Main Memory Databases for Enterprise Applications

Jens Krueger, Florian Huebner, Johannes Wust, Martin Boissier, Alexander Zeier, Hasso Plattner
Hasso Plattner Institute for IT Systems Engineering
University of Potsdam
Potsdam, Germany
{jens.krueger, florian.huebner, johannes.wust, martin.boissier, alexander.zeier, hasso.plattner}@hpi.uni-potsdam.de

***Abstract*—Enterprise applications are traditionally divided in transactional and analytical processing. This separation was essential as growing data volume and more complex requests were no longer performing feasibly on conventional relational databases.**

While many research activities in recent years focussed on the optimization of such separation – particularly in the last decade – databases as well as hardware continued to develop. On the one hand there are data management systems that organize data column-oriented and thereby ideally fulfill the requirement profile of analytical requests. On the other hand significantly more main memory is available to applications that allow to store the complete compressed database of an enterprise in combination with the equally significantly enhanced performance. Both developments enable processing of complex analytical requests in a fraction of a second and thus facilitate complete new business processes and –applications. Obviously the question arises whether the artificially introduced separation between OLTP and OLAP can be revoked and all requests be handled on a combined data set.

This paper focuses on the characteristics of data processing in enterprise applications and demonstrates how selected technologies can optimize data processing. A further trend is the use of cloud computing and with it the outsourcing of the data centre to optimize cost efficiency. Here column-oriented in-memory databases are helpful as well as they permit a greater throughput, which in turn enables more effective use of the hardware and thus saves costs.

I. Introduction

Data management in enterprise application has hardly changed in the last decades. Relational databases are in use whose architecture is based on characteristics of transactional data processing defined 20 years ago. In the meantime however demands on enterprise applications have changed. In one respect shorter processing times in complex applications are necessary, but it is also vital to use the most current data as decisive support for analytical requests. Further the automated processes in enterprise application result in larger data volume which must be processed efficiently – transactional as well as analytically.

In addition enterprise applications have become more complex over time to meet more sophisticated demands and to counter balance properties lacking in data management systems. Examples include the redundant storage of aggregates, the materialization of pre-computed result sets in dedicated tables or the outsourcing of processes in specialized systems or asynchronous programs.

While most of these variants see the solution in the redundant storage of data to achieve adequate response times for specific requests, the flexibility of the system is compromised through the necessity of a predefined materialization strategies. Furthermore this greater complexity increases total costs of the system.

The trend of Cloud Computing of recent years offers additionally potential for saving costs. Both - providers as well as customers benefit as operational savings are passed on to service users. As well the inherent flexibility of the use of the cloud concept, including its customized payment model for clients to optimize their costing as only the really needed computer capacity is employed. In relation to data management demands arise that want to be considered in the context of enterprise applications and their architecture.

This paper will address these requirements by examining the data processing technologies and how these requirements and data characteristics optimize supporting modern enterprise applications. Section II is devoted to the latest trends in the field of data management of enterprise applications. Section III analyses current requirements of enterprise applications and how these can be addressed with column-oriented main memory databases. Section IV demonstrates the potential advantages and innovations possible through the employment of column-oriented in-memory databases. Work done so far in this field is introduced in section V. The article concludes with a summary and an outlook in section VI.

II. Trends in Data Processing

This section will focus on different developments in the area of data processing, that looked at individually are nothing new – for example already at the end of the 70s the vertical partitioning (column-orientation) was examined – however the merging has resulted in new deployment areas whose requirement are directly derived from current enterprise applications. This follows another trend of recent years that views the specialized data base as advantageous to the general purpose Database [1] and thus motivates a data management specialized towards enterprise applications, where the processing of data in main memory alone plays a central role.

On one hand the development of hardware in recent years increasing the size of main memory has led to in-memory databases becoming possible and so moving it into the focus

978-1-61284-449-7/11/$26.00 ©2011 IEEE

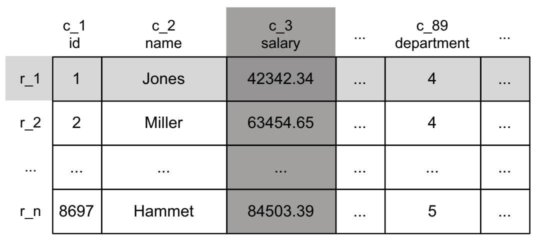
of research. On the other hand in-memory databases that store data column-oriented make it possible to merge already existing transactional and analytical databases again, as this enables a substantially faster data access and an analysis based on the transactional data available [2].

This is made possible through the combination of different technologies and the consideration that today's enterprise technologies primarily use read access. Besides data structures optimized for main memory access for example also compression methods and Insert-Only strategies are employed and which will be discussed in the following section. Another current trend is enterprises move to no longer operating their own computer centers but use the services of so-called Cloud Computing. So finally we will focus on Cloud Computing and how column –oriented main memory databases fit in there.

A. In-Memory Databases

As mentioned above today servers with large main memory capacities are at our disposal that may reach hard disk size. Servers with a size up to 2TB of main memory are available as standard, so that database management systems (DBMS) that respond to all requests directly from main memory are possible without using hard disks or others as a primary persistent source, called In-Memory Databases(IMDB) or Main-Memory-Databases (MMDB) [3], [4]. Today it is basically possible for conventional database systems to become interim data storage facilities, where a buffer pool manages administration. This as well as the characteristics of the data structures optimized for hard disk access, as for example B+ trees and pages create on one hand a greater admin burden and on the other hand a less than ideal filing storage in comparison to a database that is completely main storage based. Nevertheless the question arises to what extend complete databases can be maintained in an only main memory based storage. But although data volume in enterprise applications is steadily increasing, the growth of data in general is by no means as fast as is unstructured data, as for example in social networks. Transactional processing in enterprises is based on actual events connected to the real world as is the number of customers, products and other entities in the real world. These events of transactional processing do not grow as rapidly as final main memory capacities. On the basis of organizational and functional partitioning most systems do not exceed the active size of 1TB.

Compared to the increase in main memory capacity the memory access latency and the memory range have improved little. In order to optimize full memory range utilization modern CPUs use multilevel cache architectures. It is possible to circumvent range limitations in memory access or at least minimize it as shown in [5] and [6]. These so-called memory hierarchies use the fact that smaller but significantly faster memory technologies operate closer to the CPU, but lead to that access in main memory depend on it. The fact that data is being read in cache lines with a typical size of 64 bytes is essential. The consequence of this is that memory access happens in blocks and that all algorithms and data structures



	c_1 id	c_2 name	c_3 salary	...	c_89 department	...
r_1	1	Jones	42342.34	...	4	...
r_2	2	Miller	63454.65	...	4	...
...	...	...	...	...	...	...
r_n	8697	Hammet	84503.39	...	5	...

Fig. 1. Data access on employee table: row- and column-wise access.

need to be optimized in view of this fact, to use existing architecture efficiently [7], [8].

In [9] Manegold et al. show that processing time of memory internal data access is determinable by the number of cache misses. In this way CPU cycles and cache misses can be used as an equivalent of time. What follows is that to optimize performance of main memory driven systems primarily optimizing cache utilization as well as reduction of cache misses are of importance. Data is read sequentially to achieve best possible performance, so that the loaded cache lines are utilized as complete as possible and only the data is read that is required.

Figure 1 shows the sample table *employees*, highlighting row- and column-wise data access. Different access patterns and workloads as online transactional processing (OLTP) or online analytical processing (OLAP) are directly affecting the number of cache misses. In OLTP typical single row selections, the number of cache misses increases with the higher selectivity of attributes. This enables row-stores to utilize the cache very effectively when complete rows are accessed. With narrow projections (e.g. "SELECT id, name, department FROM employees WHERE id = 1", 1) on a single row access, row-stores are comparably behaving as column-stores, potentially resulting in a cache miss per requested attribute and a low utilization of cache lines. Columnar stores in contrast are able to use caches effectively when a high number of rows is accessed using a high attribute selectivity, as shown in section II-B.

A potential optimization in regard to cache-use could be for example a table structure that groups attributes that are requested most frequently, if not the entire row is requested, that these fit into a cache-line and prevent cache misses. [7]

In IMDBs the focus is therefore on the optimization of the physical data layout in main memory, which is optimized towards the actual occurring access pattern. These patterns are normally defined by existing OLTP or OLAP workloads. Furthermore it is important to ensure that the available memory range is fully exploited and data, if possible is read sequentially. These demands are analogue to hard drive based database systems as these also benefit from whole blocks being read and utilized. However other optimizations are being used here as parameters are altered as well as the hard drive being superfluous in the access hierarchy as external memory.

B. Column Oriented Databases

In the last decade, due to the growing demand for analytical functions like ad-hoc requests for transactional data, column-oriented databases, particular in the environment of OLAP applications, have became the focus of academia.

Column-oriented databases are founded on a vertical partitioning by columns, which were considered for the first time at the end of the 70s and developed further by Coupeland and Khoshafian 1985 [10]. Each column is stored separately while the logical table schema is retained through the introduction of unique positioning keys. The use of positioning keys leads to further memory consumption however can implicitly be avoided with the aid of positioning information of these attributes in columns. The latter prevents sorting the attributes singularly, but offers a more efficient reading in case of accessing further attributes, as the implicit positioning key is used directly for addressing purposes.

The column-wise organization of data offers in particular advantages to reading fewer attributes, as during requests no unnecessary columns must be read as is the case for example in row-oriented structures on hard drives.

Therefore greater speed is achieved simply through reduction of the data volume to be read for attribute-focussed requests, as they occur in analytical cases [11]. The downside is that vertical partitioning raises costs compared to conventional row oriented structure in the production of complete ratios. In case of a hard drive based database system all files must be accessed so that the sequential reading of a tuple is replaced by random and therefore slow access. The same is true for written access as the complete ratio has to be distributed to the files. Figure 2 illustrates both access patterns in relation to the storage variant.

Especially in the context of transactional or rather combined workloads the above described disadvantages have prevented the use of column oriented schema. Only the combination of these filing patterns and the in-memory database enables such scenarios. In addition assumptions regarding transactional processing have changed over the years. It is of note that in the transactional world most requests are of a reading nature. Even when those consists of approximately 50% access keys a column oriented main memory database can play out its advantages as the configuration of complete ratios through fast random access functionality of the main memory in comparison to the hard drive can offset the disadvantages of vertical decomposition. Particularly when the benefits in the reading sector taken into account in the overall assessment are based on few attributes distributed over many rows.

Here the possibility of sequential reading on dedicated columns comes into its own that is faster by factors than random access, and applies to main memory as well as hard disk systems even after multiple shifting. As later shown in section III, this is useful for the optimal storage of enterprise data. Further advantages of column orientation become obvious through the data characteristics in enterprise applications. An attribute shown in later sections, is low level occupation of tables. This means that many columns have a low cardinality of distinct values. This fact as well as one of the qualities of column oriented data organization that columns are maintained separately, enables attributes to be compressed individually with the benefit that the range of a dedicated column is limited by its contents.

C. Compression in Databases

Moore's law says that computing capacity of processors doubles every 18 months and this has shown to be valid for more than 35 years. However it is not valid for the development of hard disk and main memory access speeds, so that for the majority of requests, despite cache optimized data structures, I/O delay represents the biggest bottleneck. The growing gap between speed increase of CPUs on one hand and storage on the other hand is a known problem and will be described for example in [6] as well as [12]. Lightweight and lossless compression techniques assist to diminish the bottleneck as they allow using the available range more efficiently, where potential additional costs for unpacking are absorbed by improved CPU performance.

While early work as shown in [13] had its focus on improving the I/O performance by reducing the data to be loaded, latter work [14], [15] focuses on the compression effects in regards to read performance, for example through the application of 'late-materialization' strategies. In this instant the focus is on light compressions that allow direct access to single values without having to unpack the entire data.

Compression techniques use the redundancy within the data and knowledge of the particular data domain to be compressed to do so most efficiently. On the basis of different properties of column oriented data structures compressing these is especially efficient [16]. As all data within a column a) belong to the same type of data and b) usually have a similar semantic, this results in a low information contents (entropy), which means there are many instances of little differing values. Especially enterprise applications, concerned with working through or capturing repeating processes and events, never exhaust the value range available to them based on the type of data they are working with. It is very often the case that only few values are being used, because the business for example uses only a few and a limited amount of different materials and products.

During Run Length Encoding (RLE) the repetition of a value is stored via a value couple (value, run-length). For example the sequence ”aaaa” is compressed to “a[4]”. This approach is particularly applicable to sorted columns with little variety in their attribute values. In unsorted columns the efficiency is lowered, so that a sorting appears to be obligatory. In a column oriented scenario an ID becomes necessary for the tuple, leading to the above described increase in expenditure. However is the column unsorted, Bit-Vector-Encoding is suitable. Essentially a frequently occurring attribute value within a column is associated with a bit-string, where the bits reference the position within the column and only those bits fix the occurring attribute value through their positioning. The column

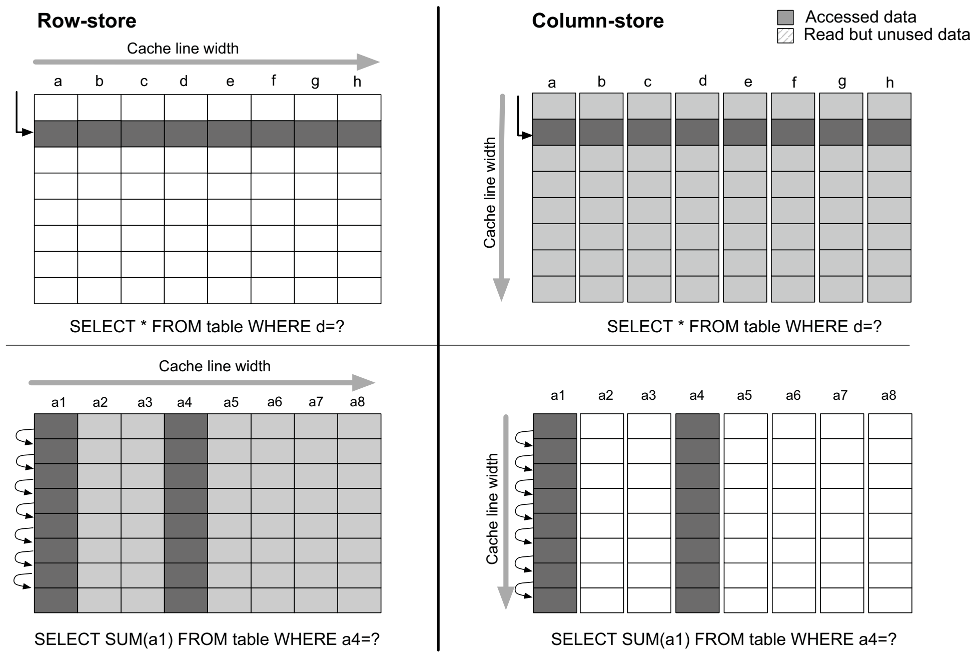


Fig. 2. Data access: row- and column-oriented

is then stored without the attribute value and can be restored with the assistance of the bit-vector. This procedure is ideal when the cardinality of the individual values is low or a single value dominates the load. Another prominent example of a light weight compression technique is dictionary encoding, where frequently occurring patterns are replaced by shorter symbols. In case of a table attribute every distinct value is put down in a dictionary and the value in the column replaced by a key that points to the value in the dictionary. A further optimization is the bit-wise compression of these keys in order to reduce length, so that only the required cardinality can be shown with it in the dictionary, which increases the benefits of this process especially when only few distinct values exist. A sorted dictionary however allows further optimization on the request page, as original sorting can be deducted from the keys without looking them up in the dictionary.

Information density in relation to the used storage space is increased through the compression within the columns. This enables more relevant information to be loaded into the cache for simultaneous processing, which leads to a better use of the frequency range. These techniques are also applicable for row oriented storage schema, but only than produce real advantages when used in combination with column oriented filing and data processing in main memory.

Another important aspect of compression is the disadvantage that modification of data results in further compression of the data stock. To circumvent this effort and nevertheless to allow speedy insert-, update- and delete operations all alterations are collected in a buffer, where data sets are placed uncompressed. The result of all requests to the system is the unification of the compressed main area and the buffer, so that also non-compressed alterations are taken into account. The collected alterations are integrated asynchronously in defined intervals in the already compressed existing data. This process is called merge.

Beyond that performance can be improved as the column oriented request operation – as described in [16] has knowledge of the compression method used, so that CPU – while reading the data – decompresses it in a parallel process. An important aspect during compression is to compare compression ratio and the cost for decompression. Generally the aim is to delay decompression as long as possible, to make the most of the decompressed data and only to decompress those data that is really necessary.

Particularly in the field of enterprise application leveraging compression for analytical queries can create significant advantages. Requests that do not rely on pre-aggregate results, but are computed in real-time can be processed considerably faster, when compression techniques are used where aggregation is possible without decompression.

D. Insert-Only Strategies

Insert-Only data systems define a category of database management systems where new data sets are only inserted. Even update and delete operations are handled as inserts. However that does not mean that viewed logically data sets cannot be deleted or updated. These modifying operations are transferred in a technical insertion, during which a time reference is logged. Inserted data sets are only valid for a period of time and values are therefore time dependent and thus the entire history of data modification is stored. Storage of complete histories of all enterprise data is particularly important in the context of enterprises as the tracing and storing of such histories is legally required in many countries. An example is a client who has a billing complaint that has been forwarded to an outdated address. In such a case it must be possible to reconstruct this. Another example is the alteration of an accounting document. Here the alteration history is vital. Also this kind of data storage enables analysis of historical data as any moment in time can be reconstructed. Today dedicated business warehouse systems are being used that offer this functionality by allocating a time dimension during the loading process to the received data of transactional systems. In addition to the aforementioned benefits does the holding of older versions enable Snapshot Isolation, this means without an explicit locking procedure the database can guarantee that a transaction will operate over the entire run-time of the data that have not been altered by other transactions. This consistency is implemented during read requests by the time reference by which the version of the data can be produced at the beginning of the transaction. This simplifies data management and usually is sufficient for enterprise applications as due to complex locking logic implementing locking on the application level of the data to be processed is necessary anyway.

There are several techniques for the storage of time dependent values. Delta sets are used for the storage of alterations in discreet representations. Here all older versions have to be read to determine the actual valid data set. In order to modify a data entry it is not necessary to read the entire row as all unaltered values can be filled with, for example a pre-modified standard value, like a Not Altered reference. This reduces the data volume to be read required for insertion. Besides not all columns have to be read for this operation. The main disadvantage of this method is based on the fact that all previous versions must be read to generate the actual valid version. This becomes the more costly the more modifications occur over time.

The second option to store time-based values in a row is interval storage. The difference is that for every row a validity interval is stored. When a new row is inserted the actual point of time is fixed as starting point and the end point is left open. In case this row is modified the point of ending is fixed and a new row is inserted with this very time reference as its starting point. Although the old row had to be read, there are advantages during the search as the previously valid row, thanks to the stored interval, is easy to find and not all rows have to read.

A further frequently referred criticism of the Insert-Only technique is the expected increase of storage requirements. To counter this, on one hand update operations in the context of enterprise applications occur significantly less than anticipated and on the other hand the applied compression technique 'Dictionary Encoding' makes sure that unaltered attributes use little additional storage space as the dictionary remains untouched.

E. Cloud Computing

Traditionally the server and their applications of a business are located in private or exclusive computer centers. The availability of broadband internet connections makes it possible to dispense of internal computer centers and to utilize dynamically the computer capacity of a Computing Cloud of an external server. Cloud Computing is of interest to business as no capital expenditure occurs and through the use of scale effect running costs can be minimized.

The cost to customers can also be reduced by taking advantage of the elasticity of the cloud concept. Enterprises pay only for the required computing performance. Is less or more computing output required, the supplier can make this automatically available through an interface. While in classical computer centers hardware has to be dimensioned for a maximum load, using cloud computing enables to employ only the actually required hardware resources which are expanded or minimized depending on the required capacity.

Cloud computing systems are not customer-based (on-premise) but are used and scaled depending on demand (on-demand). The operating risk of the computer center is outsourced from the enterprise to the manager of the cloud. This goes together with the promise that employees from everywhere at any time have access to their data within the cloud, although this can lead to security problems.

Apart from outsourcing to computer centers companies can receive complete applications as a service. This is generally described as 'Software –as-a-Service'. Within the context of software-as-a-service are three different levels on which services are offered. Complete applications can be received as a service. This is called Application-as-a-Service. If a provider only makes a platform available, e.g. a data database as a service, it is called Platform-as-a-service. At the most basic level the provider makes exclusive infrastructure as computers or virtual machines available and we speak of Infrastructure-as-a-Service.

Priorities for database systems running for customers exclusively on-premise are not identical with systems that are offered as Platform-as-a-service or in combination with use of Application-as-a-Service. The latter are offered by a service provider to a wide circle of customers. Therefore they must scale well with many different customers, but must be low cost for the service provider, so that he can offer it profitably to the customers.

In traditional disk based database systems the throughput and speed of the hard drive are the limiting factors. Particularly in database systems which are used as service, it is important that these can achieve a high throughput and a guaranteed high speed. As here the hard disk is the particularly limiting factor, is it possible through the use of main memory data bases to achieve a higher throughput that in turn leads to lower costs for the service provider.

Newer proposals as RAM Cloud [17] therefore promote to operate cloud environments only in main memory based storage. Through the use of main memory architectures the throughput and speed of the hard disk are no longer the bottleneck but the speed of the processor, how fast it can read data from main memory. As these systems are much more effective more customers can be served by one system (multi-tenancy) and existing hardware can be utilized more efficiently, which in turn is more cost effective.

III. Requirements of Business Applications

Due to increasing data volume, new requirements of business processes and growing demands for timely business data, enterprise applications are becoming more complex to compensate for defects within data management infrastructure. This includes for example the redundant storage of transactional and analytical data. The reason that current database systems can't fulfill the requirements of modern enterprises is that these usually are only optimized for one application case: either OLTP- or OLAP- processes. This fact leads to a discrepancy of enterprise applications in relation to the underlying data management, as with conventional databases particularly complex operations are not run within reasonable time. While these problems are largely recognized for analytical applications and therefore usually data warehouses are used, they also apply to transactional applications, which are complex in modern systems. To compensate for this disadvantage of the data management infrastructure, complex operations and longer running processes are outsourced in batch-jobs.

As a result this approach slows down the speed of business processes and external requirements can possibly no longer be fulfilled. Another frequently used approach is the materialization of pre-rendered results. Materialized views of analytical applications in data warehouses are one example of this. However, they lead to a diminished flexibility and hinder maintenance of such OLAP systems. An analogue procedure is applied in OLTP systems where the materialization is administered by the application as application logic has to be considered. The performance problem of redundant data storage by predefined characteristics is met, while at the same time greater complexity and diminished flexibility is accepted. Besides the growing program complexity necessary for consistent safety the lack of flexibility in current applications presents as an obstacle as requirements of the program cannot be implemented.

The actual enterprise applications used represent only a fraction of the database functions available particularly with regard to transactional behavior and complex calculations on data. In summary it can be said that the current approach of today's enterprise applications leads to greater complexity in data modification and storage. The following section will show the characteristics of enterprise applications, how these can be employed to current software technologies (e.g., column oriented in-memory databases) and which improvement and simplifications this enables.

A. Request Distribution

Databases in enterprise management are generally classified through their particular optimization –either for OLTP or OLAP. It has been assumed that the work loads of enterprise applications largely consist of insert-, update-, and delete requests. So the TPC-C benchmark [18] is made up of only 54% of read request and 46% of write requests.

	OLTP	**OLAP**
select	84%	94%
insert	8%	2%
update	5%	3%
delete	3%	1%

TABLE I
DISTRIBUTION OF *select-*, *insert-*, *update-* AND *delete*-OPERATIONS

To determine whether read or write optimized databases are more suitable for such workloads 65 installations of Enterprise Resource Planning Systems (ERP) were examined with regard to their performed database requests. The result of this research can be seen in figure I. This clearly shows that distribution of requests is rather more read focused. A minimum of 80% of all requests are select requests. This fact underpins the application of read optimized databases, potentially with a write optimized buffer as described in [19] and [20].

B. Value Distribution of Enterprise Data

Apart from the distribution of database queries the value distribution and characteristics of data are of importance for column oriented databases. A broad assumption for enterprise application data is that it presents with a high value distribution, i.e., that many different values exist for each attribute. However, studies have shown that values of each column are barely distributed, i.e., that only few different values exist.

To proof this result the frequency of various values per attribute was measured. This analysis was done using the main tables of the finance management as well as sales and distribution data of a variety of customer systems. In the enterprise application examined for example an accounting note of around 100 attributes exist while the corresponding positions consist of 300 attributes each. The table of material movements consists of 180 attributes.

Figure 3 shows the frequency percentage grouped together according to value distribution per column of all examined tables of customer systems analyzed. It obviously shows that most table attributes correspond with the last group with only one value. This value can either be a zero or a default value. What follows is that not all attributes are used in the application which is of great interest for application optimization. It has to be noted that these are average values of all customer systems examined and that their signature depends on the individual business and its respective sector.

As shown in fig. 3(a) only 43 of 180 attributes in the material movement table have a significant number of distinct values. Therefore approx. 75% of all columns have a relatively low value distribution. The attribute with the highest number of distinct values is the attribute 'motion number' which at the same time is the primary key of the table. Furthermore, 'transport request number', 'number', 'date' and 'time' have a high value distribution. It is worth noting that 35% of all columns only have one value.

Consequently very important corporate characteristics can be determined. Depending on the application and the industrial sector in many columns only one value is deposited. These applications benefit significantly from a light weight compression. Additionally, only columns that are of importance for analytical requests show a certain cardinality of different values. This fact speaks for the use of column oriented databases that only have to read the projected attributes.

C. Single-and Mass Data Processing

Although transactional systems work on each instance of objects, analysis shows that the majority of used data is processed jointly. It is generally distinguished between processing individual instances of business entities and the common processing of multiple instances. The former for example include an order or a customer, while determining all payable accounts or the top ten clients on the basis of the turnover require processing many data sets.

In principal, enterprise applications are based on individual business instances, nevertheless it is not possible to establish the state of an enterprise on the basis of individual data sets. E.g., to ascertain the progress of a project, first of all the context must be clarified. Various attributes of various events are read to construct it and usually are aggregated. Examples of such contexts include dashboards or work inventory lists. Dashboards are used to demonstrate the current state of a project or another semantic unit. Work inventory lists are used to create performance tasks from other objects –such as updates and invoices. Furthermore there are business entities whose state is only dependent on current events instead of being determinable by pre-rendered aggregation. Examples include the accounts of financial accounting or the inventory management.

In general this kind of mass data processing involves reading of few sequential attributes instead of individual relations. Regarding the use of column oriented in-memory databases it can be said that these benefit the more, the more operations belong to this category.

D. Application Characteristics

Different enterprise applications have different requirements of data processing. Table II shows the analysis cut by different types of business applications.

The transactional characteristics were drawn in light grey and the analytical characteristics in dark grey to highlight the different requirements of each application. It is immediately apparent that no application can be categorized as purely transactional or analytical. Although for example the processing of orders is inherently transactional, the sales analysis as well as the reminder management with their analytical functions depend on it. The last two mentioned applications work - reading as well as writing- directly on transactional data. The reading operations are thereby mostly complex analytical queries on large amount of data [2].

How transactional and analytical behavior interacts with one another can be determined through the context of the respective operation. Thereby the context is defined as the metric. This is established by the required data which is needed to enable decision making in a business process. When the context of a decision is relatively small, the data volume required is minimal. The larger the context, the larger becomes the data volume to be read.

E. Insights

While the considered applications can be understood as independent software applications, the main task of complex business systems is to unite these applications and to link them to one and the same data source.

Transactional applications typically work on atomic business entities and generally require databases optimized for transactional workloads. However, analytical applications have different requirements of a database. Applications in this area follow clear patterns and usually cover clear business requirements as data cleansing, data consolidation etc. Typical requests of such applications often select all existing data and reduce step-by-step, normally following a navigation path or a drill-down-strategy, i.e., forming aggregations on all data then adding restrictions like selecting a particular product or a specific region.

With the aid of column oriented in-memory databases it is now possible again to combine database systems for transactional and analytical workloads. Column oriented in-memory databases are ideal to form aggregations over columns. Due to the compression process it is far more complex to insert, delete, and update data. It could however be shown that in the described enterprise applications even in transactional applications these operations are only a small part of the total. Due to the fact that most columns present with a slight value distribution, the application of column-based compression is ideally suited to conserve memory and reduce reading time of individual values. Even in transactional workloads, generally several data sets are read and altered at the same time. The advantage of row based database systems in such scenarios is of minor importance in the reconstruction of individual tuples.

IV. Advantages of Column oriented Databases for Enterprise Applications

The following section will show how the above suggested data management in the context of enterprise application can generate advantages.

A. Technical Advantages for Enterprise Applications

The possibility to analyze transactional data also creates new opportunities for business intelligence. So for example there are new options for the implementation of event-driven systems, vastly improved networking of independent enterprise applications, and cost savings in the maintenance of corporate environments. Especially the aspect of networking is essential in enterprise applications as value creation takes place through the integration of individual processes.

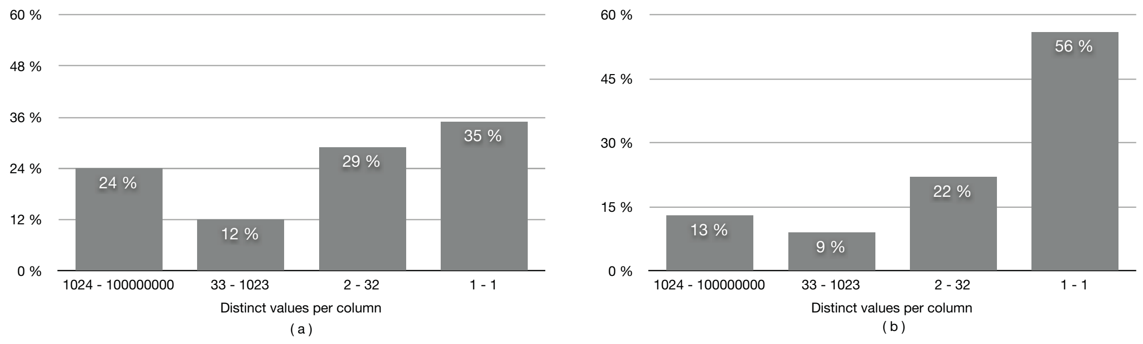


Fig. 3. Grouped value distribution of inventory and finance tables

	Demand Planning	Sales Order Processing	Available to Promise	Dunning	Sales Analysis
Data granularity	Transactional	Transactional	Transactional	Transactional	Aggregated
Data operations	Read & Write	Read & Write	Read & Write	Read & Write	Read-Only
Data preprocessing	No	No	No	No	Yes
Data timeframe	Historical & Recent	Recent Only	Historical & Recent	Historical & Recent	Historical & Recent
Data update cycles	Always Up-to-Date	Always Up-to-Date	Always Up-to-Date	Always Up-to-Date	Cyclic Updates
Data volume per query	Large	Small	Large	Large	Large
Query complexity	High	Standard	High	High	High
Query predictability	Medium	High	Medium	Medium	Low
Query response time	Seconds	Seconds	Seconds	Seconds to Hours	Seconds to Hours

OLTP Characteristics are colored light grey   OLAP Characteristics are colored dark grey

TABLE II
COMPARISON OF IDENTIFIED APPLICATION CHARACTERISTICS

It is now possible to pose analytical questions directly within transactional processes. A whole range of new possibilities arises, e.g., for the real-time detection of credit card fraud or the query of current reminder items of a customer. Currently such functionality is usually implemented with the help of Operational Data Stores or rather real-time optimized Data Warehouses, whereby either only time restricted data can be processed or it comes to considerably delayed response times (Data Warehouses).

Because the entire enterprise application data is now unified in a single source, not only real-time access to all data becomes inherently possible, but the often propagated but in actual systems ultimately unrealizable goal of a ‘single source of truth’ is achieved.

Event-Driven-Enterprise Systems become possible with the association of analytical requests on the finest granular data in real-time. Analytical events (i.e., events that are not just specific to individual business entities, but are only recognizable by strong analytical queries), which previously came from a Data Warehouse on a deferred basis and for which the corresponding data had to exist in the DW anyway, are now recognizable at any point in the source system and are immediately propagated to the respective component. Also, the data analyzed is not limited to predefined data sets within the DW. This enables for example a far more precise Event-Driven Process Optimization which allows a company to react to specific events of running processes in real-time and respond where appropriate.

A further trend in enterprise applications is the increased Cross-Linking of analytical and transactional systems. An example of this is the linking of Supply Chain Management with Customer Relationship Management. This enables for example to ascertain whether or when certain stocks or discounts are available during a customer enquiry. These connections are currently implemented individually which is elaborate and costly due to the lack of standardized data models and the use of systems from different suppliers. Besides the horizontal linking of business applications as cross-linking, new possibilities arise also with vertical linking. Vertical linking connects analytical and transactional systems. Today this connection is often uni-directionally implemented via ODS. It is referred to as Closed-Loop-Function in bi-directional connections. Depending on the requirements of actuality of data this can mean integration on multiple levels (transactional systems as receiver and transmitter, ODS as integrator, DW as transmitter and receiver). Column oriented in-memory databases help to keep the entire business data in one source which means inherently the unified modeling of data as well as the immediately availability of networking between all components.

Last not least the use of high performance systems has cost saving potentials. By way of avoiding redundancies, complex system landscapes with subsystems that hold the same business data in different formats (individual business entities versus pre-aggregated totals) no longer exist. This does not mean just less hardware and thus less maintenance costs, but also less expense for keeping redundant data consistent. This in turn means a faster and easier development and maintenance of systems in general.

B. Advantages from a User's Perspective

An interesting question arises from the technological potentials referred to in the previous chapter about how far processes can benefit from already existing or rather now possible analytical techniques. Now that employees below management level have access to complex analyses, they are enabled to work more independently and take on more responsibilities. Better overviews facilitate managers to reach more accurate decisions more independently than before. Be it proposals or error correction procedures via a sensory perception test during handing-in of forms or far more complex analytical functions. Furthermore, business processes become far easier to control. Sales figures or customer feedback can directly control production or oil rigs produce depending on current market prices or demand etc.

Of course, upper management levels benefit the most. In today's enterprises they more than ever have to take decisions fast and accurately or recognize trends – particularly since the crisis in 2008. Here it is important that the possibility of performance predictions as well as back-links into running operations exist. Current systems do not fulfill these criteria. Systems which allow analysis directly on transactional data enable changes in decision support. Current analytical systems are primarily push systems. In such systems the management poses questions to the people in the IT department with respective competence who only deliver answers after at times a delay of days. The problem here is that the data radius around the originally posed question is relatively small. New and further reaching questions arise out of the answers (so called follow-ups), which cannot be answered by already existing analyses, new time consuming analysis needs to be conducted. However, in in-memory databases it is possible to pose analytical questions directly to transactional systems. There are no intermediate cleansing-, integration-, or transformation processes and just one data source is queried. This indicates a shift from push to pull systems. It enables the management to access all data and information in real-time without risk that these do not represent the current state of their company 100%. Thus a 360° view of the enterprise is possible. This allows direct intervention into time-critical processes which previous reporting made impossible. Particularly in emergency situations this is a crucial advantage as time-critical decisions have to be taken.

Analysis on transactional data combined with possibility of deriving predictions from real-time data permits much more precise decision support. Prognosis and simulations require transactional data of an enterprise including break and irregularities – instead of pre-aggregated events – for most accurate predictions. Current prediction options are limited in their viability. Not only with more precise but especially with more flexible predictions and time-critical interventions into running processes, corporate governance is able to gain more precise insights into their company and control running processes.

C. Column Oriented Databases in SaaS Applications

To achieve high scalability, applications offered as Software-as-a-service often use the same tables for different clients. Nonetheless it is frequently necessary that some customers can adjust the data model according to their needs. For this reason in row oriented databases various techniques are used, as for example virtual tables (pivot table layout [21]), extension tables or reserve columns that can be used later.

Due to the row oriented positioning of data sets in classical database systems, it is also too work-intensive to insert new columns, as in the extreme case, it leads to a reorganization of the entire storage area and makes tables temporarily unavailable as locks prevent read- or write operations during the process. Especially in the software-as-a-service area applications must remain available during upgrades which include alterations of the data model. As far as columns to be inserted in a column oriented database are concerned, not the entire storage area has to be re-organized as new columns are added in a complete new storage area.

If a new column has to be added, only a stub needs to be created and the meta-information of the table has to be updated so that new attributes are accessible. This is a great advantage allowing for interruption free development and integration of new software. Additionally a new column only materializes after an attribute has actually been inserted.

If a table is used by different clients, it is no longer a problem with column oriented databases to add further columns for each customer.

V. Related Works

As already mentioned in section II the main part of related work concerns specialized databases. This research area contains in-memory databases as well as column oriented data structures. The former was treated in [3] while the latter work on in-memory databases focuses on rather specialized applications. The authors of [22] and [23] describe in-memory databases for transactional workloads, however they use row oriented data structures. In contrast Boncz et.al. in [6] deal with main memory processing and binary association tables, which store data complete vertically partioned and optimized for analytical workloads.

In the meantime the idea of column oriented databases was implemented in many academic projects, for example the main memory based MonetDB [6] or the hard disk based C-Store

[24]. Ramamurthy et.al. describe in [1] a database design for mixed workloads in which various physical data structures are used similar as described in [25].

Another important research subject in the area of enterprise applications addresses the possibility of Real-Time Reporting and Operational Reporting. The latter is a typical application in the area of Active Warehousing [26] which primarily assists decision support. But tactical decisions require typically most current information. To provide theses more accurately than in Active Warehouses Operational Data Stores were developed (ODS). In contrast to data warehouses, which usually update in intervals, in ODS updates occur in real-time of the actual transaction. [27] describes ODS as a hybrid with characteristics of transactional systems as well as data warehouses. ODS offers transactional response times to most current data with a decision support function. In contrast to data warehouses only a small data-set is preserved.

It is possible to work directly on business entities with in-memory databases instead of materializing aggregates and views. Grund et.al. [28] have developed a concept which uses Insert-Only to enable the use of reporting on historical data. Materializing techniques as for example in [29] and [30] use redundancy to cover transactional as well as analytical workloads. However any redundancy offers little flexibility as for example redundancy-free strategies implemented with column oriented in-memory databases where at any time all data are accessible instead of subsets or materialized data.

There are different service providers for cloud computing which offer cloud services promising reducing costs and almost unlimited scalability. Kossmann et.al. [31] have examined a number of architectures of OLTP workloads and found out that there are great variations regarding costs and speed. Depending on workload a company should choose a service provider accordingly.

VI. Outlook

This paper has presented a new approach to data management in enterprise application. By means of analysis of realistic systems and particularly of productive systems characteristics of enterprise application could be determined. The most important realization during data analysis was the minimal distribution of data, i.e. the relative low number of distinct attribute values. Interesting is also that most attributes in enterprise systems are hardly used.

These characteristics have benefits for column oriented in-memory databases. These features of this data management enables enterprise applications which are not possible with conventional databases and this with clearly simplified data schema. Newly created possibilities as for example Operational Reporting on transactional data enable a more precise, a more grounded and faster decision support as no intermediately stored transformations for analytical systems have to be performed and instead transactional real-time data can be used at all times.

With help of the discovered characteristics a prototype on the basis of the SAP ERP finance management was developed which runs directly on a column oriented in-memory database. It could be demonstrated that these surpass conventional databases theoretically as well as through a prototypical implementation in the area of financial accounting and inventory management. Further an extension of the dunning procedure was implemented. Here it was shown that through use of a read optimized database not only improvements were achieved in the run-time, but parallel to that it enabled a customer segmentation, to for example treat otherwise reliable customers differently. Therefore it is no longer necessary to use fixed limits and configurations in payment management as this can occur as a rule directly in business logic. This resulted in improved run-times in inventory management and payment procedures, but also opened complete new possibilities within business processes, which in traditional databases would not be able to perform anyway near as comparatively.

Looking ahead there are further interesting and promising areas of applications for column oriented in-memory databases in the field of enterprise. It is planned to apply the insights gained in view of multi tenancy and cloud computing. Cloud-based in-memory databases have advantages in the scalability of multi tenancy systems and thus potential to optimize costs a particular critical point in the cloud computing environment.

References

[1] R. Ramamurthy, D. J. DeWitt, and Q. Su, “A case for fractured mirrors.” *VLDB J.*, vol. 12, no. 2, pp. 89–101, 2003.

[2] J. Krüger, C. Tinnefeld, M. Grund, A. Zeier, and H. Plattner, “A case for online mixed workload processing.” in *DBTest*, S. Babu and G. N. Paulley, Eds. ACM, 2010.

[3] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. Stonebraker, and D. A. Wood, “Implementation techniques for main memory database systems.” in *SIGMOD Conference*, B. Yormark, Ed. ACM Press, 1984, pp. 1–8.

[4] H. Garcia-Molina and K. Salem, “Main memory database systems: An overview.” *IEEE Trans. Knowl. Data Eng.*, vol. 4, no. 6, pp. 509–516, 1992.

[5] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, “Dbmss on a modern processor: Where does time go?” in *VLDB*, M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. Morgan Kaufmann, 1999, pp. 266–277.

[6] P. A. Boncz, S. Manegold, and M. L. Kersten, “Database Architecture Optimized for the New Bottleneck: Memory Access.” in *VLDB*, M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds., 1999.

[7] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden, “Hyrise - a main memory hybrid storage engine.” *PVLDB*.

[8] J. Krueger, M. Grund, M. Boissier, A. Zeier, and H. Plattner, “Data structures for mixed workloads in in-memory databases,” 2010.

[9] S. Manegold, P. A. Boncz, and M. L. Kersten, “Generic database cost models for hierarchical memory systems.” in *VLDB*. Morgan Kaufmann, 2002, pp. 191–202.

[10] G. P. Copeland and S. Khoshafian, “A decomposition storage model.” in *SIGMOD Conference*, S. B. Navathe, Ed. ACM Press, 1985, pp. 268–279.

[11] C. D. French, ““one size fits all” database architectures do not work for dds.” in *SIGMOD Conference*, M. J. Carey and D. A. Schneider, Eds. ACM Press, 1995, pp. 449–450.

[12] N. R. Mahapatra and B. Venkatrao, “The processor-memory bottleneck: problems and solutions,” *Crossroads*, p. 2, 1999.

[13] G. V. Cormack, “Data compression on a database system.” *Commun. ACM*, vol. 28, no. 12, pp. 1336–1342, 1985.

[14] D. J. Abadi, S. Madden, and M. Ferreira, “Integrating compression and execution in column-oriented database systems.” in *SIGMOD Conference*, S. Chaudhuri, V. Hristidis, and N. Polyzotis, Eds. ACM, 2006, pp. 671–682.

[15] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte, “The implementation and performance of compressed databases.” *SIGMOD Record*, vol. 29, no. 3, pp. 55–67, 2000.

[16] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden, “Materialization strategies in a column-oriented dbms.” in *ICDE*. IEEE, 2007, pp. 466–475.

[17] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for ramclouds: scalable high-performance storage entirely in dram,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, 2010.

[18] “TPC-C Benchmark - Standard Specification - Revision 5.11,” http://www.tpc.org/tpcc/spec/tpcc$_{c}urrent.pdf$.

[19] P. A. Boncz, M. Zukowski, and N. Nes, “Monetdb/x100: Hyper-pipelining query execution.” in *CIDR*, 2005, pp. 225–237.

[20] J. Krüger, M. Grund, C. Tinnefeld, H. Plattner, A. Zeier, and F. Faerber, “Optimizing write performance for read optimized databases.” in *DASFAA (2)*, ser. Lecture Notes in Computer Science, H. Kitagawa, Y. Ishikawa, Q. Li, and C. Watanabe, Eds., vol. 5982. Springer, 2010, pp. 291–305.

[21] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, “Multi-tenant databases for software as a service: schema-mapping techniques.” in *SIGMOD Conference*, J. T.-L. Wang, Ed. ACM, 2008, pp. 1195–1206.

[22] S. K. Cha and C. Song, “P*time: Highly scalable oltp dbms for managing update-intensive stream workload,” in *In Proceedings of 30 nd International Conference on Very Large Data Bases (VLDB 2004*, 2004.

[23] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, “The end of an architectural era (it’s time for a complete rewrite).” in *VLDB*, C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, Eds. ACM, 2007, pp. 1150–1160.

[24] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O’Neil, P. E. O’Neil, A. Rasin, N. Tran, and S. B. Zdonik, “C-store: A column-oriented dbms.” in *VLDB*, K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P. Åke Larson, and B. C. Ooi, Eds. ACM, 2005, pp. 553–564.

[25] J. Schaffner, A. Bog, J. Krueger, and A. Zeier, “A hybrid row-column oltp database architecture for operational reporting.” in *BIRTE (Informal Proceedings)*, 2008.

[26] A. Karakasidis, P. Vassiliadis, and E. Pitoura, “Etl queues for active data warehousing.” in *IQIS*, L. Berti-Equille, C. Batini, and D. Srivastava, Eds. ACM, 2005, pp. 28–39.

[27] W. H. Inmon, *Building the Operational Data Store*. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[28] M. Grund, J. Krueger, C. Tinnefeld, and A. Zeier, “Vertical Partition for Insert-Only Scenarios in Enterprise Applications,” in *IE&EM*, 2009.

[29] J. Kiviniemi, A. Wolski, A. Pesonen, and J. Arminen, “Lazy aggregates for real-time olap.” in *DaWaK*, ser. Lecture Notes in Computer Science, M. K. Mohania and A. M. Tjoa, Eds., vol. 1676. Springer, 1999, pp. 165–172.

[30] W. P. Yan and P. Åke Larson, “Eager aggregation and lazy aggregation.” in *VLDB*, U. Dayal, P. M. D. Gray, and S. Nishio, Eds. Morgan Kaufmann, 1995, pp. 345–357.

[31] D. Kossmann, T. Kraska, and S. Loesing, “An evaluation of alternative architectures for transaction processing in the cloud,” in *SIGMOD Conference*, A. K. Elmagarmid and D. Agrawal, Eds. ACM, 2010, pp. 579–590.