Experience Report

Exploiting Advanced Database Optimization Features for Large-Scale SAP R/3 Installations*

Bernhard Zeller Alfons Kemper

Universität Passau
94030 Passau, Germany
<*lastname*>@db.fmi.uni-passau.de

Abstract

The database volumes of enterprise resource planning (ERP) systems like SAP R/3 are growing at a tremendous rate and some of them have already reached a size of several Terabytes. OLTP (Online Transaction Processing) databases of this size are hard to maintain and tend to perform poorly. Therefore most database vendors have implemented new features like horizontal partitioning to optimize such mission critical applications. Horizontal partitioning was already investigated in detail in the context of shared nothing distributed database systems but today's ERP systems mostly use a centralized database with a shared everything architecture. In this work, we therefore investigate how an SAP R/3 system performs when the data in the underlying database is partitioned horizontally. Our results show that especially joins, in parallel executed statements, and administrative tasks benefit greatly from horizontal partitioning while the resulting small increase in the execution times of insertions, deletions and updates is tolerable. These positive results have initiated the SAP cooperation partners to pursue a partitioned data layout in some of their largest installed productive systems.

*This work was supported by an SAP contract within the so-called Terabyte-Project.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

1 Introduction

During the last years the database volumes of ERP systems like SAP R/3 have been growing at a tremendous rate, making them them hard to maintain. Most of the data is stored in only a few very large tables complicating the work of database administrators even further. Especially when a company operates globally and the ERP system has to be accessible at any time, the time slots for performance improvement tasks like re-creation of statistics, indices or execution plans are very small or even non-existent.

Traditionally, performance problems in ERP systems are solved in three steps: First, the ERP system is tuned by creating additional data access paths at the database level (i.e., indices) or by starting many parallel jobs at the application level to minimize execution time or by upgrading to the next software release. In a next step the hardware is tuned by optimizing the storage system or the CPU utilization. If all of these measures fail new computer and storage systems are installed.

However, all of these measures do not alter the sizes of the database. One obvious solution to this problem would be to exploit divide and conquer techniques and to balance the data and the workload across several database instances. But most ERP systems—including SAP R/3—are unable to handle more than one database instance.[1]

To overcome this obstacle, horizontal partitioning—an optimization feature most commercial database system provide today—could be used. Horizontal partitioning was already investigated intensively in the context of distributed databases and it's benefits (e.g., use of divide and conquer techniques) and trade offs (e.g., possibly higher update costs) are well known (see Section 6).

[1] There is work in progress at SAP to alleviate this bottleneck; but until now no such system is generally available.

However, distributed database system are mostly implemented using a shared nothing architecture consisting of many single nodes connected by a wide area network (WAN) or a local area network (LAN). Each single node maintains its own secondary storage media (disks), main memory and CPU, i.e., one CPU has to handle only a few (or even just one) partitions. In this context performance improvements can be gained by exploiting the local computing power of each node, using divide and conquer techniques, and by saving communication costs. In contrast to that the database management systems of today's ERP systems use centralized databases with a shared everything architecture—meaning that there is a small number of CPUs on a single machine with only one main memory and only a few disks/disk controllers. Here, savings in communication costs will have less or even no impact on the performance. Moreover, due to the limited resources hazards on the disk access level or high CPU loads are likelier to happen than in the distributed scenario (see Section 4.3). Until now no detailed performance evaluations of horizontal partitioning techniques in such centralized large-scale installations with a shared everything architecture were available. Therefore, it is difficult for the users to decide whether or not to use horizontal partitioning or to further rely on traditional database layouts in the presence of mission-critical tasks.

We, therefore, were contracted by SAP to investigate the impact of a partitioned database schema on the execution times of the most performance critical statements in a centralized scenario. When analyzing the performance of a database system it has to be taken into account that end users typically don't access a stand-alone database system; rather they use a comprehensive application system in which the database system constitutes an integrated component. In order to derive performance evaluations of practical relevance to the end users, the entire application system, including the database system, has to be benchmarked. In [13] this aspect was already taken into account for decision support queries and a standard database benchmark was used to analyze the performance of database management systems as back-ends of an SAP R/3 system. In addition, in [16] tuning techniques for SAP R/3 systems are described rather than just isolated database tuning techniques.

In this performance evaluation we advanced this methodology and simulated a "real world" application, instead of relying on a standard database benchmark—thereby improving the practical relevance even further. We used SAP R/3 standard components and the database schema we used is part of the comprehensive SAP R/3 company data model. The analyzed statements were extracted from SAP R/3 daily business applications and classified corresponding to their special "SAP structure" (e.g., *select single*, *for all entries*, or *select up to n rows*). A representative of each class of statements was chosen as the basis for our analysis. The data was extracted from an actual productive SAP R/3 system rather than relying on artificially generated data, as, e.g., the TPC-C benchmark does [25].

We analyzed a variety of partitioning schemes (data with and without index partitioning) and compared the performance with a conventional non-partitioned database configuration. Our results show that also in the context of centralized shared everything database systems horizontal partitioning can improve the performance of certain tasks significantly. Especially joins, in parallel executed statements, and administrative tasks benefit greatly from horizontal partitioning while the resulting small increase in the execution times of insertions, deletions and updates is tolerable. These positive results have initiated the SAP cooperation partners to pursue a partitioned data layout in some of their largest installed productive systems.

For the sake of readability we use the term *partitioning* as an abbreviation for *horizontal partitioning* throughout the rest of the paper.

The remainder of this paper is organized as follows: Section 2 describes briefly the architecture of an SAP R/3 system. Section 3 gives an overview of some traditional approaches to meet certain performance problems. Section 4 gives a brief overview of the implemented partitioning features. It shows how problems can be solved using partitioning and describes possible drawbacks of horizontal partitioning. In Section 5 we describe our performance evaluation environment and present our results. Related Work is addressed in Section 6. Section 7 concludes the paper.

2 Overview of SAP R/3

SAP R/3 is the market leader for integrated business administration systems. It integrates all business processes of a company and provides modules for finance, human resources, material management, etc. SAP R/3 is based on a (second party) relational database system which serves as an integration platform for all components of SAP R/3. The database system manages the SAP database which stores all business data of a company (e.g., customer and supplier information, orders, ...), all of SAP R/3-internal control data, an SAP R/3 data dictionary, and the code of all application programs. Virtually no data are stored outside this SAP database, thereby avoiding the use of a file system.

SAP R/3 [26, 3, 23, 13] is based on a three-tier client/server-architecture with the following layers (see Figure 1):

1. The presentation layer. It provides a graphical user interface (GUI) usually running on PCs that are connected with the application servers via a local (LAN) or a wide area network (WAN).

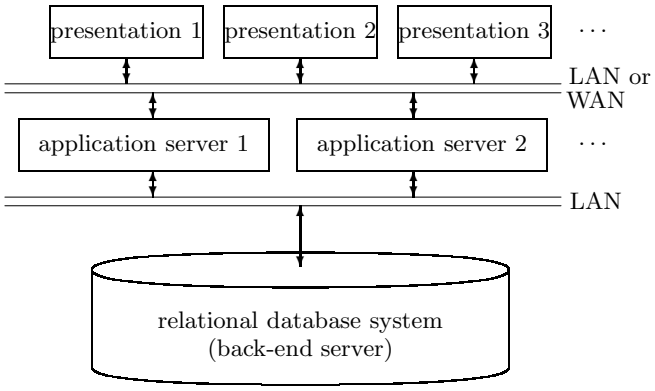

Figure 1: Three-Tier Client/Server-Architecture of SAP R/3

2. The application layer. It comprises the business administration "know-how" of the system. It processes pre-defined and user-defined application programs such as OLTP and the implementation of decision support queries. Application servers are usually connected via a local area network (LAN) with the database server.

3. The database layer. It is implemented on top of a (second party) commercial database product that stores all data of the system, as described above.

In a small company that uses SAP R/3, the application servers and the database system could be installed on the same middle-range machine and users would enter business transactions or issue decision support queries using their PCs. Such a configuration, however, is not practical for large companies with a very high volume of data and transactions. In such companies, all application servers and the database system would be installed on separate dedicated machines. To this end, SAP R/3 has been ported to a large variety of hardware and operating system platforms, and it is also operational on a number of commercial RDBMSs.

SAP R/3 is a comprehensive and highly generic business application system that was designed for companies of various organizational structures and different lines of business (e.g., production, retailing, finance, ...). This genericity and comprehensiveness resulted in a very large company data model with over 13.000 database tables. To manage the meta data (e.g., types and interrelationships) of these tables, SAP R/3 maintains its own data dictionary which is (like all other data) stored in SAP's relational database and which can be used by SAP application programs.

An example for a large-scale installation of SAP R/3 is the SAP R/3 System of Deutsche Telekom AG [21]. Deutsche Telekom is one of the key global players in the telecommunication market. Deutsche Telekom makes extensive use of the Financial Accounting component (FI) of SAP R/3 to manage the accounts of their customers. There are extremely high volumes of data involved in booking the large number of invoices, dealing with payment received, processing debits and reminders: Each of their 15 SAP R/3 systems has to handle 200,000 invoices, about 12,000 reminders, and approx. 10,000 modifications to customers' accounts per day. To each of these systems up to 1000 users are connected at a time. To handle this workload 51 Unix enterprise servers (RM600E) as database and application servers, running under the SINIX operating system from Siemens Nixdorf, PCs from a variety of vendors as clients, and 34 storage subsystems from EMC with a total capacity of 30 Terabytes are installed at the computing centers of Deutsche Telekom. The backup is handled by 68 magnetic tape drives, which back up the whole database within 2 hours.

3 Traditional Performance Tuning Techniques

Performance problems mostly arise when the tables of a database grow and the existing data access paths become inefficient. Consider for example, an application that searches German customers with a sales volume of 100 K € and above. To do so, the application accesses an index on the *Sales* field of the table

$$\textit{Customer}\ \langle \textit{Name, Address, Sales} \ldots \rangle$$

and selects all Customers with a sales volume of 100 K € and above. To find the German customers the application has to filter the selected tuples by examining the *Address* field. While the database is small only a few entries meet the restriction on the sales volume field and the response time is sufficient. When the database grows also the number of customers with a sales volume of 100 K € might grow and filtering all matching entries becomes inefficient. To solve this problem, traditionally, an index is created indexing the fields *Sales* and *Address*. But additional indices slow down updates and insertions and make the database harder to maintain. Moreover, when the table is very large, also the indices are very large and don't fit into main memory all together. When B-trees are used for indexing the indices are often degenerated because in most ERP systems ascending ordered numbers are used as artificial keys [5].

Another problem of ERP systems results from jobs that process a large volume of data, e.g., stock-taking jobs. To ensure that mission-critical daily business (order entry, order fullfillment, ...) is not affected by these jobs they have to be processed within fixed time slots. When the data volume grows more instances of a job are started at the application level and the data is processed in parallel to meet the time restrictions. But at the application level exists no knowledge of the database schema and therefore it is hard to spread the processing units across the running jobs correctly. A wrong distribution of the processing units leads to an unbalanced workload and to bottlenecks and can cause access conflicts at the disk page level. Consider

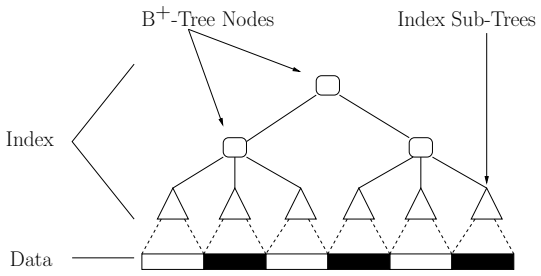

Figure 2: Non-Partitioned Index Layout

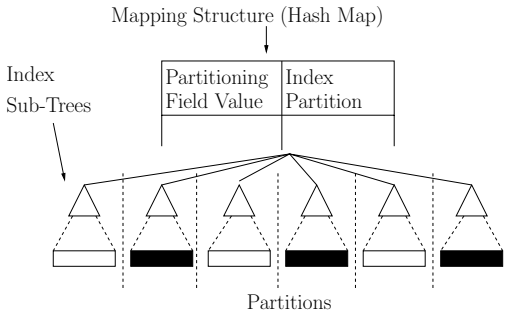

Figure 3: Partitioned Index Layout

for example, an application that stores the data of several stores and clusters the data according to product groups to support searching. Due to the clustering data of different stores is stored on the same disk page which causes I/O conflicts when the data of several stores is processed in parallel. Moreover, during the growth of the tables an existing, well designed process unit distribution can become obsolete due to data skews.

Another area where large tables occur are data warehouses. The data of these data warehouses is extracted from the ERP systems and stored in a separate database. Sliding window techniques are used when storing the data, i.e., only a snapshot of a few years of the data of an ERP system is stored in a data warehouse, e.g., the data of the last 6 years. When new data is loaded in the warehouse the oldest data is deleted. This deletion process is very expensive in a traditional, non partitioned, table layout. Although there are possibilities to speed up this process [14] it might still not be fast enough.

If all traditional improvements fail, new hardware is used to solve the performance problems. Special disk layouts are used in conjunction with special storage systems (e.g., RAID systems) to speed up I/O.

4 Exploiting Partitioning for Tuning Purposes

The benefits of partitioning were already investigated in detail in the context of distributed database systems (see Section 6 for details). However, our results show that partitioning is also useful in centralized systems to keep large volumes of data manageable.

Section 4.1 describes some of these advantages gained by horizontal partitioning. Section 4.2 gives a short overview over possible partitioning techniques. In Section 4.3 some possible drawbacks of partitioning are shown.

4.1 Possible Benefits of Partitioning

Partitioning can be used in centralized systems with a shared everything architecture to keep large volumes of data manageable (see Section 5). E.g., when statistics have to be gathered and the time slots for administrative tasks are small, the workload can be divided by gathering statistics partition-wise. Also, index re-creation and table re-organization can be done partition-wise, thereby reducing the off-line times of the database and minimizing the impact on running applications. In the presence of (equi)joins partitioning can improve the overall performance when the partitioning fields are a subset of the join attributes and both tables are partitioned accordingly. Here, the joins can be done partition-wise and in parallel.

In the case that B^+-trees are used for indexing the main memory usage and the look up time can be improved using index partitioning. When a B^+-tree is partitioned, a mapping structure is needed that indicates which index partition indexes which part of the table. This mapping structure corresponds to the root node in a non partitioned B^+-tree and the index partitions correspond to the subtrees below the root node.

However, the mapping structure is generally a small memory resident hash map that is not bounded by the page size and therefore can manage an arbitrary number of index partition references. The root of a B^+-tree can only store a limited number of references depending on the size of a disk page and the size of a stored reference [8]. When the limit is reached the node is split and a new root is created. After this point in time the mapping structure of the partitioned scenario (i.e., the hash map) corresponds to a small tree in the non-partitioned scenario (see Figures 2,3). This leads to more main memory consumption when the index nodes are not entirely filled and slower look up times, because binary search is used within each index node and not a single hash map lookup as in the partitioned version.

In most commercial database systems a secondary storage page (disk page) contains only data of one table (i.e., partition) and not of different tables. Therefore also parallel jobs can benefit from partitioning and avoid conflicts on disk page level when they process the data partition-wise because then the jobs work on different physical working sets (i.e., disk pages).

When it comes to mass deletions within a database, the database management systems can benefit from partitioning, too. Mass deletions are a severe problem,

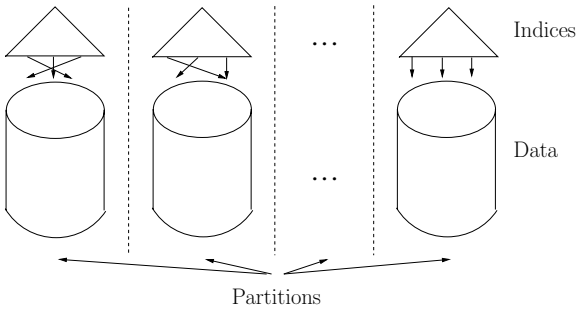

Figure 4: Equi-Partitioned Index

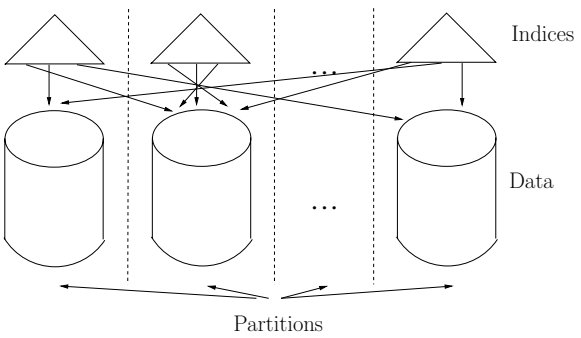

Figure 5: Non-Equi-Partitioned Index

because they often cause access hazards on disk page level (thrashing) [14]. When the data is partitioned accordingly, such mass deletions can be handled by dropping whole partitions. This outperforms the conventional tuple-at-a-time approach used in database management system without partitioning by an order of magnitude. Instead of executing hundreds of single deletions the data is deleted by just freeing some pages on the secondary storage media. In SAP R/3 systems such mass deletions can occur during an *archiving* process. Archiving is an SAP technique and is used to shrink the size of production databases by moving seldom used data to slower tertiary storage systems (tapes). During this process, mass deletions occur within the production databases.[2]

Furthermore, data warehouses can profit from partitioning by storing records corresponding to their creation date. Old data can then be deleted by just dropping a partition and new data can be added by adding a partition.

4.2 Partitioning Techniques

Most modern database systems provide several methods to partition a given non partitioned table and its indices into several smaller tables and indices, the so called *partitions*. The partitioning itself is transparent to the users, i.e., the users cannot access a partition directly. Instead, they access the table and the database system chooses the right partition. There exist many algorithms to spread the data over the partitions, but all partitioning techniques have in common that one or more fields of the table have to be chosen as *partitioning fields*. The values of the partitioning fields of a tuple determine in which partition the tuple will be stored. The most important partitioning techniques are *range partitioning* and *hash partitioning*. Also combinations of the two strategies are possible. When a table or index is range partitioned, then a partition stores all tuples, whose partitioning field value lies within a given range. The range for each partition has to be specified at table creation time.

[2]In a more comprehensive context, archiving is studied as part of the SAP Terabyte project in cooperation with the University of Passau.

Range partitioning is useful when the range of values of a field is known in advance and doesn't change. An advantage of range partitioning is, that the users can predict in which partition a tuple will be stored by looking at the partitioning field values. But range partitioning performance deteriorates when the domain of the partitioning field is large because then it is hard to distribute the data uniformly over the partitions.

When hash partitioning is used a hash function is applied to the values of the partitioning fields of a tuple. Depending on the hash value, the tuple is stored in the corresponding partition. Partitioning by hashing spreads the data more uniformly across the partitions—provided that the hash function is chosen adequately. On the other hand, users cannot predict in which partition a tuple is stored by just looking at the partitioning field values if the hash function is complex. This leads to problems when processing units of parallel jobs have to be defined.

There exist many partitioning algorithms which are mixtures of range and hash partitioning. These algorithms allow to re-partition single partitions to derive finer partitions, the *sub-partitions*. One commonly used approach is to partition a table by range and then partition the table partitions again into sub-partitions using hash partitioning. This gives the users on the one hand knowledge about the over all partitioning (range partitioning) and on the other hand keeps the sub-partitions balanced (hash partition). This way, the users can define the data they want to work on and the database system can work in parallel on the sub-partitions without being in danger of unbalanced workloads.

No matter whether the table is partitioned or not, the indices can—but need not—be partitioned. When both, the table and the indices, are partitioned, the indices can be partitioned using the same partitioning fields and ranges as used for partitioning the table (*equi-partitioning*) or by using different ones (*non-equi-partitioning*). In an equi-partitioned scenario each index partition indexes exactly one table partition. This makes it easy to drop and create data in units of partitions (see Figure 4). In a non-equi-partitioned scenario one index partition indexes data that belongs to dif-

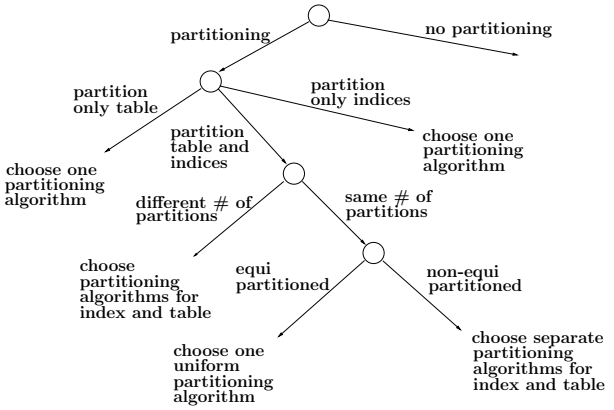

Figure 6: Possible Partitioning Scenarios

ferent table partitions (see Figure 5).

However, the partitioning strategy for indices can easily be changed dynamically by dropping and recreating the indices. This enables the administrators to adapt the database layout to changing needs. Also additional indices for only a few partitions can be created to minimize the lookup time for certain applications. In contrast to that, the partitioning strategy for the tables cannot be changed at run time in a large-scale database, because this would involve the movement of all table data. Therefore, the partitioning strategy for the tables has to be chosen very well and not only database concerns but also concerns of the application using the database have to be taken into account. Figure 6 gives an overview of the possible approaches for deriving a partitioning scheme.

4.3 Possible Drawbacks of Partitioning

Partitioning was investigated in detail in the context of distributed database systems with a shared nothing architecture. In this context partitioning improves the performance of tasks like parallel jobs or joins. In the context of centralized database systems with a shared everything architecture also performance improvements are possible as discussed in the previous section.

But partitioning can also slow down certain tasks, e.g., when row movements occur during updates. Row movement means that a tuple has to be moved from one partition to another because the partitioning field has been updated. E.g., when the plant number of a record is changed from 1 to 14 and the table is range partitioned according to the plant number (e.g., 20 plants; 4 partitions, 5 plants stored together in one partition), then the record has to be deleted in the partition storing all plants with a number less than 6 and has to be inserted in the partition storing all plants with a number between 11 and 15. The row movement doubles the costs corresponding to non partitioned tables. Especially OLTP systems like SAP R/3 have a need for fast execution of update statements to accomplish their work.

Furthermore, insertions and deletions can be influenced by partitioning because additional administrative structures have to be accessed and maintained. Such additional structures are, e.g., the meta-data storing the number of partitions, the hash maps for accessing the right partitions, and additional statistics used by the optimizer of the database management system. These additional accesses can slow down insertions and deletions which are frequently occurring operations in OLTP systems.

Also the parallelization of single jobs at the database management system level can slow down an ERP system significantly, because it conflicts with the parallelization at the application level (multiple instances of one job). Normally, the resources (e.g., CPUs, main memory, storage) of an ERP system are managed at the application level. That means, that special dispatcher applications schedule the jobs at application level depending on a given time schedule or the actual workload. This strategy allows the users to adapt the ERP systems to their needs by, e.g., defining time slots for administrative tasks.

However, the dispatcher applications have normally no knowledge of the parallelization of jobs at the database level. Therefore, they work inefficient when the underlying database is partitioned. Consider, for example, an ERP system that runs on a machine with 4 CPUs. Without parallelization at the database level, a job occupies one CPU and the other 3 CPUs can be used by other (perhaps mission-critical) jobs. Therefore, from the dispatcher application's point of view, the workload is balanced well. But when parallelization at the database level is used all 4 CPUs might be used by one job because this maximizes the degree of the parallelization and minimizes the execution time. This strategy prevents other jobs from executing which again increases their response time. This is unexplainable from the dispatcher application's point of view. Here, the parallelization at the application level (multiple instances of one job) conflicts with the parallelization at the database management system level because they have no knowledge of each other.

Therefore, the possible benefits of a partitioned table layout as described in the previous section are not sufficient to motivate the users to switch their database layout. Also the possible trade-offs and the impact on every-day transactions have to be investigated before world wide operating ERP system vendors like SAP make use of a partitioned table layout.

5 Performance Analysis

We analyzed a variety of partitioning schemes (data with and without index partitioning) and compared the performance with a conventional non-partitioned database configuration. We analyzed several state-

ments that were extracted from SAP every day business applications and carried out the analysis in an SAP 4.6C system using the SAP performance evaluation tool SSQJ.

Section 5.1 gives a short overview of the SAP tool SSQJ. Section 5.2 describes our performance evaluation environment and in Section 5.3 we present our results.

5.1 The Performance Analysis Tool SSQJ

The tool SSQJ is an SAP internal performance analysis and testing tool which is used for quality assurance and validation of new techniques. To improve the benefit of this tool also database vendors may use the tool to test their new software releases. SSQJ mainly covers the interaction between the application server and the underlying database management system. To do so, SSQJ provides 3400 test statements divided in the three areas *ABAP* (1000 statements), *SQL* (1800 statements), and *Large Table* (600 SQL statements). Like any other application of the SAP R/3 system SSQJ is coded in the programming language ABAP/4 (**A**dvanced **B**usiness **A**pplication **P**rogramming Language) [18]. Except for a small kernel, actually the entire R/3 system is coded in ABAP/4. ABAP/4 is a so-called Fourth Generation Language (4GL) whose origins can be found in report/application generator languages. ABAP/4 provides commands that allow to access the database via two different interfaces: *Native SQL* and *Open SQL*. The Native SQL interface can be used via so-called EXEC SQL commands. It allows the user to access the SAP database directly without using the SAP-internal data dictionary.

The SSQJ tool itself uses only the *Open SQL* interface but it can be used to test both *Native SQL* and *Open SQL* programs. To enable users to test their own programs and statements SSQJ provides a framework where ABAP or SQL code can be plugged in and the execution time and the resource consumption can be measured. This is done by monitoring the Open and Native SQL interfaces of the SAP system, the database and some of the internal SAP structures like caches and main memory structures. The plugged in code is stored within the tool for later use. Such an fragment of code is called a *case* in SSQJ. During the years SSQJ comes along with more and more cases making the tool more and more powerful, e.g., SSQJ includes a version of the TPC-D benchmark we integrated in SSQJ as part of another project [13]. Also the new partitioning cases are now part of SSQJ and can be used for testing.

The results of a measurement can be stored, evaluated, and compared to previous measurements. Along with the results also environment data like the SAP R/3 and the database version are stored to improve the comparability of the results. The stored results can be evaluated using the tool itself or can be exported to Microsoft Excel for further processing.

5.2 Evaluation Environment

The SAP system we used for our performance evaluation was an SAP R/3 4.6C system with one of the major commercial database systems as underlying database. The SAP system was installed on a SUN Enterprise 450 with four 400 MHz processors and 4 GB main memory. As storage system we used a SUN A1000 500 GB RAID system with RAID level 5. The commercial database system used 512 MB main memory and the SAP system an additional 512 MB.

The data we used for our investigation was anonymizated data from a productive SAP system. To increase the data volume the original data was replicated by changing the values of unique fields of the tables. This is an SSQJ standard procedure and the volume of created data can be controlled by an SSQJ parameter.

We used copies of two SAP tables (*MARC*, *MARD*) for our analysis. Both tables store material related data, but *MARC* is a table with an average tuple length of 496 Bytes whereas the tuples of the table *MARD* have only an average length of 142 Byte. Also the number of rows in each table differ to compensate the differences in the length of the tuples (*MARC*: 5 million rows, *MARD*: 25 million rows). Each table has one index, the primary index, which indexes the following fields: *MANDT* (the SAP client number), *MATNR* (the ID of the material to be stored), and *WERKS* (the number of the plant, where the material is stored). The index on table MARD additionally indexes the field *LGORT* that is used to indicate where in the specific plant the material is stored (e.g., the ID of the factory floor). The field WERKS, which stores the IDs of the different plants of a company, was used for partitioning. Therefore the creation of the data was modified to ensure that exactly 100 different WERKS values were created simulating a company with 100 plants. For each plant the same data volume was created. To be able to compare the results directly with a non-partitioned layout 3 versions of each table were created[3]:

- *Flat*: the SAP table layout, i.e., neither the tables nor the indices are partitioned.
- *Global index*: Only the table is partitioned but the corresponding index is not.
- *Partitioned Index*: The table and the indices are partitioned.

This layout of the tested data should users enable to decide if it is promising to use the new database feature

[3]We omitted the analysis of partitioned indices on top of a non-partitioned table because this partitioning scenario is of no practical relevance and we regard it as not promising.

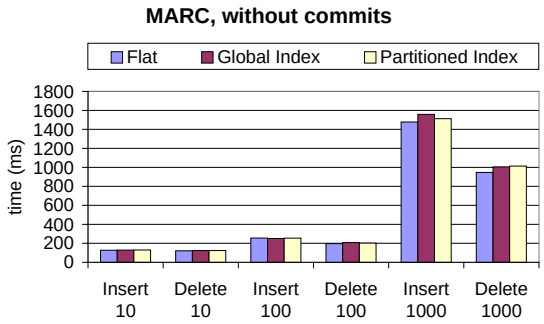

Figure 7: Table MARC, Insertions and Deletions Without Commits

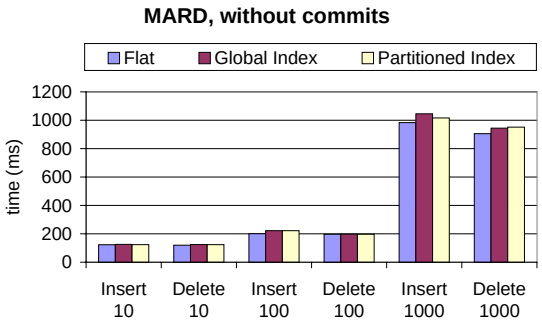

Figure 9: Table MARD, Insertions and Deletions Without Commits

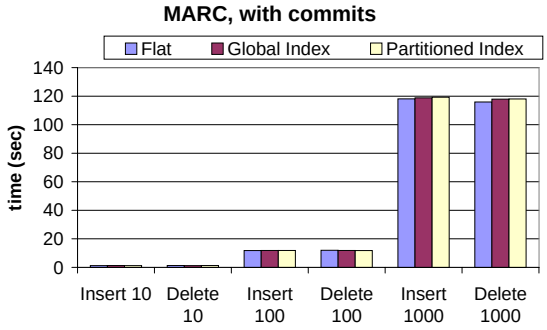

Figure 8: Table MARC, Insertions and Deletions With Commits

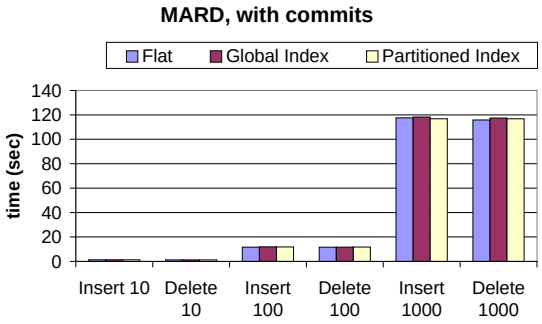

Figure 10: Table MARD, Insertions and Deletions With Commits

and to switch to a partitioned layout or to stay with the standard, non-partitioned layout.

In the partitioned cases the tables where range partitioned according to their WERKS value. It was ensured that the data of exactly one plant was stored in one partition ending up with 100 partitions. In the case the indices were also partitioned we used equi-partitioning which produced another 100 index partitions.

5.3 Analyzed Statements

To generate the statements used for our analysis we examined several SAP mission-critical daily business applications and extracted the included statements. Afterwards we classified the extracted statements and chose one representative of each class as basis for our investigation. This classification is reflected in structure of the SSQJ partitioning cases and covers most of the commonly used ABAP/4 Open SQL query types, e.g., *select single, for all entries, up to n rows*, etc.

During the performance evaluation we analyzed different scenarios, i.e., we varied the number of processed tuples, used set orientated and one-tuple-at-a-time processing techniques, and varied the number of commits. Altogether we analyzed more than 70 different versions of the analyzed statements. Presenting all the results is beyond the scope of this work. We therefore focus on the important results. However, all our results showed that partitioning is applicable at negligible costs.

5.3.1 Single Insertions and Deletions

Insertions of a single or only a few tuples are a very common operation in an ERP system like SAP R/3. For completeness we also examined the deletion of single tuples. During our tests we varied number of processed tuples, the commit rate, and the way the data is passed to the database system. Our results show that there is no significant difference in the execution times of the partitioned and the non partitioned versions when the work done is commited after processing the data (see Figures 7,9). Moreover, the difference is that small, that the overhead of commiting each processed tuple separately dominates the execution times hiding the partitioning overhead completely (see Figures 8,10).

We also tested set-oriented variants of the insert and delete statements, i.e., a temporary table was filled with the tuples that have to be processed and this table was sent to the database for processing. Here the work done is commited after the whole temporary table is processed. Again, the execution times didn't rise sig-

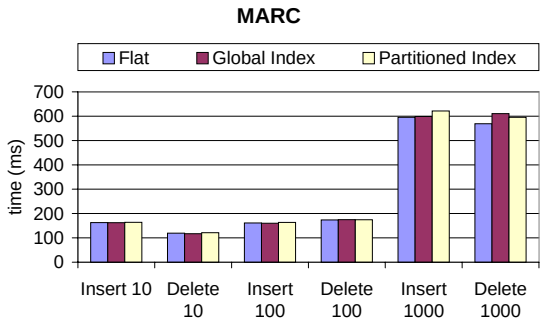

Figure 11: Table MARC, Set Orientated Insertions and Deletions

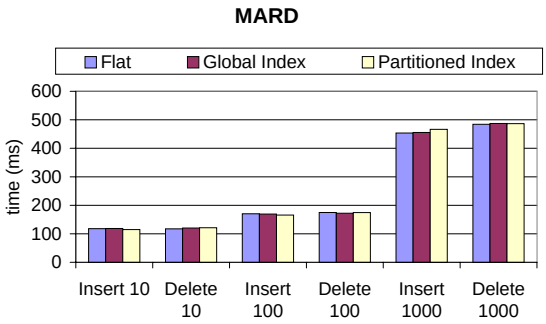

Figure 13: Table MARD, Set Orientated Insertions and Deletions

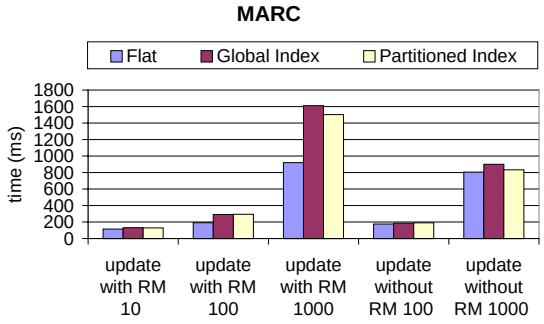

Figure 12: Update Statements on Table MARC

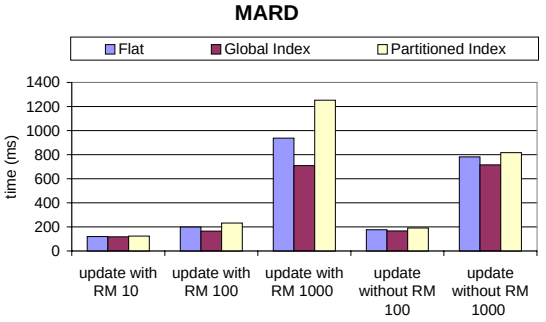

Figure 14: Update Statements on Table MARD

nificantly using the partitioned database schema compared to a non partitioned database schema (see Figures 11, 13).

5.3.2 Update Statements

When tables are partitioned updates can be a bottleneck when it comes to *row movements (RM)*. Row movement means that a tuple has to be moved from one partition to another because the partitioning field has been updated. E.g., when the plant number of a record is changed from 1 to 14 and the table is range partitioned according to the plant number (e.g., 20 plants; 4 partitions, 5 plants stored together in one partition), then the record has to be deleted in the partition storing all plants with a number less than 6 and has to be inserted in the partition storing all plants with a number between 11 and 15. The row movement doubles the costs corresponding to non partitioned tables. However, there are two reasons why this is no argument against partitioning: First, modern database systems can handle updates that fast, that a slow down of a factor 2 is tolerable for most applications. Second, in a well designed database layout updates to partitioning fields are not likely.

When other fields than the partitioning fields are updated there is again no difference between partitioned and non-partitioned tables as can be seen in Figure 12 and 14.

5.3.3 Select Statements

Another important kind of operation in ERP systems are selections, especially selections with fully specified key values. Also this kind of statements performed well in a partitioned scenario. (see Figure 15).

However, also selections where not all indexed field values but only a prefix of them is specified are important in the context of many mission-critical SAP business applications. Here, a set of tuples is selected from the table. In a partitioned database schema this kind of selections are of special interest, when the selected tuples are spread over several partitions. In our tests we pushed it even further and selected 10 tuples from every partition meaning that 100 index partitions and 100 table partitions have to be visited to answer the query. Despite this overhead the execution times in the partitioned scenario differ only slightly from the ones of the non partitioned scenario—as can be seen in Figure 17.

5.3.4 Parallel Select Statements

We also tested parallel selects, i.e. 10 parallel running jobs executed several select statements. Each of the statements selects 10.000 tuples. In one variant of the

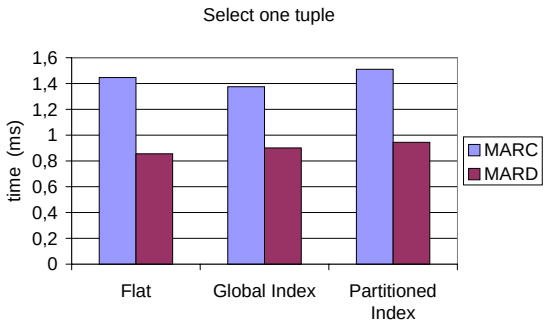

Figure 15: Select one Single Record

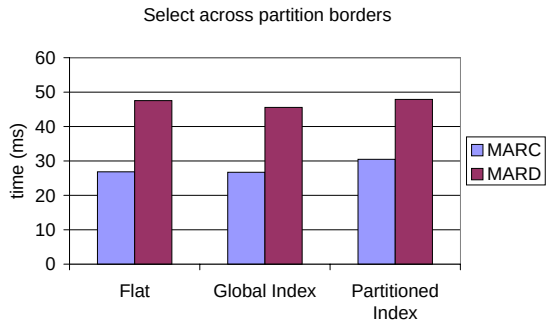

Figure 17: Selections Across Partition Borders

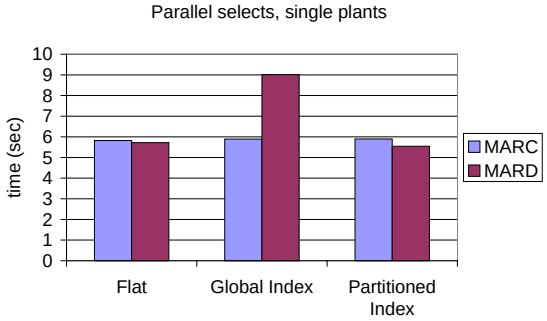

Figure 16: 10 Parallel Select Statements Selecting 100.000 Records (single plants)

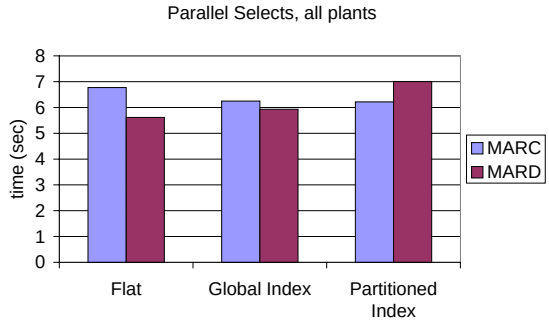

Figure 18: 10 Parallel Select Statements Selecting 100.000 Records (all plants)

tested cases called *single plants* each job only selects tuples of exactly on plant. That means that the 10 jobs work on different working sets. In the *all plants* variant the jobs select tuples randomly and were not constraint to partitioning borders.

Again the results show, that partitioning doesn't slow down execution times despite of one result where the execution time rises dramatically. This rise is caused by an problem of the underlying database system when handling global indices on partitioned tables and will be fixed in the near future.

The results show no significant difference in the running times of the single plant variant and the all plants variants. The reason for this is the small number of parallel jobs. If the number of parallel jobs rise the separation of the working sets pay off and the single plants variant would outperform the all plants variant. Our test system was too small to show this effect but SAP made tests on their own on system with several hundreds of GBs where they could verify this effect. Figure 16 and 18 show our results.

5.3.5 Join Statements

In the presence of joins the partitioned versions outperform the non partitioned versions by an order of magnitude because the used hash join algorithm can make extensive use of the partitioning by joining the tables partition-wise and in parallel. During the analysis of the join statements we investigated following two scenarios: First, one plant of the table MARC is joined with one plant of the table MARD. Second, the whole tables are joined. The results are depicted in Figures 19 and 20, respectively.

5.3.6 Administrative Tasks

Administrative tasks can benefit from partitioning by splitting up the work in several smaller parts. This is necessary because even with a small database like our test database administrative tasks like the creation of table statistics can already take hours (see Table 1). Therefore most databases vendors have adapted their administrating tools to work on single partitions rather than on whole tables. This cuts down the execution times by a factor equal to the number of partitions. If, e.g., a time slot is to small to process the whole table, only some partitions are processed and the rest of the work is done when the next time slot is available. It is also possible to process several partitions in parallel.

6 Related Work

Partitioning was already investigated in detail in the context of distributed shared nothing databases. The main focus lied on minimizing the communication

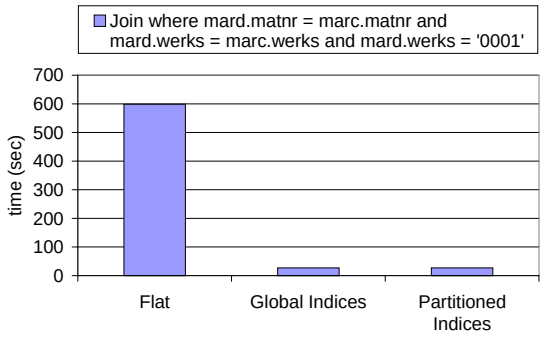

Figure 19: Joining Single Plants

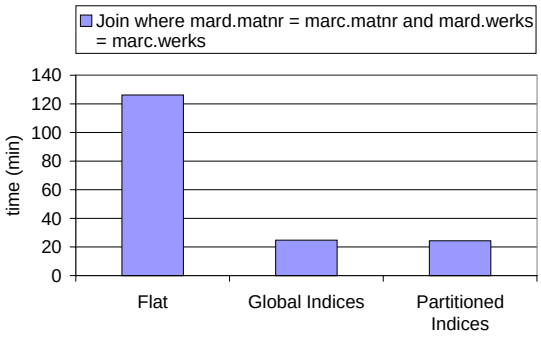

Figure 20: Joining the Whole Tables

MARD

	whole table	one partition
Flat	5 h 43 min	not possible
Global Index	5 h 8 min	12 sec
Partitioned Index	5 h 2 min	14 sec

MARC

	whole table	one partition
Flat	1 h 43 min	not possible
Global Index	2 h 1 min	10 sec
Partitioned Index	1 h 57 min	9 sec

Table 1: Analyze Table Statements

costs and to exploit the local computing resources. A lot of work was done on placing data in the context of distributed shared nothing systems. Blankinship, Hevner, and Bing Yao [1] proposed a heuristic which provides an integrated solution to the query optimization and data allocation problems in distributed database systems. Copeland, Alexander, Bougther, and Teller investigated the problem of data placement in BUBBA, a highly-parallel system for data-intensive applications [9, 2]. DeWitt et. al. [10, 11, 12] described and evaluated the data placement strategies in GAMMA, a relational database machine where all data is partitioned horizontally. In [19] Mehta and DeWitt presented a simulation study of data placement issues in shared nothing systems. Another issue that is important in the context of partitioning and already well investigated is the placement of data on different disks to achieve maximum I/O parallelism. Chen, Rotem, and Seshadri investigated in [6] the problem of adapting existing declustering methods to work in heterogeneous environments. In [7] Christodoulakis and Zioga investigated design principle of placing striped delay-sensitive data on a number of disks in a distributed environment. In [22] Scheuermann, Weikum and Zabback studied striping and load balancing techniques in parallel disk systems and showed their relationship to response time and throughput. Also the treatment of indices in a partitioned scenario was already investigated in the past. In [17] Liebeherr, Omiecinski, and Akyildiz presented an approach to process an index partition scheme where a global index is partitioned across nodes. In [24] Seeger and Larson investigated multi-disk B-trees. There was also work done on declustering and partitioning in general. In [4] Ceri, Negri, and Pelagatti investigated the problem of horizontally partitioning data on a set of resources. Ghandeharizadeh and DeWitt presented in [15] a new declustering strategy for multiprocessor database machines. Nowitzky described in [20] the new implemented partitioning techniques of most commercial databases and gave examples of possible database layouts.

7 Conclusion

In the past ERP systems like SAP R/3 were tuned using traditional approaches like database reorganizations or hardware improvements. In the presence of globally operating companies, 7 by 24 applications, and large-scale databases these traditional techniques are no longer sufficient. We therefore investigated in this work how advanced database system features, in particular horizontal partitioning, can be exploited to optimize large-scale SAP R/3 installations.

To produce results of practical relevance and to reflect the actual use of a database management system as an integrated component of a comprehensive application system rather than a stand-alone database system we carried out our performance evaluation within an SAP R/3 system. We simulated a “real world” application by extracting the analyzed statements from SAP daily business applications and by using parts of the comprehensive SAP database schema as basis for our investigations. Moreover, the tested database was generated from an actual productive SAP R/3 system rather than relying on artificially generated data, as, e.g., the TPC-C benchmark does. We analyzed a variety of partitioning schemes (data with and without index partitioning) and compared the perfor-

mance with a conventional non-partitioned database configuration. Our results show that especially joins, in parallel executed statements, and administrative tasks benefit greatly from horizontal partitioning while the resulting small increase in the execution times of insertions, deletions and updates is tolerable. These positive results have initiated the SAP cooperation partners to pursue a partitioned data layout in some of their largest installed productive systems.

References

[1] R. Blankinship, A.R. Hevner, and S. Bing Yao. An iterative method for distributed database design. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 389–400, Barcelona, Spain, September 1991.

[2] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping bubba, A highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[3] R. Buck-Emden and J. Galimow. *SAP R/3 System, A Client/Server Technology*. Addison-Wesley, Reading, MA, USA, 1996.

[4] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 128–136, Orlando, USA, 1982.

[5] F. Cesarini and G. Soda. An algorithm to construct a compact B-Tree in case of ordered keys. *Information Processing Letters*, 17(1):13–16, 1983.

[6] L. T. Chen, D. Rotem, and S. Seshadri. Declustering databases on heterogeneous disk systems. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 110–121, 1995.

[7] S. Christodoulakis and F. Zioga. Data base design principles for striping and placement of delay-sensitive data on disks. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 69–78. ACM Press, 1998.

[8] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[9] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in bubba. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 99–108, Chicago, IL, USA, May 1988.

[10] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna. Gamma - a high performance dataflow database machine. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 228–237, Kyoto, Japan, 1986.

[11] D. J. DeWitt, S. Ghandeharizadeh, and D. A. Schneider. A performance analysis of the gamma database machine. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 350–360, Chicago, IL, USA, May 1988.

[12] D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I Hsiao, and R. Rasmussen. The gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[13] J. Doppelhammer, T. Höppler, A. Kemper, and D. Kossmann. Database performance in the real world: TPC-D and SAP R/3. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 123–134, Tucson, AZ, USA, May 1997.

[14] A. Gärtner, A. Kemper, D. Kossmann, and B. Zeller. Efficient bulk deletes in relational databases. In *Proc. IEEE Conf. on Data Engineering*, pages 183–192, Heidelberg, Germany, 2001.

[15] S. Ghandeharizadeh and D.J. DeWitt. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 481–492, Brisbane, Australia, August 1990.

[16] A. Kemper, D. Kossmann, and B. Zeller. Performance tuning for SAP R/3. *IEEE Data Engineering Bulletin*, 22(2):32–39, June 1999.

[17] J. Liebeherr, E. Omiecinski, and I. F. Akyildiz. The effect of index partitioning schemes on the performance of distributed query processing. *TKDE*, 5(3):510–522, 1993.

[18] B. Matzke and A. Weinland. *ABAP/4 - Programming the SAP R/3 System*. Addison-Wesley, Reading, MA, USA, 1997.

[19] M. Mehta and D. J. DeWitt. Data placement in shared-nothing parallel database systems. *VLDB Journal*, 6(1):53–72, 1997.

[20] J. Nowitzky. Partitionierungstechniken in Datenbanksystemen: Motivation und überblick. *Informatik Spektrum*, 24(6):345–356, December 2001.

[21] SAP. Success Story: Deutsche Telekom AG. `http://www.sap.com/solutions/industry/telecom/customersuccesses.asp`, 1998. SAP, Siemens Nixdorf, and Deutsche Telekom.

[22] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *VLDB Journal*, 7(1):48–66, 1998.

[23] Thomas Schneider. *SAP R/3-Performanceoptimierung*. Addison-Wesley, Reading, MA, USA, 1999.

[24] B. Seeger and P.-Å. Larson. Multi-disk B-trees. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 436–446, Denver, CO, USA, May 1991.

[25] Transaction Processing Performance Council TPC. TPC benchmark C. Standard Specification, Transaction Processing Performance Council (TPC), 1992. `http://www.tpc.org/`.

[26] L. Will, C. Hienger, F. Strassenburg, and R. Himmer. *R/3-Administration*. Addison-Wesley, Reading, MA, USA, 1996.