the write performance in an in-memory based column-store is limited to the log performance to disk as in conventional disk-based database systems, since decomposing relations in main memory is much faster than logging to disk even while introducing a cache miss on every column access.

We could show that in mixed workload environments as applied in modern enterprise applications the in-memory column-oriented storage organization is advantageous. To what extend tables which have both a high transactional as well as an analytical load can be remodeled or partitioned in a way, which does not create new redundancy but provides a automated workload adaption, has to be evaluated in further research. For the future also analytical workloads on transactional data have to be analyzed to get a more precise workload definition of mixed workload environments. Furthermore latest hybrid developments are very promising to utilize the memory bandwidth.

References

[1] L. Agosta and K. Gile. Real-time data warehousing: The hype and the reality. Technical report, Forrester Research, Inc., December 2004.

[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a Modern Processor: Where Does Time Go? In *VLDB*, 1999.

[3] P. M. G. Apers, C. A. V. D. Berg, J. Flokstra, P. W. P. J. Grefen, M. L. Kersten, and A. N. Wilschut. Prisma/db: A parallel main memory relational DBMS. *IEEE Transactions on Knowledge and Data Engineering*, 4:541–554, 1992.

[4] P. A. Boncz, S. Manegold, and M. L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In *VLDB*, 1999.

[5] C. Bornhoevd, M. Altinel, S. Krishnamurthy, C. Mohan, H. Pirahesh, and B. Reinwald. DBCache: Middle-tier Database Caching for Highly Scalable e-Business Architectures. In *SIGMOD Conference*, 2003.

[6] F. Buytendijk and D. Flint. How bam can turn a business into a real-time enterprise. Technical report, Gartner Research, 2002.

[7] S. K. Cha and C. Song. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. In *VLDB*, 2004.

[8] J. M. Cheng, C. R. Looseley, A. Shibamiya, and P. S. Worthington. IBM database 2 performance: Design, implementation, and tuning. *IBM Systems Journal*, 23(2):189–210, 1984.

[9] M. Golfarelli, S. Rizzi, and I. Cella. Beyond data warehousing: what's next in business intelligence? In *DOLAP*, 2004.

[10] M. Grund, J. Krueger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. HYRISE - A Main Memory Hybrid Storage Engine. In *to appear*.

[11] R. A. Hankins and J. M. Patel. Data Morphing: An Adaptive, Cache-Conscious Storage Technique. In *VLDB*, 2003.

[12] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. B. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: a High-Performance, Distributed Main Memory Transaction Processing System. *PVLDB*, 1(2):1496–1499, 2008.

[13] A. Kemper and T. Neumann. HyPer - Hybrid OLTP and OLAP High Performance Database System. Technical Report May, Technische Universitaet Muenchen, 2010.

[14] J. Krueger, C. Tinnefeld, M. Grund, A. Zeier, and H. Plattner. A case for online mixed workload processing. In *Third International Workshop on Testing Database Systems*, 2010.

[15] S. Manegold, P. A. Boncz, and M. L. Kersten. Generic database cost models for hierarchical memory systems. In *VLDB*. Morgan Kaufmann, 2002.

[16] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, I. Thomas, and K. Yelick. A case for intelligent ram: Iram. *IEEE Micro*, 17, 1997.

[17] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD Conference*, 2009.

[18] R. Ramamurthy, D. J. DeWitt, and Q. Su. A case for fractured mirrors. In *VLDB*, 2002.

[19] T.-T. Team. Mid-tier caching: the TimesTen approach. In *SIGMOD Conference*, 2002.