A Relational Database Performance Considerations for Numerical Simulation Backend Storage

Jacek Nazdrowicz

***Abstract*—Problems of effective numerical calculations in scientific applications are caused by non-optimal front-end application code or ineffective system of scientific data management at back-end. In this article author presents some aspects of performance problems when relational database is used as backend storage based on real cases. Presented examples come from simulated environments with large load and many queries executed. These examples can reflect real problems with data processing in numerical calculations on data extracted from MSSQL Server database.**

***Index Terms*—flat file, relational database, numerical methods, data mangement, performance.**

I. Introduction

IN computing systems, including numerical applications, now it attaches great importance to the efficient storage system and access to data. As the computing power of today's hardware grows, mathematical and physical models are becoming more complex which naturally increases the scale calculations and dramatically increase the amount of data. While the data space is not a problem, access to them in a very short acceptable time is often a big challenge. This problem is directly translated into the total time and the end result calculation or simulation. This paper focuses on efficient access to data and its storage. Currently applications (systems) are layered (multi-tiering). In the simplest configuration, there is a numerical application layer, which is focused all its logic (here are the main calculations performed) and the data layer (used by the application logic). One can say that it is the logic of the data - the data layer is rather passive.

Numerical application for scientific objectives calculations require huge and effective system for data storage. Nowadays there are many technologies and techniques for storing data. Hardware appliances with appropriate resource share protocols implemented and specialized software giving us possibility to create appropriate and fast response data storage system. Most of contemporary numerical applications use flat files for storing data which contain records with unstructured relationships. These files can be saved in disk drives located locally in application server or remotely in arrays.

To get data quickly one has to allocate memory and processors, which effectively manage databases and satisfy scalability. Contemporary numerical applications flat file-based often met the problem of constrained scalability what impacts directly on functionality and possibility to solve complex physical problems. Relational database is very good solution because it manages data very effectively [1].

In some publications one can find such practical applications. In the paper [2] there are SQL-compliant databases presented and benefits of using them in FEM (Finite Elements Method) applications. In Microsoft report [3] implementation SQL Server with FEM front-end applications are presented. One can find some useable database engine features, which may be used in calculation optimization and additionally examples of tables database structure, indexes, integration services. The Database utilization also appeared as a part of solution for online data access (through Internet) system for a Finite Element Analysis (FEA) [4]. The objective of such system was to design mechanism to access the results from readily accessible data sources for further manipulation across network in other location, applications and many scientists. This article continues consideration of application database engine as a backend storage for numerical calculation presented in [9].

The use of appropriate techniques and IT equipment for the purposes of the calculation requires the determination of how data is to be stored, which is to be accessed and what is the most important - what will be the performance of such a system. Among the contemporary methods of data access and storage clearly distinguish two: first - data flat file that application directly access reading from and writing data to it and second - relational database as a more complex structure.

Thus, in the construction of such a system should include:
- whether it will be a lot of data reads,
- whether it will be a lot of records,
- what will be the size of the data,
- whether the data will be mixed (e.g . text , numerical),
- whether the data will be LOB (Large Objects e.g. XML),
- economics solutions.

The number of readings data records determines the manner of data storage implementation. It is known that hard drives are much slower than RAM, so it is important skilful use of available opportunities.

II. Two Types of Backend Storage

A flat file stores data in plain text format and serve as a bare means of storing table information. Flat file does not hold any relations between data. Difference between flat file and relational database is meaningful – flat file is processed entirely, while relational database gives capability to access single record.

J. Nazdrowicz is with the Department of Microelectronics and Computer Science, Lodz University of Technology, ul. Wolczanska 22/223, 90-924 Lodz, Poland (email: jnazdrowicz@dmcs.pl)

Many applications are created using flat files at the backend because of their simple structure; additional big advantage is that their consume much less space than relational database structured files; disadvantage is that the information can only be read or write. Data representation in a flat files used as a backend in applications meets certain standards. In simplest form every column of file is specific data type. Particular data separation can be achieved using various delimiters, included in flat files to ensure fixed-width data formatting. They can reduce the overhead of locating different fields in a row of file.

Relational database application will largely depend on the nature and purpose of the data, access requirements, and other applications rules which need to access database. Flat-file databases are more simple but limit data access allowing for manual process and structured programs operation. Relational databases are more complex but provide advanced capabilities and more efficient access to required data located in file. A huge advantage is possibility to use hierarchical order of data (B-trees, heaps), indexing and optimizing access to data with queries from RDBMS.

Relational database (RDB) is a collection of related tables. Each table is a physical representation of object that consists of columns and rows. Columns are the fields of a record or the attributes of an object. The rows contain the values or data, called records. Relationships can exist among the columns within a table and among many tables.

Nowadays relational model of data is much more popular than before; although numerical applications were linked to databases long time ago [5], now this solution has greater importance. One of the driving factor behind that is growing volume of data to process, other one – considerably growth and development relational databases and tools for managing.

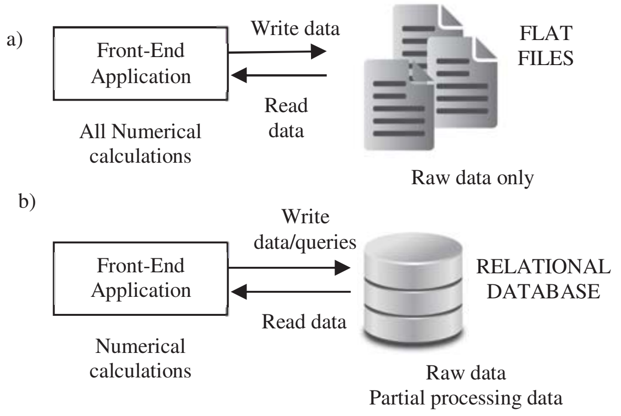


Fig. 1. Processing data with (a) flat file, (b) relational database backend storage.

In Fig. 1. there are presented differences between flat file and relational database backend storage for scientific applications.

The use of relational databases (including SQL Server) carries with it many benefits, because user - application designer for numerical calculation has a much greater impact on data storage and access to data. This, however, carries with it certain consequences. In the article [9] author presented the possibility of using the database engine SQL server for storage and provision of data for the application. The complexity of such a solution makes it necessary in designing to pay attention to many aspects of the nature of programming that can significantly affect the efficiency of use and performance. Although in the majority of cases the SQL Server can be used in such applications in the basic range, without launching a number of additional features, however incompetent and non-optimal referring to store in the database may expose the author's solution to many unpleasant surprises and a significant time extension of the calculation.

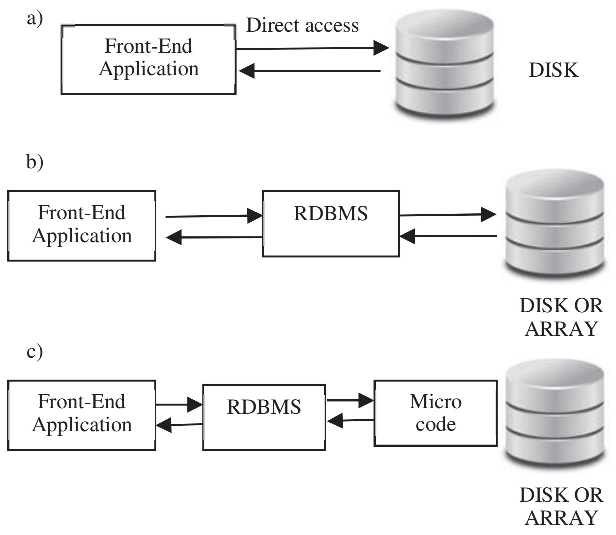


Fig. 2. Data storage system for application: (a) direct access to disk, (b) with RDBMS, (c) with Microcode of disk array.

Fig. 2 presents the idea of the use of available components to build a storage system for the more general application. In all cases, we have as a frontend application (regardless of the development environment in which it originated) and a data storage system (backend). Data storage system can be implemented in two main ways: as a direct read/write to disk (access block – fig. 2a) or indirect management software access to data (e.g., relational database – fig. 2b, c - Oracle or MS SQL Server). Fig. 2c can be seen as the most advanced version of solution: between RDBMS and disk/hard drive there is an additional element in the form of a module containing microcode managing multiple access to the drive and organizing drives in the so-called RAID (Redundant Array of Inexpensive Disks) groups. In any case, (a, b, c), the data is written to the file, however, much different way of their organization. In the case of 2a the data are written to the flat files, the other - to file relational databases.

While comparing the two methods of accessing data at a glance, it appears that direct file access is faster, because there is no additional software between the application and the disk subsystem, however this is not always true. It mainly depends on:

- programming skills,
- good knowledge of the architecture for storing data,
- excellent knowledge of implemented algorithm and analytical skills for potential adverse events that may occur during program execution.

RDBMS greatly relieves the developer of having such knowledge, since many conflicts and adverse situations that may arise while accessing data it solves itself and there are transparent for programmer/user. There are however (and this is not uncommon) situations in which the system solves the problem, but thanks non-optimal code, the application executes very slowly (which, of course, it could be avoided).

Communication with the database application is done using the extended SQL query language called T-SQL (SQL transaction). This extension allows you to create loops, conditional statements and variables. Additionally, it allows you to create triggers, stored procedures and functions at the level of the database itself. These features give the user additional opportunities to control the behavior of database and programming capabilities extend much that can be successfully used in numerical applications and solve many performance problems.

III. Relational Database Based Data Factors of Performance. An Overview

In designing the system code database layer, therefore, one should pay special attention to the following areas that may cause performance problems which affects whole numerical calculation system:

- the physical disk subsystem,
- incorrect assessment of the requirements cache for RDBMS,
- optimal data storage,
- indexing strategies, execution plans and statistics in the database,
- data paging,
- competition for access to resources database (mechanisms to block access to data),
- competition for resources and the expectation of access (latch).

It should be remembered that these problems to a large number of cases can coexist and depend on each other, and therefore the problems of databases performance are often multi-dimensional problems and requires taking into consideration the correlation between them.

A. *Physical disk subsystem*

The physical disk subsystem is the part that directly affects the quality of the whole system performance. Any performance issues at this level directly affect the performance problems at the level of the RDBMS. The main performance parameter of disk subsystem is the number of operations per second (i.e. IOPS - Input Output operations per second). The need for adequate disk subsystem should already be identified in the initial phase of building the system. Why is this so important? Currently you can choose storage as a local drives or arrays. First option is cheaper, however, increase the efficiency can be made only by changing the type of disc so this choice is very limited. If you additionally will need enough space for the data, the problem may be quite serious. A better solution is to select a disk array in which disks can be aggregated and grouped appropriately and fairly easily scale the size of volumes (Fig. 3).

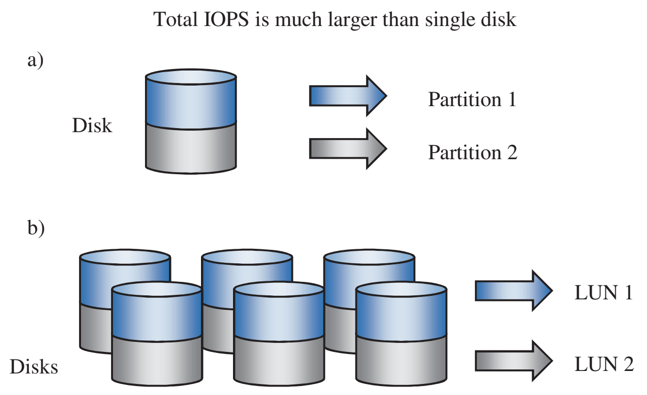


Fig. 3. Data storage system for application a) local disk, b) disk array.

It is worth mentioning how disk space is available for local drives, and as in the case of disk array (fig. 3a and b). In case of local disk logical disks are called Partitions, in case of arrays – LUN (Logical Unit Number). Difference is meaningful – flexibility of IOPS and size expanding.

Performance testing is often done empirically and makes changes in the course of performance analysis (hence the use of arrays is more efficient). During the analysis, it is necessary to take into account the parameters of physical disks and available partitions / LUNs on the side of the operating system (because RDBMS works under its control). Here it is important to pay attention to the limits that for a given parameter and the area is necessary.

The disk subsystem performance analysis should first of all take into account the following parameters:

- the number of operations per second,
- time to read and write data,
- the queue to disk / volume.

The number of operations per second tells what is the real value of this parameter in a production environment. This parameter, of course, does not exceed the limit value derived for the specific system disks. The observation of this size allows us to evaluate whether the use of the subsystem by the application numerical approaches the technological frontier. Solving the problem is by setting up RAID group on the right amount of physical disks. Knowing what technological limitation IOPS has a single disk, one can form composite volumes and increase this limit (generally the total IOPS is the sum of the individual disks IOPS).

Time to read and write data - this parameter results directly from disk configuration and the competition for access to volumes by many processes (not only SQL server). In the case of volumes located on a local drive both the operating system and RDBMS use them, resulting competition for access time for each file. Even if the operating system files and SQL Server are located on separate logical volumes but on the same physical disk, problem still exists and total IOPS does not extend (in the case of SAS drive about 210 IOPS). A better solution is to deploy operating system and database files on separate physical disks (each of the disks has 210 IOPS) – a profit is therefore twofold. The situation is similar in the case of arrays, except that a large number of disks allows for a much improved performance.

Another issue is the length of the block recorded on the disc. It is known that the read / write operations (IOs) concerns a single block, so it is natural that the extension of the block of data stored significantly affects the transfer of data from/to the application in a proportionate manner.

Queue to disk – means how many IOPS waits for disk service. Large value means latency of data operation. This parameter influences performance of RDBMS meaningfully and results from technological limitations.

B. *RDBMS cache*

Buffer cache for SQL Server directly impacts on performance because all disk operations pass through it. Buffer memory cache is located in the memory of the server therefore at the stage of the project, the system must equip it with the right size. How much can you allocate memory for an instance of SQL Server? Good practice says, that a maximum of 80% of the total available memory. How much is actually necessary? It depends on the nature of the operations performed on the database. If there are domination a large variability read pages of data, the buffer should be relatively larger to accommodate them in it.

It is worth mentioning about the mechanism would occur when reading data from the database. Read the data side of the disc is stored in memory in the buffer manager. This reading is referred to as physical. Re-reading this data is not from physical disk but from buffer cache – it is definitely faster. This is called logical reading (fig. 4). Of course pages in the buffer cache does not exist forever, the oldest are preempted in favor of these new, if space runs out. This means that physical reading from disk again will take place, which takes more time. A large number of readings can consequently affect page lifetime if the buffer cache is not large enough.

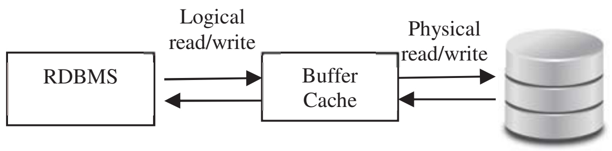


Fig. 4. Read and write data through buffer cache.

C. *Optimal data storage*

In modern relational databases data are stored in minimum one data file (e.g. in MS SQL this is file with .mdf or .ndf extension). Sometimes it is worth in very loaded calculations to split data tables between few data files (called partitioning). Having efficient disk subsystem we can split data files with partitioning tables among many physical disk or RAID to get more performance than locate one or many files on one physical disk only (fig. 5). The best results are in reading data or writing by many processes.

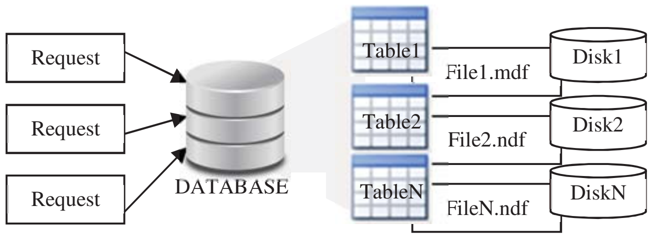


Fig. 5. Partitioning tables among database files and physical disks.

The Fig. 5. shows maximum optimized table deployment – in real world we do not have so much possibility. Disk quantity is less than tables and one should optimally tables deploy taking into account their load. If all data are on one file, risk of performance degradation is much more because all requests are directed to one physical disk, and total IOPS must be split on many processes. Thus, in that case disk subsystem must be more efficient.

D. *Data paging*

We are dealing here with two important parameters: the number pages moved per second and Page Life Expectancy. Paging refers to move pages from disk to buffer cache and vice versa (for saving data in the checkpoint). A large number pages per second parameter testifies to the fact that very often the pages are transferred to the buffer cache and be released at the same time and takes over old pages place. If these two things are happening very intensively at short intervals – it slows down the system meaningfully. The second parameter confirming these problems is Page Life Expectancy, which shows the life of the page in buffer manager. A high value of this parameter indicates that the data pages are not often exchanged in the buffer cache, and the application that wants to read the data does not have to reach into slower disks (read physical) but high-speed memory buffer (read logic) only. The low value of the parameter in turn testifies to the fact that there is such a large scatter read pages that do not fit all in the buffer and need to be replaced frequently. It also gives a signal that the buffer should be expanded.

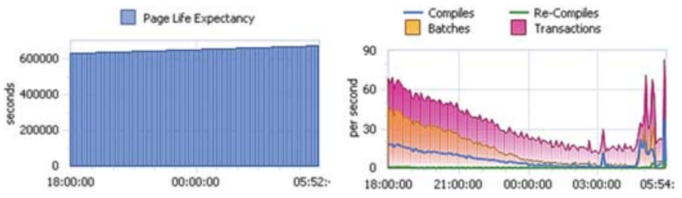


Fig. 6. Dependancy between Page Life Expectancy and number of transactions.

As one can see on fig.6. paging strongly depends on number of transactions and T-SQL compiles[*)]. Generally, increase time of life page in buffer is inversely proportional to transaction amount, complies and executed batches. For many calculations the good practice is to fit buffer cache appropriate to needs, do calculations in small transactions (to free space in buffer cache), optimizing the query to shorten its duration time, split transaction into smaller transactions, avoid query concurrencies. Another issue one has to take into consideration is to optimize indexes to avoid whole tables scan (because large object will be placed in buffer manager and its size can exceeded). Sometimes additionally queries has to be recompiled (this often happens in case of parametrized procedures or functions) what it will take a time. This will enforce many physical data IOPS and slow down calculations. The appropriate index strategy must be then implemented and statistics must be refreshed.

[*)] All presented results were extracted from real database bulk On-Line Transaction Processing type environment managed by author

E. *Competition for access to resources database (mechanisms of locking access to data)*

The mechanism of placing locks in the database is a very important function because it does not allow to process a single cell or row of data derived from two different processes/threads. Therefore ensures data consistency. Locking records can be made on several levels: field, line, record or the entire table. This is what type of lock is fitted depends primarily on the specifics of the application and its needs at any given moment. RDBMS often also implies an intent lock, through which already reserves the right to set the lock before you actually will need to indicate that you will want soon to make changes in the facility. In a situation where the process will try to access read or write to the object on which another process has already started blocking, it will have to wait until lock is released. It must be emphasized that if some processes need to get to a specific record field, and the lock is attached to the whole record or table (wider range), of course, that record will also be available.

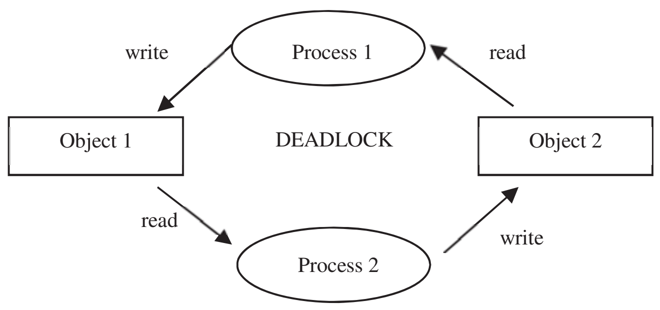


Fig. 7. Deadlock mechanism.

In the case of numerical applications, where a large number of reads and writes to the database will take place must reckon with the mass occurrence of blockages. Minimization can only take place when the application will try to avoid conflicts during reading data (though often probably cannot be done). As a general rule and good practice at writing code is to avoid blocking broader range of objects than it is needed. Why? In a situation where process 1 calculates and wants to get data from two or more fields in the record, and before another process 2 blocked the entire record or worse table although solicit from a completely different field, unfortunately the first trial and so will have to wait. This of course can significantly slow down the calculations made by the application.

The least desirable effect setting up blockades is called deadlock, which for intensive applications with writing and reading can occur. Such interlocking objects takes access to the data by two or more processes. When such a situation occurs? Then, when you want to save the cross processes in one object and read in another, the second process performs a similar operation on the same premises - fig. 7 [10].
Unfortunately in such case there is necessary manually solve the problem by selecting one process to kill. Another problem is when lock chain appears. There are many processes locked by previous one, and there is one (so called head blocker) on which rest processes must wait to finish.

F. *Competition for hardware resources and the expectation of access (latch)*

This mechanism operates similarly to the lock mechanism, with the proviso that relates to server resources. Latch appear when there is a need for an application to gain access to the resource, while access is currently impossible. For obvious reasons, the process of trying to get a resource has to wait, which, of course, introduces a delay.

The example of fig. 8 shows how strong is the latches influence on the response time of the server. One can see clear correlation between the two characteristics over time. Appearing peak - lock latch type (fig. 8b) - automatically causes the appearance of a long response time of the server which in turn has a negative impact on data processing applications - extending the data processing.

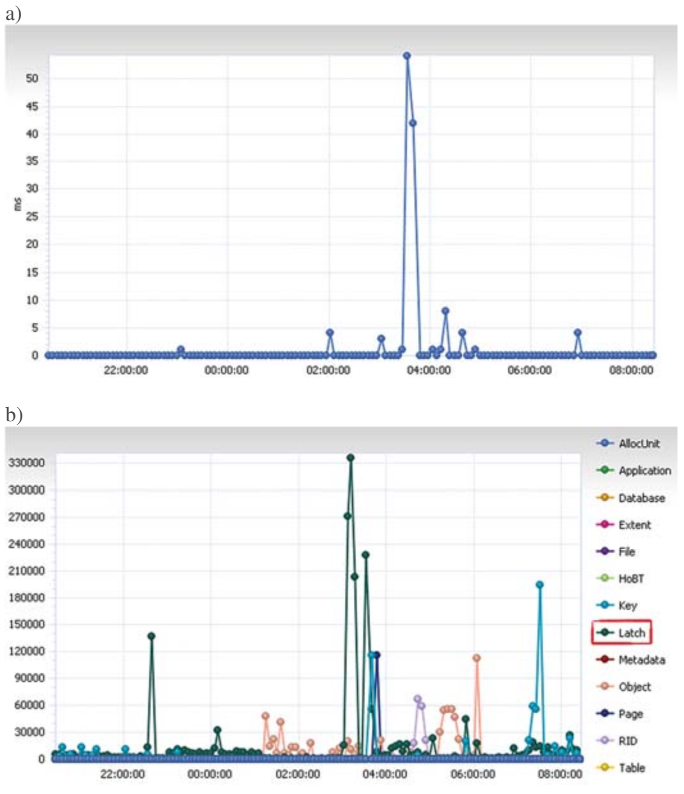


Fig. 8. Dependency between response time a) and latches amount b).

The following fig. 9 shows the degradation of the performance of the server, which results directly from imposed on a large number of transactions (counting). There are shown a dramatic increase in the number of disk operations (IO), which is maintained for several hours. You can see here that the number of locks also remain relatively high during much of this time. The plot, of course, still tell us much, but in many cases you may notice some correlations between the different categories (e.g. number of IOs and perform backup). Full analysis of the problem thus requires a more comprehensive approach and a broader view of performance monitoring data.

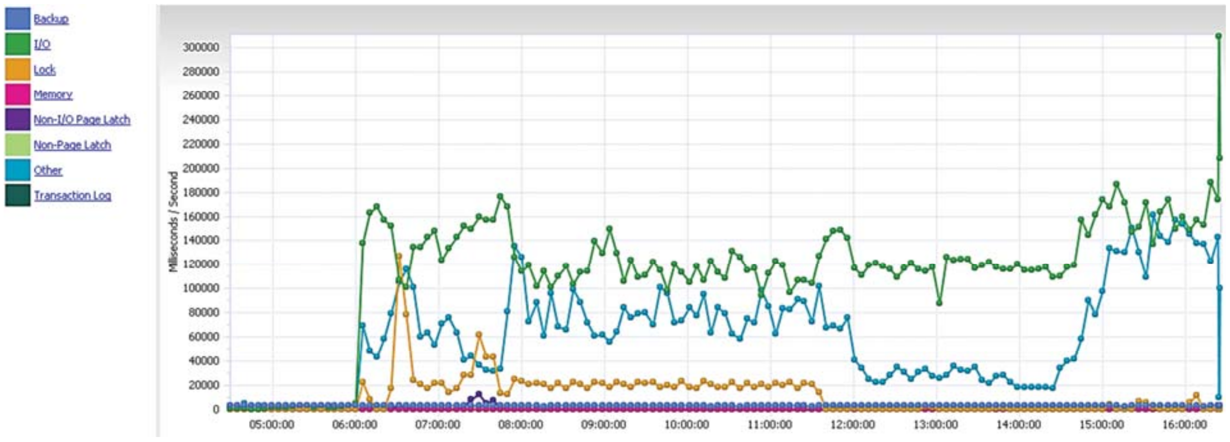


Fig. 9. SQL Server wait times caused by some RDBMS areas.

Much more information is obtained from the graphs (fig 10a and b) – these graphs are referred to large volume data processing with many I/Os operations – typical of FEM applications. It can be seen primarily that a large number of reads and writes take place. One can see that the number of readings per second (oscillates around the number 3500) it dominates the number of writes per second (it fluctuates around 100). Of course, such situations can occur normally in the event of a large number of transaction processing operations, but their influence on degradation of the system may be significant if it is directly concerned the disk subsystem. This case can see in fig. 10a, which represents the physical IOPS of SQL Server database, which are physical disk operations. One can see that the characteristic 10a and 10b are very similar which clearly indicates that properly the number of IOPS operations performed on the disk. It is very disadvantageous particularly in the case of a large number of transactions - numerical operations, because it causes significant delays and slowing down numerical application. These delays are a consequence of the emergence of latches, which in a sense is the equivalent of a locks on the same premises relational databases.

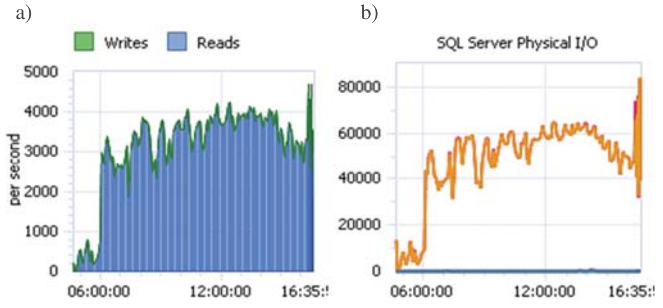


Fig. 10. Read/write times per second a), SQL Server Physical IOPS b).

Another issue is the number of operations per second that can be read from this graph. It oscillates around the number 50000-60000 IOPS which of course in case of single disk is not to achieve (a single SAS disk reaches about 210IOps). Therefore, we attempt to use the database for complex numerical calculations (in particular FEM) to investigate empirically the upper limit of IOPS that will be achieved for a specific analysis of the numerical model. Such information may prove to be decisive for the shape of the future system for calculation of course including the use of local drives/arrays and single disk/disk groups.

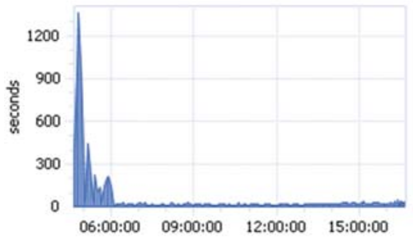


Fig. 11. Page Life Expectancy.

Another figure (fig. 11) shows the lifetime pages in memory SQL Server (buffer cache). It is seen that when data are read from disk (fig.10) page life shortens dramatically and many pages are thrown out from buffer cache and new ones are coming. This is the worst scenario can appeared in case of numerical application, because bulk physical data operation can happen, and database response to application will decrease.

G. *Indexing strategies*

An index is a structure associated with a table that take part in query execution[6][7]; it contains keys built from one or more columns in the table. Indexes are stored in a commonly known B-tree structure for speeding up to find rows associated with the key values. It is especially useful in case of searching for data. Indexing data is complex operation, taking a time for appropriate creation but in effect front-end calculation applications works more effective. It is especially important in algorithms with many recurrencies and complicated equations to solve where data are exchanged with storage backend many times. Of course there is one important disadvantage of this solution - indexes take additional storage space. Indexes are very closely connected to statistics and as they changed – indexes must be refreshed, too.

Indexes are very useful, making possible to use in data storage access during numerical FEM calculations. Tests were performed for 17000 and 34000 elements. Time duration was measured with SET STATISTICS TIME ON option. Results are shown on fig. 12 and 13 where one can see and compare execution time of solution with and without index applied.

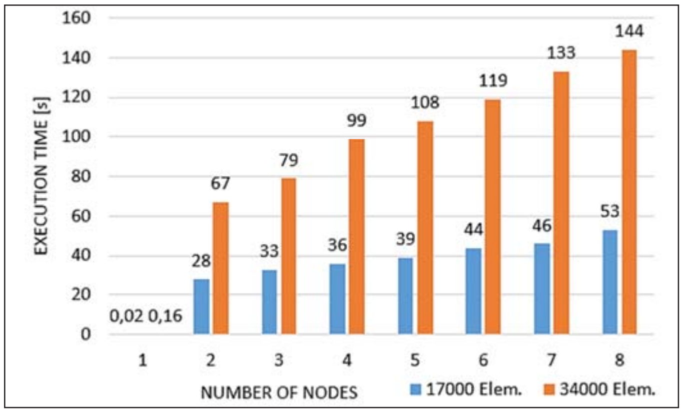


Fig. 12. Execution time of selecting nodes coordinates for various number of elements without index application.

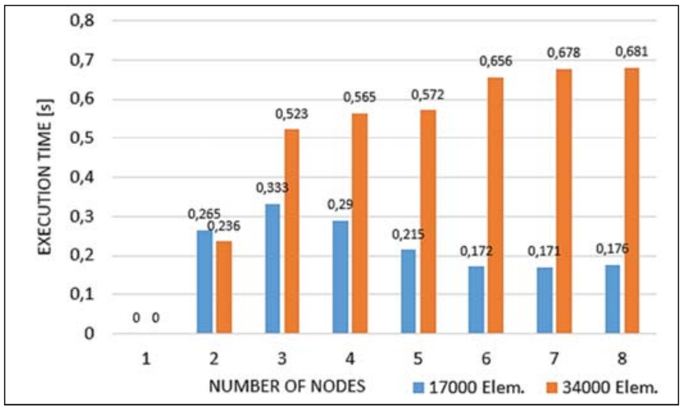
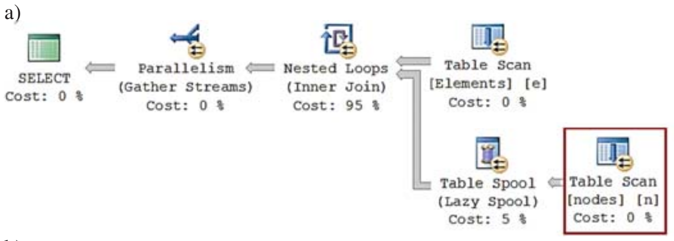


Fig. 13. Execution time of selecting nodes coordinates for various number of elements with index application.

Execution plan calculates total cost of query execution. System of selecting execution plans makes decision which plan is the most optimized. During that it takes into consideration available heap (unstructured data), indexes structure and statistics. On fig. 14 one can see the same query extracting numerical data in two ways: without and with index. Red frame points at clue block.

a)



b)

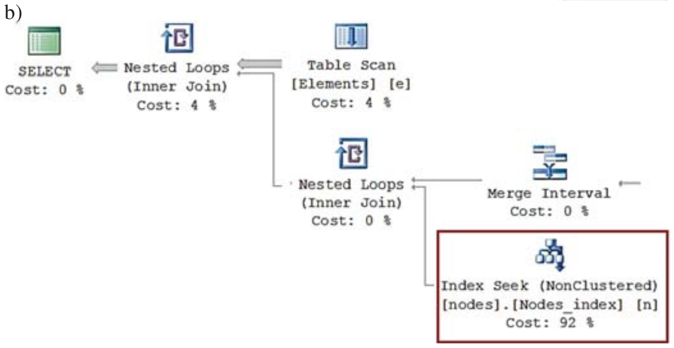
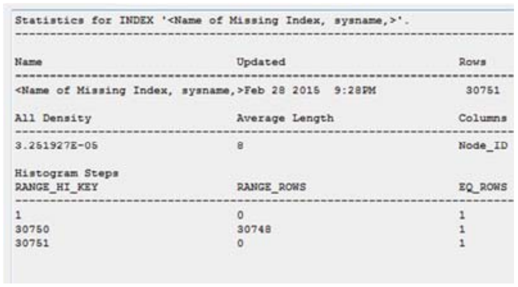


Fig. 14. Example of execution plan query extracting joined 2 tables FEM data from database (a) non optimized query, (b) optimized query after hint index applied.

When index is lack whole table is scanned (all data in table are processed), when appropriate index is applied only necessary data are sought and then selected for further processing. Test carried out FEM data with execution data extracting showed that non-optimized query last 28 seconds (fig. 12), whereas after index applied – fraction of the second (fig. 13). Significant differences between these solutions are marked with red frame in fig. 14a) and 14b), where a comparison of volume of data processed is shown. In the case of table scan, tables placed in query (with join clause too), are bulk read record by record (and saved in buffer cache). Because as we mentioned before, all data passes through buffer cache, it can fill up very fast. Next scans will generate physical IOs of the same table because of fast changes in buffer, in effect will increase time of execution and server waits will appear.

Another problem we can meet during execution of queries is unnecessary thread parallelism which sometimes do not accelerate response to numerical application. It happens in case of strong dependency of threads with others especially in resources. In such case one thread cannot continue because it must wait for resources used by other thread. Then user must empirically appropriate adjust value of Cost of Parallelism.

Indexing strategies are widely explained in many publications related to RDBMS tuning and optimization [8]. We have to notice that optimization is related either to indexes or to query form itself.


```
Statistics for INDEX '<Name of Missing Index, sysname,>'.
--------------------------------------------------------------------------------

Name                                Updated                              Rows
--------------------------------------------------------------------------------
<Name of Missing Index, sysname,>Feb 28 2015  9:28PM                    30751

All Density                         Average Length                       Columns
--------------------------------------------------------------------------------
3.251927E-05                        8                                    Node_ID

Histogram Steps
RANGE_HI_KEY                        RANGE_ROWS                           EQ_ROWS
--------------------------------------------------------------------------------
1                                   0                                    1
30750                               30748                                1
30751                               0                                    1
```

Fig. 15. Sample distribution statistics.

An important element during such optimization process are statistics, which are sampled map of data distribution in a table (fig. 15). They help query optimizer to create the best strategy for execution all the physical operations of joining, sorting, ordering and selecting, updating, deleting the data. With time however, statistics become out-of-date because of data updates. In such situation statistics no longer reflect data distribution in table (after many updates) and in consequence queries execution time can suddenly grow. User should refresh and rebuild statistics manually then. Sometimes when there are many changes on database in unit time it good practice is to rebuild or reorganize indexes and statistics automatically [13] when data processing is minimized.

Indexes and statistics must be refreshed periodically if they data are updated. Without it indexes are not effective and often can impact negatively on query execution time.

IV. Conclusions

The main objective of this article was to bring closer problems with using relational database-based storage for bulk data processing (calculations) and point on many areas problems can occur during operation (numerical calculations). All these problems are performance nature and are multidimensional i.e. they can depend on others. Application RDBMS as a backend for numerical calculations meet the same problems like in other cases (e.g. business OLTP- online transactional processing) because it operates also with the same rules concerning queries optimization and SQL server system components. Many business cases are based on numerical methods, they also operate on large volume of data that is why performance problem will be similarly.

Although RDBMS application as a backend storage is encouraged, person who wants to implement must know disadvantage of such solution. To get the best performance database must be continuously monitored and all structures must be refreshed. Statistics changes can imply needs for recreating indexes for better response time.

Knowledge about administering relational database engine, its functionality and performance problems has potentially large impact on future work and research. Good architecture for scientific data management needs to take them all into consideration.

References

[1] J. Gray, et al.: “Scientific Data Management in the Coming Decade” Microsoft Research Technical Report MSR-TR- 2005-10, 2005, available at: http://arxiv.org/ftp/cs/papers/0502/0502008.pdf

[2] F. E. Karaoulanis, C.G. Panagiotopoulos, E.A. Paraskevopoulos, “Recent developments in Finite Element programming”, First South-East European Conference on Computational Mechanics, SEECCM-06, Kragujevac, Serbia and Montenegro, June 28-30, 2006.

[3] G. Heber, J. Gray, “Supporting Finite Element Analysis with a Relational Database Backend”, Technical Report MSR-TR-2005-49, April 2005, available at: http://research.microsoft.com/apps/pubs/default.aspx?id= 64535.

[4] J. Peng, D. Liu, K. H. Law, “An Online Data Access System for a Finite Element Program”, http://eig.stanford.edu/publications/jun_peng/ data_access_system.doc.

[5] R. I. Mackie, “Using Objects to Handle Complexity in Finite Element Software”, Engineering with Computers, 13(2), 1997, pp 99-111.

[6] P. Gulutzan, T. Pelzer, “SQL Performance Tuning”. Addison-Wesley Professional, Boston, 2003.

[7] S. Dam, G. Fritchey, “SQL Server 2008 Query Performance Tuning Distilled”, Apress, New York, 2009.

[8] K. Delaney, “Inside Microsoft SQL Server 2005”, The storage engine, Microsoft Press, Redmond 2007.

[9] J. Nazdrowicz, “A Relational Database Environment for Numerical Simulation Backend Storage”, Proceedings of the 22nd International Conference "Mixed Design of Integrated Circuits and Systems", June 25-27, 2015, Torun, Poland.

[10] K. Delaney, “Inside Microsoft SQL Server 2005. The Storage Engine.” Microsoft Press, Redmond 2007, pp. 375-380.

Jacek Nazdrowicz was born in Poddębice, Poland, in 1975. He received the MSc degrees in Technical Physics, Computer Sciences and Marketing and Management from the Lodz University of Technology, Poland, in 1999, 2000 and 2001 respectively and the PhD degree in Economics Sciences, Management discipline, in Lodz University of Technology, in 2013.
From 2014 he attends doctoral study in Lodz University of Technology, electronics discipline. His research interests include modelling and simulation MEMS devices and their application in medicine. He also participates in EduMEMS project (Developing Multidomain MEMS Models for Educational Purposes).
Since 2007 he also works in mBank as a System Engineer of SQL Server databases. He has the following certifications: MCSA Windows 2012, MS SQL Server 2012 and Storage Area Network (SAN) Specialist.