

Self-tuning of Data Allocation and Storage Management: Advantages and Implications

Andreas Lübcke
Otto-von-Guericke-University Magdeburg
Department of Computer Science
Institute for Technical and Business Information Systems
D-39016 Magdeburg, Germany
P.O. Box 4120
andreas.luebcke@ovgu.de

1 Introduction

Many important business applications use complex database management systems (DBMS). These DBMS have to be administrated and optimized for an optimal performance, especially in time-critical applications. Administration and optimization are very complex and costly tasks. Therefore, researchers and DBMS vendors focus on development of self-tuning techniques for a continuous adaption, e.g., the COMFORT automatic tuning project [6] or the MAPE approach by IBM [2]. However, the optimization and usage of self-tuning techniques for allocation and storage management of data are less investigated. DBMS vendors inform their customers about advantages and disadvantages of optimizing data storage in their user manuals and tuning guidelines. They recommend usage of functionality to optimize data storage with respect to the higher administration costs. But, DBMS vendors do not publish guidelines for this. Optimization of data storage is a complex (high administrative needs) task with many of options and parameters. For instance, the number of parameters for table space creation is huge, e.g., `page size` or database `partition group`. These two parameters affect essentially the performance of data allocation and storage management. In the scope of data allocation and storage management, many parameters dependencies and implications are unobserved. Our approach will observe the affect of the parameters. This paper presents first steps for better solutions and estimations for data allocation and storage management based on the parameter `page size` and its configuration.

2 Approaches for Automatic data allocation and storage management

In this section, we present current approaches from two DBMS vendors for automatic data allocation and to point out their limits. The first approach is from IBM used in DB2 to allocate data dynamically with `system managed spaces (SMS)` and `database managed spaces (DMS)`. Oracle implemented another approach which provides a vertically integrated file system and volume manager.

2.1 IBM's Automatic Storage Approach

IBM's automatic storage approach uses `system managed spaces` and `database managed spaces` [1], which are elementary types of table spaces in DB2. In general, SMS table spaces are better suited for databases with many small tables, because they require less maintenance effort.

Data, indexes and large objects are stored within the same table space. DMS table spaces are recommended for large databases with large tables. A DMS table space consists of one or more container in form of a file or raw device. Thus, data, indexes and large objects can be separated from each other.

The hybrid approach of DB2 prevents administration overhead of DMS table spaces for many small tables, and take advantage of data allocation with DMS table spaces for large tables [7]. The huge benefit of this approach is the lowered administrative costs, because DB2 automatically increases the size of a database [3]. Hence, we do not have to think about creating table spaces (for database size), adding containers or observing table space usage. This option can only be set while creating a database, and cannot be changed afterwards.

While creating a new table space, DB2 performs the decision between SMS and DMS by itself. Regular and large table spaces will be created as DMS table space, because they contain mainly large tables and a huge amount of data. In contrast, user temporary table spaces and system temporary table spaces are created as SMS table space. These two table space types contain mostly small tables. A database administrator (DBA) can not influence this decision.

2.2 Automatic Storage Management by Oracle

Oracle uses another strategy to enhance the performance of data storage called: Automatic Storage Management (ASM). Oracle implements a volume manager for single-instance databases and a file system for application cluster configurations [4]. ASM uses disk groups to store data files. A disk group is a collection of disks which is managed as a unit by ASM. Inside a disk group, ASM provides a file system interface for Oracle database files. The data stored in a disk group is stripped or distributed.

In contrast to IBM, ASM is based on a new type of table space, and independent of **dictionary managed table spaces** (DMT) and **local managed table spaces** (LMT). LMT and DMT use different management of allocating **extents** (group of contiguous free blocks) to a **segment**. DMTs manage the **extents** by using the data dictionary. And each LMT manages its own free and used space by using a bitmap structure. Now, ASM does the **extent** management automatically. Regarding this fact, the administration gets less complex, because some parameters do not have to be considered anymore, e.g., **pctused** or **freelists**.

2.3 Limitations of Existing Approaches

The following considerations will summarize the state of the art for of automatic storage management. We point out the latest improvements, but we will also reveal limitations of existing approaches. An overview of following considerations can be found in Table 1.

The latest approaches of DBMS vendors principally improve the allocation of new space to existing data containers. DBAs do not need to monitor each data container, because DBMS automatically increases the size of data containers. These approaches reduce administration costs, but they also imply new considerations by DBAs. New parameters have been introduced in DB2 and Oracle like **initial size**, which sets the starting size of a table space in DB2. The **initial size** can be set by the database manager (perhaps a standard value), but no algorithms are given to reconstruct these estimations. So, we have to trust the database manager. Furthermore, partition groups

Functionality	Oracle	IBM
Auto resizing	yes	yes
Distribution scheme	manually	manually
Distributed allocation	automatically	automatically
Granularity	coarse	finer
Auto create	no	no
Setting parameters	manually	manually
Complexity	probably equal	increased
Migration	possible	no

Table 1: Summary of key features

have to be created and evaluated to distribute data in DB2. Similarly, a DBA has to take into account disk groups in Oracle. Once these distribution schemes have been created, the DBMS automatically allocates data. The granularity of distribution varies from DBMS to DBMS. In Oracle, the distribution only concerns with whole disks. Partition groups in DB2 follow the same strategy, but data belonging to one table space can be distributed by hand. Therefore, data containers of one table space can be distributed over several disks. In contrast to Oracles approach, several table spaces can be distributed over the same disks.

Nevertheless, DBAs have still to create each data container by hand for a new database. This implies a number of parameters have still to be set manually. In Oracle, some parameters disappeared, and some new parameters are introduced. The complexity may remain equally. In DB2, the complexity even has been increased by introducing new parameters. So, the simplified data allocation involves a more costly administration.

The migration of existing databases is another aspect, we have to consider. Oracle supports a migration to the ASM approach. A coexistence of different allocation approaches is supported, too. In contrast, DB2 does not support any migration. IBM's approach can only be used for the design of new databases. Either the option is set while creating a database or the database cannot use automatic data allocation.

3 Steps to Overcome Complexity of Optimizing Data Storage

This section discusses the need of complexity reduction to implement new algorithms for data allocation and storage management. We will explain our decision for the parameter `page size`. Finally, we introduce our approach using heuristics.

3.1 Research Strategy

Our previous considerations show the high complexity of optimizing data allocation and storage management. The complexity of this process has to be reduced, because we cannot develop an approach for all aspects at once. Due to the complexity of this process, we only consider one aspect for now.

Hence, we have to find a starting point to improve data allocation and storage management. An appropriate starting point should influence performance of data allocation directly. Afterwards, we can examine the dependencies to other aspects. This inductive strategy gives us the possibility to overcome the problem of high complexity. The results will be integrated into our approach. Afterwards, we will repeat this strategy for the different aspects of data allocation and storage management. In this way, we will gain a more general solution.

We decide to optimize the `page size` [5] in the first step. This decision is based on a simple but important reason. The data allocation using pages is one of the fundamental concepts of current relational DBMS. Hence, the `page size` is an important parameter regarding the performance of DBMS at all. Our approach will start to design an adviser for `page size` using heuristics.

3.2 Optimizing Data Storage Using Heuristics

In this part, we will describe the first steps for our approach. We use our knowledge base in database administration to derive heuristics for data allocation and storage management.

There are two problems regarding `page size`. First, the `page size` can be too small. Amplified, a too small `page size` implies probably that only one tuple fits into the page. But even worse, a tuple can be bigger than one page. The administration overhead and resource consumption will be excessive. Second, the `page size` has been chosen too large. Hence,

DBMS have to search in large pages (perhaps for small tuples), and performance (data access) will decrease.

At this point, we have again to reason the granularity. Oracle defines the **page size** for the whole database. We need to find a **page size** which fits all kind of data. In other words, the average cost of data access has to be minimized. In contrast, DB2 provides the opportunity to set the **page size** for each table space. The complexity of decision is increased by this choice, but on demand, the **page size** can be optimized for each single table in our database. As a result, we obtain a more powerful tool to improve data allocation and data access.

We set the **page size** to discrete values, but pages contain tuples which have not discrete values according to their **tuple size**. So, we need to know the **tuple size**, too. We can obtain this information from DBMS. If not, we compute the **tuple size** for an existing database using statistic functions. Given the number of rows and the size of a table, the **tuple size** is computed as:

$$\text{tuple size} = \frac{\text{size of table}}{\text{number of rows}}.$$

Whereby, the size of a table is estimated by DBMS. We need another way to compute the **tuple size**, if the DBMS does not provide the size of table calculation. We estimate the **tuple size** using **page size**, number of pages allocated for this table, and number of rows:

$$\text{tuple size} = \frac{\text{page size} * \text{number of pages}}{\text{number of rows}}$$

This estimation assumes that the pages are filled completely. This imprecise estimation does not work for database design, because we will not have data in a new database. We recommend a function based on number of attributes and their data types. Therefore, we need the length of each attribute ($length(A)$) of table (R). Our first proposal is:

$$\text{tuple size} = \sum_{A \in R} (length(A)) + \text{overhead}.$$

We can improve this estimation using another statistic. If supported, the average length of an attribute ($avg_length(A)$) is obtained from database statistics. We assume a more exact estimation, because we compute it with the real consumed space of an attribute. Therefore, we have only to replace $length(A)$ by $avg_length(A)$ in our algorithm. Again, the adjusted algorithm is only adaptable for existing databases.

A function that maps the **tuple size** to the **page size** is needed, i.e., number of tuples stored in one page. Therefore, the optimal value of tuples per page has to be figured out. Extensive case studies are necessary to evaluate the optimal value. Perhaps, we have to evaluate the optimal value for each DBMS, because internal algorithms and specific implementations will influence the result. In addition, we have to point out thresholds for the **tuple size** regarding the **page size**; because we need to map continuous values (**tuple size**) onto discrete values (**page size**). We assume closed intervals for **tuple sizes** and assign them to each possible discrete value of the **page size**. In this way, we can assign each table to the corresponding table space (optimal **page size**). Therefrom, we can develop a solution for a fine granularity regarding the **page size**.

In case of a coarse granularity regarding the **page size**, e.g., used by Oracle, we need to extend our algorithm. We assume one **page size** for an amount of data or even a database. To compute an optimal value for the **page size**, we have to introduce a weight function. There are two possible approaches. First, we can compute the average **tuple size** for the whole database. The result seems to be a crude estimation. A second approach could compute **tuple sizes** for each table (t). We assume the number of rows of each table as weight to their **tuple size**. The sum of weighted **tuple sizes** is divided by the total number of rows of a database (D). The

estimated **tuple size** is defined as:

$$estimated\ tuple\ size = \sum_{t \in D} \frac{number\ of\ rows(t) * tuple\ size(t)}{total\ number\ of\ rows(D)}$$

This algorithm only works for existing databases. Furthermore, the algorithm estimates the access rate of data is evenly distributed.

Both algorithms for **page size** estimation mostly based on the same functions. So, we can develop one framework with two different aspects regarding the granularity of **page size**.

4 Conclusion

This paper points out the state of the art of automatic data allocation and storage management in current DBMS. We discussed the limitations of current approaches and considered the need of complexity reduction. We argued that the **page size** is an appropriate starting point for the future steps. Finally, we used our knowledge base to discover heuristics for **page size** optimization. Our first step was the development of algorithms to estimate the optimal **page size** for a table or a whole database.

We will perform extensive case studies to derive a function which computes the optimal number of tuples per page. To develop a useful adviser for **page size** configuration, we have to extend our approach. First, we will improve our estimations of **tuple size** (including the *estimated tuple size*) for new databases. Second, we integrate the workload into the **page size** estimation for a whole database.

References

- [1] W.-J. Chen, A. Fisher, A. Lalla, A. D. McLauchlan, and D. Agnew. *Database Partitioning, Table Partitioning, and MDC for DB2 9*, volume First Edition. IBM Redbooks, 2007. Draft Document June 18,2007.
- [2] IBM. An architectural blueprint for autonomic computing. White Paper, June 2006. Fourth Edition, IBM Corporation.
- [3] IBM. Introducing db2 9, part 4: Automatic and other enhancements in db2 9. Technical Article, June 2006. IBM Corporation, <http://www.ibm.com/developerworks/data/library/techarticle/dm-0606ahuja2/index.html>.
- [4] Oracle. Automatic storage management: Technical overview. Technical White Paper, November 2003. Oracle Corporation.
- [5] P. Rob and C. Coronel. *Database Systems: Design, Implementation, and Management*, volume Seventh Edition. Course Technology.
- [6] G. Weikum, C. Hasse, A. Moenkeberg, and P. Zabback. The COMFORT Automatic Tuning Project, Invited Project Review. *Information Systems*, 19(5):381–432, 1994.
- [7] P. Zikopolous, G. Baklarz, and L. Katsnelson. *IBM DB2 Version 9 New Features*. McGraw-Hill Osborne, 2007. Page 75 et seq.