Extracting and Updating Data Performance During Bulk Numerical Calculations in Databases

Jacek Nazdrowicz

***Abstract*—Nowadays, more complex engineering problems need to carry out increasingly complex numerical calculations. In order to obtain the results as soon as possible, engineers must simplify the physical model on the one hand, apply appropriately efficient infrastructure to carry out calculations on the other. This infrastructure includes both hardware and software. Problems in numerical calculations in scientific applications are often caused by non-optimal front-end application code implementation or by 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. Of course, the biggest problem in numerical computation processing is that the large amount of data stored on the storage area can slow down the entire computing system and in turn directly affects the computational efficiency. Presented examples come from simulated OLTP (Online Transactional Processing) 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 with large storage system connected.**

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

I. Introduction

THE problems of modern computing are mainly processing and storage of huge amounts of data. Computer science, which dominates in many areas has become a powerful tool for solving complex engineering problems using numerical methods.

Scientific numerical calculations require very effective, high-performed system for data storage processing [1]. Currently, we have many solutions and technologies delivered by many vendors for storing data. Hardware appliances with appropriate resource, share protocols implemented, and software processing data give us possibility to create well-optimized, fast response data storage system.

Proper selection of the components making up the backend storage is extremely important, because it directly affects the performance of the entire system. This requires not only a broad theoretical knowledge, but also the experience gained from long-term follow-up characteristics of the performance of individual components and the ability of correlation results. Database is the most known component for storing processing data. In some publications one can find its practical applications [2][3][4]. 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 of SQL Server with FEM front-end applications are shown. 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.

J. Nazdrowicz is with the Department of Microelectronics and Computer Science, Lodz University of Technology, Lodz, Poland (e-mail: jnazdrowicz@dmcs.pl)

In the basic systems, we can identify the application layer and the storage layer. This is of course a very general division. Approaching the matter in more detail, we can see that the system can consist of network and data storage layer which in turn consists of the database engine and data recording system i.e. physical disks. Between the database and disks there can be another layer network infrastructure for data transmission. This is illustrated in Fig. 1.

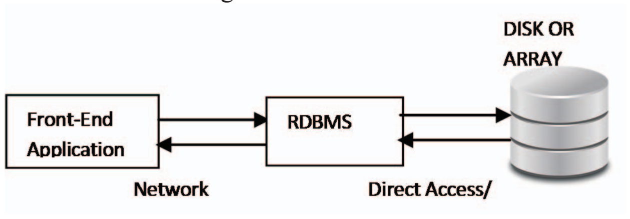


Fig. 1. Data processing system with front-end and back-end components.

Both the application and the database engine are appropriate software, so in heterogeneous environments, the most common approach is the separation of functions between different physical or virtual servers. Obviously, with a large number of servers, the most economical solution seems to be the virtualization of servers, but note that it may take place at the expense of performance because the so-called hypervisor distributes it among the virtual servers. The division between the physical machines is the most natural while ensuring appropriate efficient communication channel between servers.

Separating functionality on servers on the one hand allows to obtain better performance, on the other hand increases the administrative effort, the cost of infrastructure and maintenance. Looking for the most efficient infrastructure to maintain computing environments, particular attention must be paid to both the functionality of solutions as well as its economy.

In numerical applications, it attaches great importance to the efficient storage system and access to data. Since the computing power of today's hardware grows very fast with next generations, mathematical and physical models are becoming more complex which dramatically increases the calculations scale and the amount of data. While the data spaces are not a huge problem, access to them in a very short acceptable time is often a big challenge. This problem is directly related to the total time and the end results of calculation or simulation.

The use of appropriate techniques and computation equipment requires the determination of how data is to be stored, which is to be accessed and what will be the performance of such a system. Among the contemporary methods of data accessing and storing there are two the most important: first - data flat file (application directly reads from and writes data to file) and second - relational database as a more complex structure. The first one was widely described in [9]. Here, it is taken into consideration second one solution (relational database), because this allows to store data in structured manner and offers more possibilities to optimize data access.

To create complex heterogeneous numerical calculation system it is important to determine:

- whether it will be a lot of data reads,
- whether it will be a lot of data writes,
- 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).

The use of database engine such SQL Server brings up many benefits - application developer implementing numerical calculation has a much greater impact on way of data storing and accessing. Obviously, it has certain consequences. In the article [9] author showed the possibility of using the database engine SQL server for storage and provision of data for the numerical calculation application. The complexity of such a solution makes it necessary on designing stage to pay attention to many aspects of the nature of programming that can significantly affect the efficiency of use and performance.

II. Data Processing in OLTP Databases

The use of OLTP database systems in numerical calculations allows for a more efficient search, processing and updating data. Data stored in a structured manner allows for faster access to them through the use of internal mechanisms and database structures based on b-trees (indexes). Another issue is the equipment itself for data store and the mechanisms for extracting and updating data. Fast storage space for data is still very expensive, and in the case of numerical data for the simulation of complex numerical models, the amount of them can be quite substantial. That is why the way of access and store structures on OLTP database side are so important.

It is essential in order to design a system containing both of two components: the database system and storage space optimize comprehensively. This is important from the point of view of database software vendors who implement features those directly affect the utilization of space and utilize their specific characteristics and properties.

Here, database and storage space performance aspects will be discussed and after that what are mutual relationship between both areas.

III. Database Side Performance Considerations

First of all, we have to consider how data are stored in SQL Server database and how these data are accessed. Seeking every time data file from beginning to end to search appropriate data leads to large delays and consequently to stop the calculation in case of bulk transactions. We can find out in [6][8] data in SQL Server are stored in pages and extents. To effectively access the data indexes must be (clustered or nonclustered) applied, organized in b-tree structures (indexes require appropriate managing, especially in case of writing data). An index (clustered or non-clustered) 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. In OLTP databases when many data is read and write in the same time indexes must be updated, too. SQL Engineer has to ensure, that writes will not have meaningful performance impact on indexes. This can be achieved with Fill Factor parameter. During index creation or rebuilding, the fill-factor determines free space on page level to fill with data. This allows to growth of index data in the future. Specifying this value as 70 means, that 30 percent of each page will be left empty, providing necessary space for index expansion during data insertion to the underlying table. The size of empty space depends on changeability level of data. If there will be no space, indexes can become ineffective without rebuilding indexes (which very often takes a time).

All disk operations pass through so called Buffer Cache, which is located in the memory of the server. So, if this looks as a very important part of whole system, how much memory for an instance of SQL Server should be allocated? In many sources one can find that SQL Server memory should have a maximum of 80% of the total available. Less memory will cause more often paging, less buffer cache will cause more often exchange data with disk, and generally will slow down the system. Of course, memory amount depends on the nature of the operations performed on the database. If there are domination a large variability read/write pages of data (like in OLTP), the buffer should be relatively large to accommodate them in it.

Buffer Cache is very useful during reading data from the database. Database engine reads the data direct from disk drive when it is accessed them first time. After that these data are placed in the buffer cache (this is so called physical reading). Next reference to these data is by use the buffer cache – obviously, it definitely speeds up extracting data (logical reading) (Fig. 2). 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.

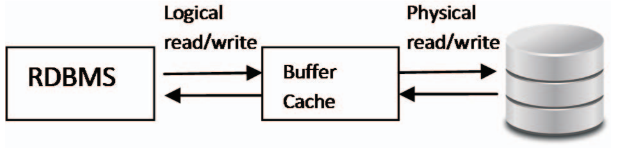


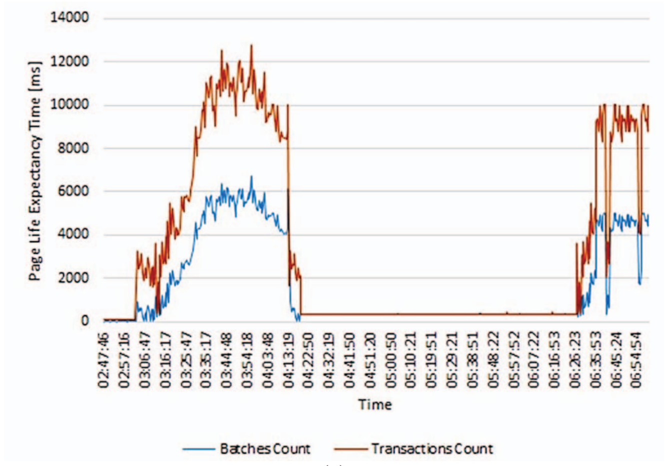
Fig. 2. Read and write data through Buffer Cache.

In case of OLTP databases and numerical calculations one has to take into consideration large number of transaction per short unit of time, which can last long (depend of numerical algorithm). Such algorithms very often refer to the same data, that is why this should reside in buffer cache. The serious problem is that volume of these data is very large and data is often exchanged with data read from the physical disk. In Fig. 3 there is real OLTP processing with large number of batches (SQL statements) executed and large number of transactions[*)].

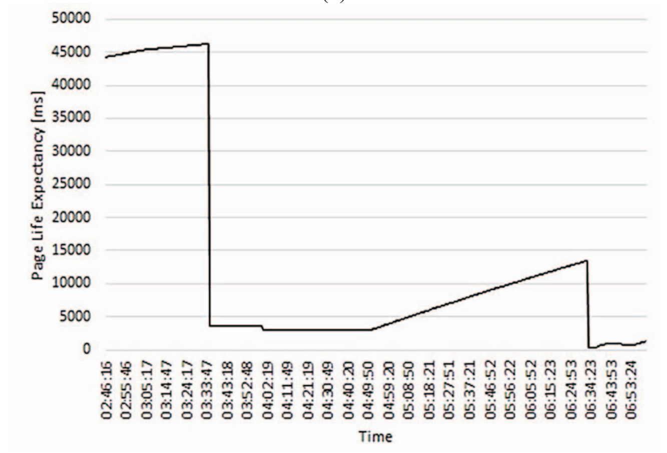
Large number of transaction and batches executed obviously causes data paging between storage and CPU. Data are located based on the nature of data processing – its characteristics, intensity and hardware possibilities. Sudden growth data requests and updates fill the buffer cache with data for further processing very fast. However, data volume is very meaningful, data changes very fast during calculation processing and also expires very fast. Consequently, it must be exchanged with old one. That is why during calculation processing data can be taken directly from disk drive. A parameter confirming such situation 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 live in the buffer cache for long time, and the application that wants to read the data does not have to take them from slower disks (physical read) but from fast memory buffer (logical read) only. The low value of the parameter in turn claims that there is such a large dispersion of read pages that do not fit to these in the buffer and need to be replaced frequently. This is also information for system engineer that the buffer should be expanded (if it is possible).

Cache hit ratio is the parameter that shows how buffer cache is utilized, what is the percentage of data taken from it during processing. Looking at the Fig. 4 it can be noticed that parameter cache hit ratio is very low for period of time (this parameter tells what is the percentage level of data is taken from buffer cache requested by SQL query). We can also correlate this characteristic with Page Life Expectancy and Physical IO parameters (Fig. 6b). Low level of cache hit ratio causes physical IO instead of logical and exchange data between buffer pool and physical IO.

Dramatically drop hit ratio value is reflected immediately in other parameter – Percent Disk Busy (Fig. 4b). We see in this figure the period of disk time is utilized at 100% (measured totally for all reads and writes).

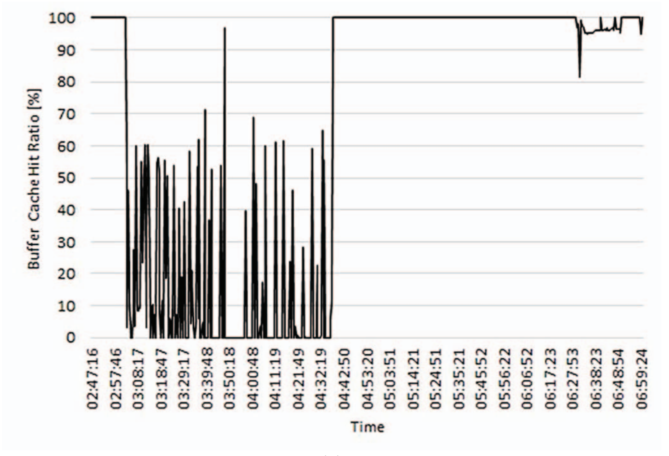


(a)

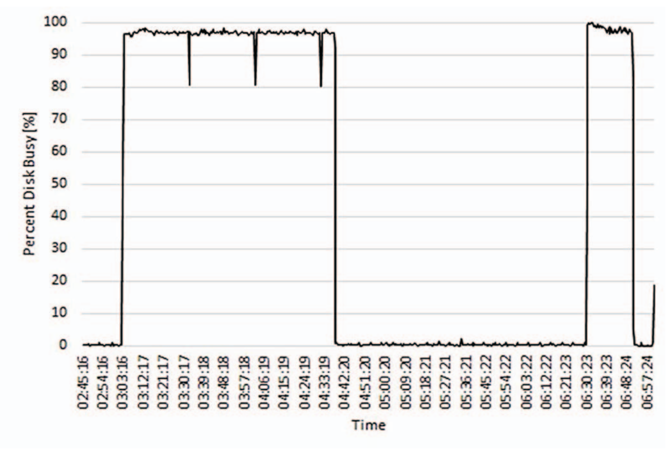


(b)

Fig. 3. Dependency between Page Life Expectancy (a) and number of transactions (b).



(a)

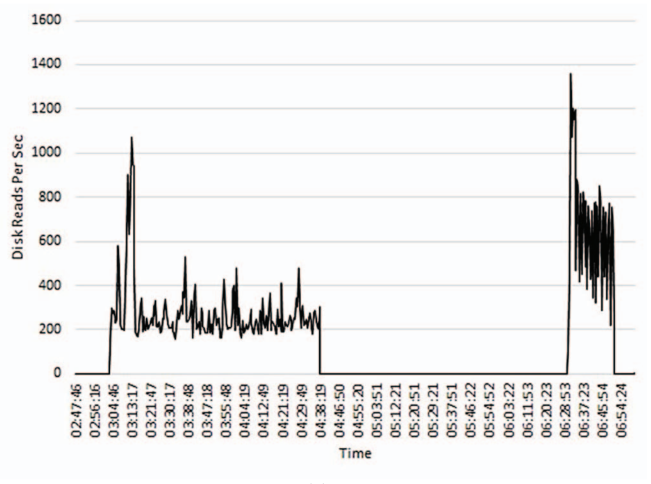


(b)

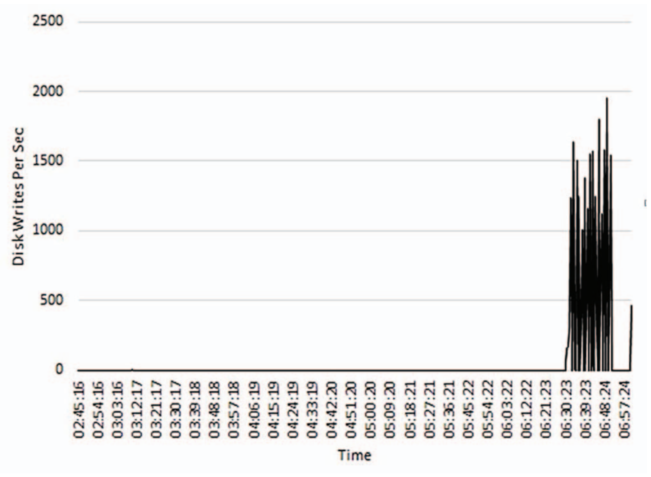
Fig. 4. Buffer Cache Hit Ratio (a) and Disk Busy (b) parameter characteristics.

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

Figures 5 present Disk Reads per second and Disk Writes per second characteristics. First figure 5a shows meaningful growth disk reads two times during batches execution (fluctuates between 300 and 600). During second batches execution it can be seen that additional writes appeared (about 1500 IOPS) – Fig. 5b. That is why total data transfer was meaningful and it could impact negatively on backend physical disk volumes performance.



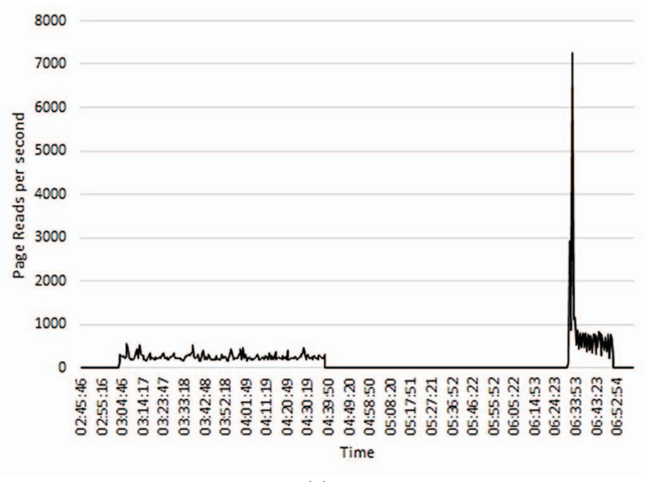
(a)



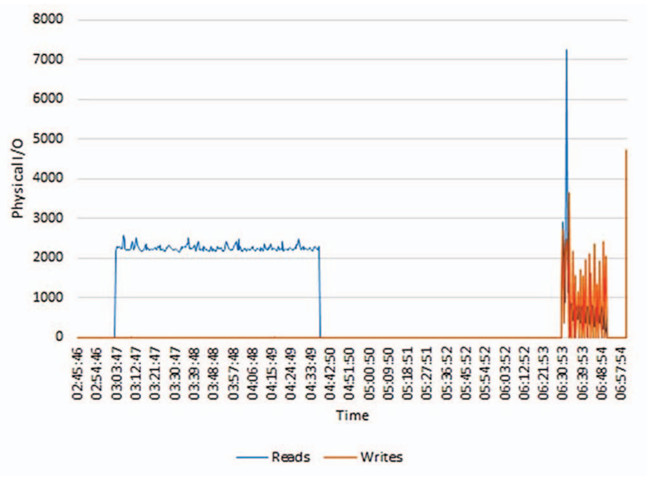
(b)

Fig. 5. Disk Reads per second (a) and Disk Writes per second (b).

In next figures 6 Page Reads per second and Physical IO are shown. First figure 6a reflects situation seen on previous characteristics - meaningful growth page reads during transactions. Notice, that in that time there were not many writes, that is why exchanges in buffer cache are so intensive. When we take a look at second period of time, we can see that buffer cache fluctuates; the lowest Buffer Cache Hit Ratio level is 80% but here writes dominate and they can be done directly to physical disk (that is why again disk is busy in 80%). So, if writes dominates why page life expectancy parameter is again low? The answer is simple - Buffer cache is shared among many databases (it is one for whole SQL Server instance) and Page Life expectancy is calculated for many databases placed on different physical disks.



(a)



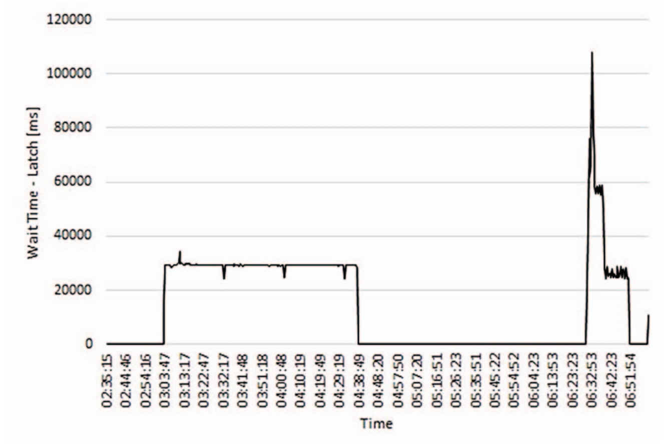
(b)

Fig. 6. Page Reads per second (a) and Physical I/O (b).

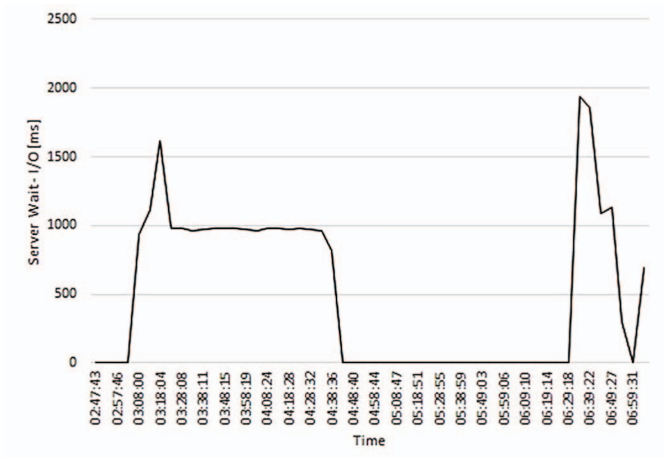
This example shows that even though one database does not utilize buffer cache, the other one can do this and has meaningful impact on the system as a whole. Moreover, one non-optimized database can slow down other well-optimized one. This is why system engineer must have knowledge and observe behavior of all databases on one instance.

It is worth to check how delay characteristics are looking. There are two parameters it can be observe – Wait Time and Server Wait (here *sys.dm_os_latch_stats* and *sys.dm_os_wait_stats* DMVs - dynamic views can be used). Taking into consideration above figures, we are mostly interested in IO Latches (Fig. 7a). As we expected, they appeared in the same period of time where disk was busy in 100% (compare Fig. 4b) causing serious bottleneck. Notice, that wait time is 30s in first case, in second one accidentally 3 times more. Confirmation of this situation is next Fig. 7b on which it can be observed that server waits for IO operation as many 1 – 2 seconds. Such long lasted state extends meaningfully calculation, what is worse, expands transaction log.

We can observe on fig. 6 that paging strongly depends on number of transactions and T-SQL compilations. Generally, increase time of life page in buffer is inversely proportional to transaction amount, compilations and executed batches amount.



(a)



(b)

Fig. 7. Wait Time – Latch (a) and Server Wait for I/O (b).

IV. Storage Side Performance Considerations

SQL Server stores data on disks which can be connected locally or via network (Fig. 8a and 8b). To optimize space utilization the best option is to use centralized storage called array, to which many applications/servers can be connected and storage space can be divided for Logical Units (volumes) and presented to them as DAS (Direct Attached Storage). There are some reasons to do that. First of all flexibility, which allows to expand data volume dynamically as necessary in case of rapid growth. Using so called thin provisioning one can declare volume as target space logically without full physical cover of that space. This feature can be useful to optimize storage maintaining costs and to declare space as necessary. In case of locally connected disks we do not have such flexibility, because amount of disks is very limited. Also, adding new disks is impossible in most cases. Second reason is to use external storage engine and infrastructure to optimize data access and update. Arrays have internal processor, front end, backend and cache components which are optimized for data reading and writing. The most important part – cache – meaningfully improves data access thanks to temporarily store often reading data. Also writing data uses cache and there is internal mechanism to save data in case of failure including battery and microcode which moves data from cache to disk to store them permanently.

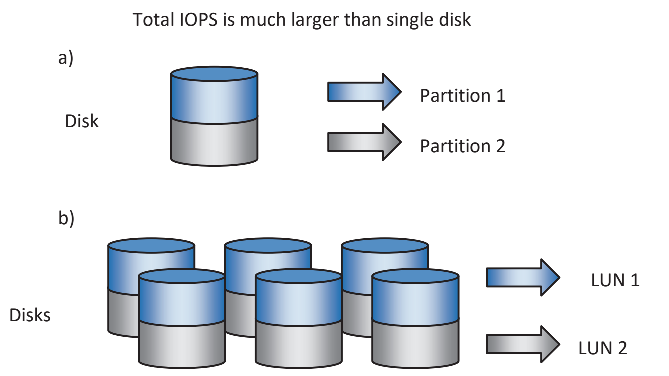


Fig. 8. Data storage system for application local disk (a), disk array (b).

Third reason is to use array microcode to use multipathing, load balancing, tiering and to moving chunks of data from one disk to other to balance physical disk load. Multipathing allows to split data coming from HBA (Host Bus Adapter) between many paths and deliver to disk more efficiently. In case of path failure dedicated software marks this path as degraded and whole load from this path is moved to other one without processing disruption. Tiering is the feature of arrays, thanks to that most loaded chunks of data are moved to other level of disks (so called tier). Disk levels have disks of various types (SSD, SAS, SATA). If data load is too high on given tier, data are automatically move to higher speed disks.

The physical disk subsystem directly affects the numerical calculation performance during OLTP processing. Any performance degradations on disks immediately negatively affect the performance SQL Server data processing. Therefore it is necessary to check regularly the level of disk subsystem consuming parameters like IOPS. Performance testing is often done empirically and makes changes in the course of performance analysis (therefore use of arrays is more efficient than local disks).

There are three most important parameters of disk subsystem should be taken into account for database application of numerical calculation:

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

The number of operations per second (IOPS) - the observation of this parameter is necessary and allows us to evaluate whether the use of the subsystem by the numerical software approaches the technological frontier. In presented example to solve the problem it is necessary to set up RAID group on the right amount of physical disks. Knowing technological IOPS limitation of specific single disk, it is possible to form composite volumes and increase this limit.
In the example above, total IOPS requested was greater than RAID Group IOPS, that is why server was in wait state.

Time to read and write data - this parameter comes directly from disk configuration and the application/database access competition to particular volumes/LUNs by processes. Notice, that physical disk divided on two or more logical volume does not give performance advantage because total

IOPS does not extend (in the case of SAS drive about 210 IOPS). It can be necessary to deploy accessed data to separate physical disks (each of the disks has 210 IOPS) – a profit is therefore twofold. The situation is similar in the case of arrays applied, except that a large number of disks allows for a much improved performance.

Queue to disk – describes how many IOPS waits for disk service. Large value means latency of data operation. Then requests cannot be immediately serviced and must be placed into the queue.

In case of bulk numerical calculations like described earlier it is justified to consider RAID group for storage objective, because physical disk is a bottleneck itself; there is a problem with requests service in acceptable time (disk is busy in 100% and queue to disk enlarges). The main problem is that there are too much IOPS required and single disk cannot fulfil this requirement.

What is the best RAID solution for numerical calculation? First of all it is necessary to make oneself aware that data protection is not a priority. Priority is to get as much IOPS as possible, that is why the simple combination physical disks in RAID 0 (stripping) is the best solution. Moreover, RAID 0 (stripping) having no additional parity data, offers zero write penalty what is especially important during updating data. It is done directly on physical disks and maximum available IOPS is utilized.

V. Correlation SQL Server and Storage Array Parameters

Above aspects of the performance processing of large amounts of data in the OLTP database systems, which also have a numerical applications are critical to the optimization process. Proper optimization process however, requires a broad perspective on the issues of data processing over the entire path of the outermost point data storage or database (a physical disk for data storage). Unfortunately, it often happens that the disk volumes are not directly presented to the database engine, and through the operating system (Fig. 8). Analysis and optimization of data processing requires consideration of intermediate disk volumes which are at the operating system level (e.g. by LVM - Logical Volume Manager).

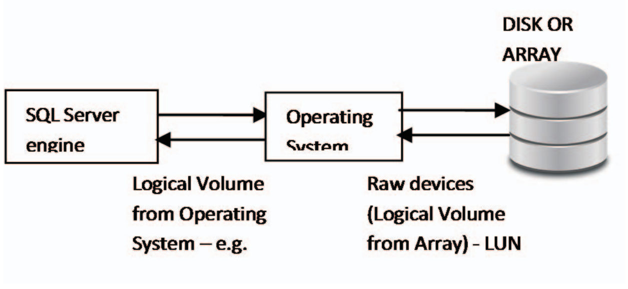

DISK OR ARRAY
SQL Server engine
Operating System
Logical Volume from Operating System – e.g.
Raw devices (Logical Volume from Array) - LUN

Fig. 8. Dependency between database engine and storage volumes.

The simplest analysis can be done only with physical disks parameters and SQL Server characteristics. Above characteristics shows that in numerical calculations one of the serious problems is access to hardware resources. To identify source of problem and find solution it is required to find relationships between particular components.

In case of OLTP SQL Server databases and numerical calculations locks and latches have the largest impact on performance. Locks have been already described in [9] – they appear on database level. Latches however are related to buffer and resource waiting. There are many types of latches and analysis of performance problems requires good knowledge of software and hardware, because various elements can be related to the same latch (for example Network IO can reflect either problems connection between application and database or problems with iSCSI storage implementation).

In Fig. 9 model of correlation SQL Server and Storage parameters is shown, which can be very useful in case of analyzing performance problems on backend side for numerical calculation application. Problems begin with high value of Server Wait parameter. Then one can suspect that any resources block data processing. Here is T-SQL code can be used to extract the information about latches currently on the server:

```
DBCC SQLPERF ('sys.dm_os_latch_stats', CLEAR);
GO
SELECT * FROM sys.dm_os_latch_stats;
GO
```

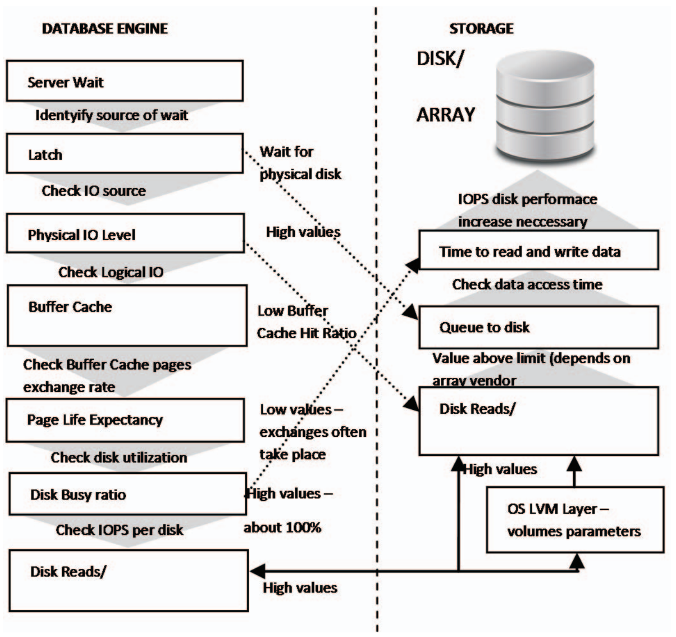


Fig. 9. Model of correlation SQL server and storage performance parameters.

Model presented in Fig. 9 shows direct correlation between parameters database engine and storage with dotted arrows. That means these characteristics are closely related and changes noticed on database side can be also observed on related parameters on storage side. Notice also, that in analysis OS LVM (related to logical disks) can have significance, because, depending on environment, volumes can be seen in database as RAW devices (in Oracle DB engine) - presented directly from disk array or as partitions created on Operating system. In that situation the same disk parameters must be observed on array and operating system especially when RAID is implemented on operating system or array.

The last component on Database side to identify performance problem is buffer pool. Looking at the characteristic of Buffer Pool Cache Hit Ratio and using the following T-SQL query:

```
SELECT count(*)*8/1024 AS 'Cached Size (MB)'
  ,case database_id
FROM sys.dm_os_buffer_descriptors
```

one gets information about cache utilization and all the data pages that are currently in the buffer pool.

For a large number of calculations, in case of insufficient amount of RAM and the same buffer cache space, stored procedures recompile many times. It is recommended to implement buffer pool extension (available in SQL Server 2014) and increase maximum IOPS performance of disk array volumes with increase number of disk in RAID group. Also, RAID 0 type volumes should be used, and also (if it is technologically possible) - multipathing. To enable Buffer Pool Extension Feature it can be done with the following T-SQL code:

```
ALTER SERVER CONFIGURATION
SET BUFFER POOL EXTENSION ON
    (FILENAME = 'E:\BPE\SQL2014.BPE', SIZE = 10
     GB);
```

Taking into consideration that data are stored in 64 kB blocks (recommended) total Input/Output operations per second can be calculated from following formula:

$$\mathrm{IOPS} = \frac{\mathrm{MBps\ Throughput}}{\mathrm{KB\ per\ IO}} \cdot 1024$$

This value is useful to estimate total IOPS serviced by disk subsystem. Knowing that SQL Server data page size is 8192kB (effectively 8060kB), one can estimate how many IOPS service one page of data.

VI. Conclusions

The main objective of this article is to present performance problems with using OLTP relational database-based storage for bulk data calculations. As one can see these problems have performance nature and depend on many elements. Front-end application having SQL Server database as a backend for numerical calculations meet very similar problems like in other cases. Obviously, it is because there are general rules governing queries optimization, execution plans and others.
Although Database engine application as a backend storage is encouraged, implementer has to be aware of disadvantages of such solution. Before of all database must be continuously monitored in point of view all internal structures and physical external storage. As it is presented in this article, parameters of both components tightly depends on each other. The most important areas in SQL server, having impact on performance of database engine, are: Buffer Cache and time of access to disk or volumes.

Engineering of OLTP database administration in numerical calculation environment requires deep knowledge and experience about administering relational database engine, its functionality and performance. To create optimal architecture for scientific data management it needs to take them all into consideration, because it may cause a significant time extension of the calculation.

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.

Jacek Nazdrowicz was born in Poddębice, Poland, in 1975. He received the MSc degrees in Technical Physics (Computer Physics), Computer Sciences (Software Engineering and Networking Systems) 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 participated in EduMEMS project (Developing Multidomain MEMS Models for Educational Purposes). He also educates in COMSOL software. Now he participates in Strategmed project (supported by the National Center for Research and Development)
Between 2007 and 2016 he worked 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.
Since 2016 he also works in Fujitsu Technology Solutions in Lodz in Remote Infrastructure Management Department in Storage Team as a Storage Engineer (SAN Infastructure, Brocade, Netapp, Eternus products). He educates in many data storage and backup technologies (IBM, DELL/EMC, Hitachi, Fujitsu).