xSellerate: Supporting Sales Representatives with Real-Time Information in Customer Dialogs

Johannes Wust[1], Jens Krueger[1], Sebastian Blessing[1]
Cafer Tosun[2], Alexander Zeier[1], Hasso Plattner[1]

[1]Hassso Plattner Institute, 14440 Potsdam
[2]SAP AG, 69190 Walldorf

johannes.wust@hpi.uni-potsdam.de, jens.krueger@hpi.uni-potsdam.de
sebastian.blessing@hpi.uni-potsdam.de, cafer.tosun@sap.com
alexander.zeier@hpi.uni-potsdam.de, hasso.plattner@hpi.uni-potsdam.de

Abstract: The introduction of 64 bit address spaces in commodity operating systems and the constant drop in hardware prices make large capacities of main memory in the order of terabytes possible. Storing the entire ERP data of large companies in main memory becomes technically feasible and economically viable. Especially column-oriented in-memory databases are a promising platform for enterprise applications to run even complex reports in merely seconds. Response-times in the order of seconds mean that we can use enterprise applications in completely new ways, for example, on mobile devices. In this paper, we demonstrate how mobile applications backed by in-memory data management can support mobile workers. We illustrate this with xSellerate, a running prototype of an application that supports sales representatives with real-time product recommendations and availability checks during customer dialogs. This way, organizations can leverage their operational data to support their sales force in the field and thus, achieve a competitive advantage over rival companies.

1 Introduction

With an increased address space in commodity operating systems, as well as dropping prices for memory, servers with large capacities of main memory become available at a competitive price. Combined with increased computing power due to multicore CPUs, this change enables to store and process entire data sets of even the largest companies in main memory and opens the way for in-memory data management in enterprise computing.

Especially column-oriented in-memory databases have the potential to boost the performance of data-intensive enterprise applications by orders of magnitude and reach sub-second response time for most analytical queries. First installations in large companies demonstrated a reduction of particularly long running queries from several hours on traditional row-oriented and disk-based databases down to merely seconds [PZ11]. This increase in performance has the potential to run analytical queries on the latest operational data, while simultaneously processing the transactional load on a single database system,

making analytics in real-time become true. This so-called real-time business promises significant cost reductions and revenue gains for companies: As analyzed in [Eco11], companies that have implemented real-time business techniques and were able to estimate revenue impact reported revenue gains of over 20%, and cost reduction of nearly 20%. Future gains are expected to reach even higher revenue increases of up to 28%.
Beyond the speed-up of existing applications, this step change in database performance based on in-memory data management has the potential to enable applications in completely new ways, such as analyzing enterprise data from mobile devices. Working with these devices is expected to be even more common in the future: According to a study by IDC, 1.19 billion workers worldwide will be using mobile technology in 2013, accounting for 34.9% of the total workforce [IDC11]. Providing these mobile workers with the information they need in their current context of work will allow them to deliver a distinctive experience to their customers. We demonstrate this potential based on a scenario we developed in cooperation with construction tools producer HILTI. Sales representatives at the construction site of customers can query the availability of products in real-time to tell their customers instantly when a product will be available. In addition, our application xSellerate supports the sales representative with product recommendations based on historic sales information. With the help of this information, the sales representative can offer related products more targeted to the customer.
The remainder of this paper is structured as follows: Section 2 gives a brief overview of column-oriented in-memory databases. In Section 3, we motivate the access of enterprise data from mobile devices and demonstrate the benefits with xSellerate, a mobile application in customer dialogs in Section 4. Section 5 closes the paper with some concluding remarks.

2 Column-oriented In-Memory Databases

This section describes the system model of the relational in-memory database on which xSellerate is based on.
An In-Memory Database (IMDB) is a database system where the primary persistence resides entirely in the main memory [GMS92]. The application prototype described in this paper is based on a column-oriented in-memory database, following the system model described in [PZ11]. All columns are stored dictionary-compressed to utilize memory efficiently. While column-orientation typically favors read-mostly analytical workloads, updating and inserting data into dictionary-compressed column structures is challenging. To achieve high read and write performance, a common concept in column-oriented databases is to split the data store in two parts [SAB$^+$05, BZN05]: a read optimized main partition and a write optimized differential store.
Figure 1 illustrates the overall architecture, which is described in more detail in [PZ11] and [Pla11]. The database system is designed to run on a cluster of blades to allow scale-out. An interface service keeps track of client sessions and the distribution layer coordinates the working nodes that hold the actual data in main memory. The tasks of the distribution layer are distributing queries, metadata synchronization, as well as managing

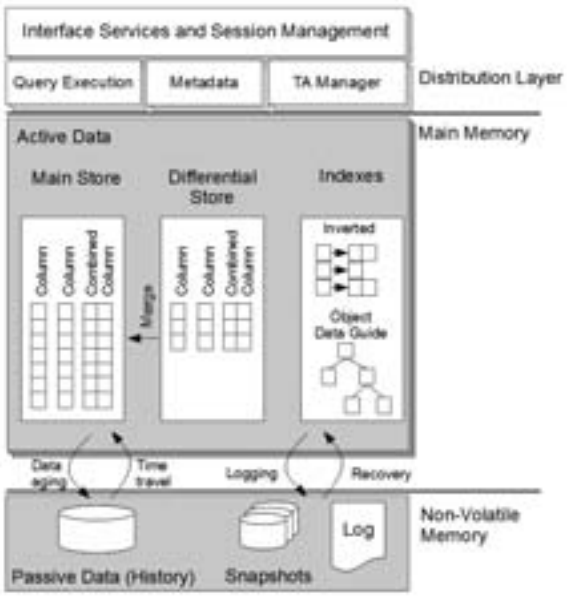


Figure 1: Conceptual overview of the underlying in-memory database [PZ11]

global transactions.
Each node that holds primary data consists of a main store, a differential store, and a collection of indexes. The read optimized main store operates on a sorted dictionary, whereas the write-optimized store appends new values to its dictionary, without sorting the dictionary each time. Each column illustrated in Figure 1 is stored physically as a dictionary vector holding the mappings of values to value IDs and an attribute vector holding the value IDs corresponding to the values of the stored records. If attributes of several columns are accessed mostly together, we can store these columns together to leverage data locality for fast read access. Indexes can be defined to further reduce access time. As enterprise applications access data mostly in terms of business objects, a special index called object data guide is provided, which is a join index for querying the tree-shaped data of business objects [PZ11].
To achieve durability in case of a system failure, the in-memory database writes log information to persistent memory. This log information is used to recover the latest consistent state in-memory in case of a failure and thus guarantees durability. Furthermore, historic data that is not accessed frequently anymore, so-called passive data, can also be shifted to non-volatile memory and loaded if needed.

3 A Case for Mobile Applications with In-Memory Database Backend

Reducing the query response time of even complex analytical queries to seconds opens the way of providing mobile workers with ad-hoc information targeted to their current context. In discussions with several companies, we have identified use cases for mobile applications, ranging from negotiations with business partners to supporting sales representatives during customer dialogs. In these situations, the right information can make the difference of a better informed decision or a better customer experience, and thus a competitive advantage for a company that can provide this information to its mobile workers.
For customer dialogs, all fixed information, such as customer history or the product portfolio can be provided up front. However, not all required information can be anticipated prior to a meeting and some information might also be out of date if it is pre-compiled. An example for outdated information is the current stock level to determine whether a customer order can be delivered at a given date. With mobile frontends for applications that are based on in-memory databases, we can compute queries on the fly. This enables mobile workers to specify the information they actually need in their context of work and prevents the application to show potentially outdated information. Hence, we see a huge potential in mobile applications that allow accessing a company's data in an ad-hoc manner from mobile devices.
Response time is an important factor for the acceptance of a mobile application. However, further factors such as the user interface or the context of use as described in [SW03, PD11] heavily influence the adoption of mobile applications. Especially the user interface needs to be designed in a way that users can quickly define the information they require in the current situation. Furthermore, the results need to be presented in a way that considers form factors such as the smaller screen. Complex results need to be presented in an aggregated form with the possibility of further drilling down if detailed information is required.
In the following Section we describe xSellerate, an application that supports sales representatives in customer dialogs.

4 Case Study xSellerate: Product Recommendations in Real-Time

In this section, we give an overview of xSellerate, an application that supports a sales representative in a customer dialog. The scenario is based on discussions we had with our project partner HILTI, a Liechtenstein based company selling top-quality products for professional customers in the construction and building maintenance industries. We first give an overview of the scenario in which a mobile application can support the sales representative while visiting a construction site. We then present an overview of the prototypical implementation in detail and focus on the use of in-memory technology to realize this scenario.

4.1 Real-Time Information in Customer Dialogs

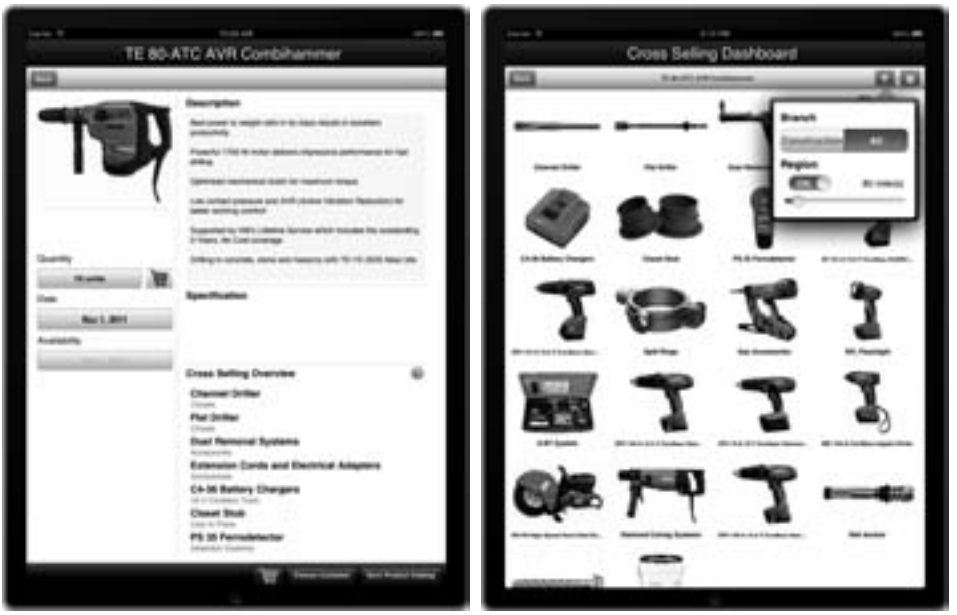
While speaking to our project partner HILTI, we have learned that a well-informed sales representative is key to a successful customer dialog. Good knowledge of the construction industry in general and the customer in specific is required to understand the challenges a customer faces; furthermore, a good knowledge of the product portfolio is necessary to offer correct products and parts. This information is rather static and can be gathered in preparation of an interview. In most cases the sales representative knows the type of construction for which the customer needs support in advance and can prepare a list of potential products to offer up front. However, the exact requirements, as well as the quantity are often not known in advance. Therefore, product offerings during the customer dialog need to be adapted to the specific needs of a customer. Once a customer decides to place an order, the sales rep needs to check whether the products are available at the requested time, which is typically a process that happens after the customer interaction, potentially leading to an additional iteration with the customer.
We prototypically implemented a mobile application to support a sales representative in such a customer dialog. On the one hand, the application provides the information whether a product is available at a specific time. Furthermore, it supports the sales rep in recommending further products. Once the customer has chosen a product, the application generates a list of further products that have been bought by comparable customers that bought the same product. To make these product recommendations more meaningful for the specific customer, the sales rep has the possibility to adjust the list of comparable customers by specifying the industry, as well as the region. These two characteristics have been implemented in our prototype, but any other characteristic of an order, such as the total order volume or the size of the customer, are conceivable parameters to be considered when selecting meaningful product recommendations.
In the following, we give an overview of the prototypical application. We focus on the description of the product recommendations; the underlying concepts of the product availability check have been presented in [TMK+11].

4.2 The xSellerate Prototype

We describe our Prototype xSellerate in this Section. We first give an overview of the features of xSellerate and then describe the architecture and the underlying data model, focusing on the application of in-memory technology.

4.2.1 The Frontend of xSellerate

The frontend of xSellerate is implemented on Apple's iPad device, so that a sales rep can easily take it to its customer interviews. Figure 2 shows the user interface of the xSellerate frontend. Figure 2(a) demonstrates the availability check: for a given product, the sales rep can determine if a product can be delivered at an expected delivery date in the required quantity. Furthermore this view shows a number of potential product recommendations



(a) Selecting the leading product (b) Product Proposal Dashboard with Filters

Figure 2: The xSellerate user interface

in the Cross Selling Overview in the bottom right. To further narrow down the product recommendations, the sales rep can limit the number of proposals and filter the data by region and industry in the Cross Selling Dashboard shown in Figure 2(b). The products are ranked based on the strength of the association between the product chosen by the customer and the recommended product, as described in the next section. Operating on 40 million records, or 45 gigabytes of raw data, our prototype calculates the product recommendations on the fly in sub-second response time, including time for network transmission.

4.2.2 Product Recommendations

In this section, we briefly introduce the basic definitions our product recommendations or so-called cross-selling is based on.

Given there are association rules of the form $A \rightarrow B$ describing that customers who bought a leading product A also bought a dependent product B, then the strength of this association is rated with reference to the key parameters "Confidence", "Support", and "lift" [BKKN08].

Confidence determines how much an order of product B depends on ordering product A at the same time. It is defined as the conditional probability of an order of B given that A

was ordered as well:

$$\text{confidence}(A \rightarrow B) = \frac{|\text{transactions}(A \cap B)|}{|\text{transactions}(A)|}$$

A product recommendation of product B given that A was already ordered is rated strong if the association $A \rightarrow B$ has a high confidence.
Support determines the overall probability that two products A and B have been bought together. It is defined as:

$$\text{support}(A \rightarrow B) = \frac{|\text{transactions}(A \cap B)|}{|\text{transactions}|}$$

However, confidence and support together are not sufficient to rate strong associations, as the relative frequency of the order of B is not taken into account. If the confidence of $A \rightarrow B$ is below the probability that B was ordered, we do not have a strong association. Therefore, a new parameter, called lift, is introduced which exactly addresses this problem. The lift factor takes the number of total orders that contain the dependent product into account and is defined as:

$$\text{lift}(A \rightarrow B) = \frac{\text{support}(A \rightarrow B)}{\text{support(A)} \times \text{support(B)}}$$

In case the order of products A and B is independent of each other the lift is equal 1, otherwise the lift is > 1. However lift alone is also not sufficient: if the support is pretty low, this means that overall, not many customers have ordered the product, and thus, that it is a less attractive recommendation.
Hence, we search our historic sales data for associations of product orders that have high scores in all three parameters, and provide these as recommendations to the sales rep using xSellerate.

4.2.3 The Architecture of xSellerate

Figure 3 gives an overview of the main components of the xSellerate Prototype. The primary sales data our application operates on is generated by enterprise applications, such as an ERP system, and stored in an in-memory database. A web server connects our mobile frontend to the database. As we calculate the parameters for product recommendations as described in the previous Section 4.2.2 on the fly, transferring all historic order information from the database to an application server would be prohibitively expensive. Therefore, we calculate the product recommendations directly in the database, bringing application logic closer to the data, which is a key design decision for fast in-memory applications [PZ11]. This leads to a very slim web server that merely establishes a secure communication channel with the mobile frontend and serves as a mediator between frontend and database.
In our prototypical implementation, we have used the SAP In-Memory Computing Engine [SAP11b] running on a 32 core server with 256 gigabyte of main memory as in-memory database. The web server is based on a slim python framework and the frontend application is implemented for Apple's iOS.

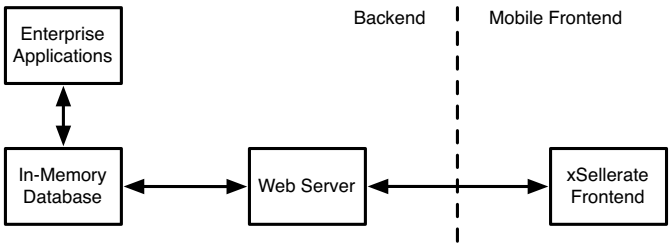


Figure 3: Architecture overview of xSellerate Prototype

4.2.4 In-Memory Data Management of xSellerate

We have adopted the SAP ERP data schema [SAP11a] for the historic sales data that is analyzed to calculate product recommendations. In total, we have loaded around 40 million records of transactional sales, as well as material and customer master data in the in-memory database, summing up to a total of 45 gigabytes of data volume. Product recommendations are calculated on the fly, as we face too many combinations of potential query parameters (products, regions, branches or customers). Furthermore, this leads to a less complex data schema, as we do not have to store any pre-calculated values in form of materialized views.
In the following, we illustrate the computation of the key parameters for product recommendations to a given leading product. The calculations of confidence, support, and lift as described in Section 4.2.2 are directly implemented in SQL and executed in the database server.
We illustrate the calculation by the example of the *support* parameter for two products. Given a customer is interested in a leading product A, we determine all products B that are included in a relation $A \rightarrow B$, meaning that if a past order that included A, also included B. Then, the support of the leading product A and the dependent product B is calculated as follows:

```
SELECT enumerator.value / denominator.value
FROM
      (SELECT COUNT(DISTINCT left.order_ID) as value
       FROM sales_items left,
            sales_items right
       WHERE
            left.order_ID = right.order_ID
       AND
            left.item_ID = lead_product_ID
       AND
            right.item_ID = dependent_product_ID) as
               enumerator,
      (SELECT COUNT(*) as value FROM sales_headers) as
         denominator;
```

In an SAP ERP data schema, the order transactions are stored in a table with all the relevant order information and the ordered items are stored in a separate item table that holds an *order_ID* as foreign key to the header table. In the SQL query above, we call these two tables $sales\_headers$ and $sales\_items$. To calculate all orders that contain the leading and dependent product, we count all distinct $order_ID$s in the $sale\_items$ table that contain both, the leading and the dependent product. To calculate the number of all orders, we count the number of records in the $sales\_headers$ table.
The calculation of the support parameter requires a self join of the $sales\_items$ table, which holds 33 millions records in our scenario. Narrowing down the result set by specifying the region and the industry further increases this complexity, as we create a view containing a join of $sales\_headers$, $sales\_items$ and the $customer$ to obtain a table containing all completed transactions, filtered by region and industry. We have shown this simplified query for the purpose of illustration; note that the actual implementation in our prototype calculates confidence, support and lift for one leading product and all its dependent products in one single query.
As discussed in Section 3, long response times for these analytical queries are not acceptable in a mobile context. In our prototype, the SAP In-Memory Computing Engine returned all product proposals under a second, scanning more than 45 gigabytes of table data; this makes xSellerate a great use case for in-memory data management
In traditional ERP installations finding product recommendations is implemented with association rules which can be configured by the user. The calculation is then implemented as a long running batch job and the results are stored in materialized views, which require additional memory and lead to a more complex data schema. These batch jobs and the materialized view are not required in our proposed application architecture

5 Conclusion

In this paper we have presented xSellerate, a mobile application that supports sales representatives with real-time information in customer dialogs. The key technology to provide the information such as availability checks and product recommendations with response-time in the order of merely seconds is an in-memory database in the backend. Although we have not verified the added value to sales representatives in the field, we have demonstrated the technical feasibility of mobile applications that analyze a company's operational data in real-time. As the amount of mobile workers steadily grows, we see a huge potential for such mobile applications as they allow companies to leverage their enterprise data in completely new ways.

6 Acknowledgments

We especially would like to thank our project partner HILTI. The interaction with HILTI helped us to jointly identify this use case for in-memory technology. The application

XSellerate has been implemented as part of a student project. Therefore, we would like to further thank Markus Dreseler, Kai Hoewelmeyer, Christoph Mueller, Tim Berning, Henning Lohse, Uwe Hartmann, and Maximilian Schneider for their contribution to this work.

References

[BKKN08] Robert Blattberg, Byung-Do Kim, Pyng-do Kim Kim, and Scott Neslin. *Database marketing: analyzing and managing customers*. Springer, 2008.

[BZN05] Peter A. Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*, pages 225–237, 2005.

[Eco11] Oxford Economics. Real-time Business - Playing to win in the new global marketplace. http://www.oxfordeconomics.com/free/pdfs/real-time.pdf, 2011. [Online; accessed 19-October-2011].

[GMS92] H. Garcia Molina and K. Salem. Main memory database systems: an overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, 1992.

[IDC11] IDC. Worldwide Mobile Worker Population 2009-2013 Forecast. http://www.workshifting.com/downloads/2010/07/29/Worldwide2011. [Online; accessed 19-October-2011].

[PD11] Kenny Phan and Tugrul Daim. Exploring technology acceptance for mobile services. *Journal of Industrial Engineering and Management*, 4:339–360, 2011.

[Pla11] Hasso Plattner. SanssouciDB: An In-Memory Database for Processing Enterprise Workloads. In *BTW*, pages 2–21, 2011.

[PZ11] Hasso Plattner and Alexander Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer Verlag, Heidelberg, 2011.

[SAB+05] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel R. Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alexander Rasin, Nga Tran, and Stan B. Zdonik. C-Store: A Column-Oriented DBMS. In *VLDB*, pages 553–564, Trondheim, Norway, 2005.

[SAP11a] SAP. ERP Software From SAP. http://www.sap.com/solutions/business-suite/erp/index.epx, 2011. [Online; accessed 19-October-2011].

[SAP11b] SAP. SAPÂ HANA 1.0 Enables Breakthrough Data Management With In-Memory Computing Engine. http://www.sap.com/press.epx?pressid=14464, 2011. [Online; accessed 19-October-2011].

[SW03] Suprateek Sarker and John D. Wells. Understanding mobile handheld device use and adoption. *Commun. ACM*, 46:35–40, December 2003.

[TMK+11] Christian Tinnefeld, Stephan Müller, Helen Kaltegärtner, Sebastian Hillig, Lars Butzmann, David Eickhoff, Stefan Klauck, Daniel Taschik, Björn Wagner, Oliver Xylander, Alexander Zeier, Hasso Plattner, and Cafer Tosun. Available-To-Promise on an In-Memory Column Store. In *BTW*, pages 667–686, 2011.