Baltic J. Modern Computing, Vol. 12 (2024), No. 1, 1-3

New Storage Devices and the Future of Database Management

S.D. Kuznetsov <kuzloc@ispras.ru>

Ivannikov Institute for System Programming of the RAS,
25 Alexander Solzhenitsyn Str., Moscow, 109004, Russia.
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
Moscow Institute of Physics and Technology (State University),
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
National Research University Higher School of Economics (HSE)
11 Myasnitskaya Str., Moscow, 101000, Russia

kuzloc@ispras.ru

Abstract. At the beginning of the paper, it is demonstrated that the technology of the most widely used SQL-oriented database management systems (DBMS) is inextricably linked with the technology of hard disk drives with movable heads (HDD). Features of HDD affect the data structures and algorithms for performing operations, methods of managing the buffer pool of the DBMS, transaction management, query optimization, etc. At present, new types of data storage hardware have appeared: block solid-state drives (SSD) and storage-class memory (SCM). SSD characteristics made it expedient to develop a DBMS in terms of their exclusive use, but so far, no such DBMS has been created, and SSDs are used simply instead of HDDs in DBMSs that do not take into account their features. The availability of SCM enables radical simplification of the architecture of the database systems and significantly improve their performance. To do this, we need to rethink many of the ideas used in disk-based databases.

Keywords. SQL-oriented DBMS; hard disk drive with movable heads; cost-based query optimization; in-memory DBMS; flash-based solid-state drive; non-volatile main memory

1. Introduction

The technology of the most common SQL-oriented (traditionally called «relational») database management systems (DBMS) is inextricably linked with the technology of storage devices on magnetic disks with movable heads (Hard Disk Drive, HDD). The first HDDs were released by IBM in 1956. HDD technology overcame the shortcomings of the early data storage devices – magnetic tape data storage (purely sequential access) and magnetic drums memory (limited capacity), providing capacity less than for magnetic tapes, but much larger than magnetic drums, and the speed of arbitrary data exchanges between the main and external memory smaller than magnetic drums, but much bigger than magnetic tapes. If we add to this the moderate cost of HDD, then these devices were quite suitable for storing databases.

The technology of the DBMS was influenced by the technological features of the HDD. First, the HDD provides external memory, input/output (i/o) with which is usually conducted in blocks of bytes of the same size. This feature leads, at a minimum, to two architectural solutions. (1) To store databases and speed up the processing of queries, data structures and algorithms should be chosen, for which the block nature of external memory is natural. In particular, the most common data structures for indexes are varieties of B-trees (Bayer and McCreight, 1972). (2) To balance the low speed of arbitrary i/o with external memory and relatively high speed of data processing in the main memory, the DBMS performs its own buffering (caching) of the external database memory blocks in the main memory (Hellerstein and Stonebraker, 2005), (Kuznetsov, 2012a).

Secondly, when performing i/o with an external storage of HDD, the disk hardware performs three basic operations: moving the heads to the required cylinder of the disk packet (seek), rotating the disk packet to the required angle distance (latency), reading or writing data while transferring it to or from the main memory (data transfer). When executing arbitrary i/o, the execution time of the first two operations is measured in milliseconds, which means that the time of reading an arbitrary data block from external memory or its writing is longer by several decimal orders than the corresponding rewriting cycle in the main memory. Therefore, when executing any SQL-level operation on the database, the decisive overhead is the amount of required i/o with external memory. This observation is the basis of cost-based query optimization based on pioneer work (Selinger et al., 1979) and is used in all mature SQL-oriented DBMS.

The above remarks are enough to be convinced of the deep dependence of the most common technology of SQL-oriented DBMSs on HDD features. Orientation to the use of these storage devices affects both the overall architecture of the DBMS, and the choice of the main data structures and algorithms.

In the late 1970s - 1980s, attempts were made to create specialized hardware for use with DBMS, including data storage hardware with fixed-head disks (head-per-track disks). Moreover, there were prototypes of devices with special microprocessors built into the magnetic heads, filtering data "on the fly" when reading from the disk (processor-per-track systems and processor-per-head systems) (DeWitt and Hawthorn, 1981). However, by the beginning of the 1990s, it became clear that this approach was a dead end (DeWitt and Gray. 1992), and for the next two decades the technology of DBMS was based mainly on storage devices of HDD category.

In the late 1980s - 1990s, an alternative database technology appeared and developed with the storage of databases in conventional volatile memory (in-memory DBMS) (DeWitt et al., 1984). In such DBMS data structures and algorithms for performing operations differ from those used in disk DBMSs. In particular, when choosing data structures, one must take into account the presence of cache memory in processors (Shaporenkov, 2006). The principles of query optimization should also differ, although there is almost no information about query optimizers in in-memory DBMS in the available literature (so it seems that these principles do not exist).

Probably the most mature representatives of this category of DBMS are TimesTen (Lahiri et al., 2013), existing since 1996 and acquired by Oracle in 2005, and solidDB (Lindström et al., 2013), existing since 1992 and acquired by IBM in 2007. These systems support very fast execution of queries to databases (since the database and all indexes are entirely stored in the main memory), however, for performing database modification operations, external memory accesses are required, so that the speed of such operations does not differ from the case when database is stored on disk.

A special case of in-memory DBMS is VoltDB (Stonebraker and Weisberg, 2013), which is a transactional massively parallel system without shared resources. In this system, the durability of transactions is supported by replicating data in multiple nodes, and external memory is not used at all. For details of the VoltDB organization (and its prototype H-Store), see (Kuznetsov, 2011).

According to rumors, the TimesTen is moving in a similar direction. These rumors are partially confirmed by the presence in the Oracle TimesTen In-Memory Database family of the High Availability option (Cheung et al., 2013) provided by replication in a clustered environment. Interestingly, although there are no similar rumors about solidDB at the time, the high availability option is also supported for this system (Salkosuo, 2010).

It should be noted that despite the presence of a number of advantages of in-memory DBMS over disk DBMSs, at present there is practically no competition between them. This, first of all, is due to natural limitations on the size of databases inherent in in-memory DBMS.

In the first decades of the 21st century, significant changes occurred (and continue to occur) in the technology of hardware storage. So-called block solid-state drives (Solid-State Drive, SSD), based on flash memory technology and relatively quickly catching up with HDD in terms of maximum capacity (up to 60 terabytes in 2016), surpassed them in most other measures (losing mainly only in price). In the next section, we will briefly discuss the opportunities of using SSD in DBMS architecture, the components of DBMS that should be maximally affected by the transition from HDD to SSD, as well as the real state of affairs in DBMS technology 10 years after the SSD on flash memory became really available.

In recent years, the prospect of the emergence of nonvolatile random access memory (NVRAM, also called, probably more expressively, Storage Class Memory, SCM) on the market became real. This memory allows byte addressing, directly accessible to processor instructions, but it preserves the contents after the power failure.

Using SCM opens the way to building a database based on single-level memory. These databases can be much faster than disk-based ones while having a simpler organization. The third section of the article is devoted to the prospects of the appearance of such DBMS and existing problems.

2. Flash-based Solid-State Disk Drives and DBMS Technology

Like HDD, SSD is a block external storage device that saves data after turning off the power. The main differences between SSD and HDD are the following:

- there are no mechanical components in the SSD, hence for any block the speed of i/o with the SSD is the same;
- while the average exchange time with an arbitrary HDD block is about 10 milliseconds for both reading and writing, the time for reading an arbitrary block in modern SSDs is about 20 microseconds (three decimal orders less than the HDD), and the writing time is about 200 microseconds (two decimal orders less than the HDD);

- while SSDs are more expensive than HDD (about 10 times in 2016), but the cost of HDD in terms of terabytes of supported memory has stabilized in recent years, and SSD is getting cheaper;
- currently, SSDs are significantly less reliable than HDDs.

2.1. SSD-oriented DBMS

Only the last feature on the list can prevent the full-scale application of SSD in the DBMS in principle. It's unclear whether SSD developers will be able to get rid of this defect, but the first two characteristics seem so attractive that 10 years ago I tried (without much success) to persuade my students to study DBMS architecture in which SSDs are used to store databases.

It is clear that the features of SSD could most of all affect the management of external memory, the management of main memory buffers and the query optimizer. In an existing disk DBMSs, since when executing queries it is often necessary to perform a full scan of tables without using indexes, one tends to place blocks of one table on the disk to avoid large movement of the magnetic heads when moving from the current block to the next one. In a DBMS exclusively based on the use of SSD, blocks of one table can be placed in an external memory in an arbitrary manner.

The time of writing a block to external SSD memory is decimal order more than the block reading time due to the need for preparation of the external memory sector before writing to it (Novotný et al., 2015). When managing the main memory buffers in a DBMS designed for the use of SSD, it makes sense to prepare in advance for writing an external memory sector and then, when pushing the changed image of a previously read external memory block from the buffer to external storage, write it not to the sector from which it was read, but to some sector already prepared for writing.

But the allocation of external memory and the management of main memory buffers are almost trivial compared to query optimization. As it was noted in the introduction, modern cost-based optimizers are based on the assumption that arbitrary i/o with external memory is so slow that the cost of the query execution plan can be estimated by the number of i/o required, neglecting the time that will be required for processing the data. However, reading from external SSD memory is 1000 times faster than on using an HDD. Therefore, when going from HDD to SSD, this assumption would have to be subjected to a rigorous revision.

It means that the direct transfer of estimates of the query execution plans from the HDD environment to the SSD environment can lead to disastrous results. Incorrect accounting of the time spent on exchanges with external memory and processing of data in the main memory can lead to the selection by the query optimizer of obviously suboptimal plans for query execution, which will lead to underutilization of the SSD potential. Of course, the queries will not be executed slower than with the HDD, but that is not enough to change the external memory management hardware. In other words, for effective use of SSD, query optimizers need to be significantly redesigned.

Despite the attractiveness of the idea of replacing HDD by SSD in hardware support of DBMS, there are practically no projects (neither commercial nor research) for development of SSD-oriented DBMSs. I managed to find only the FlashyDB project, run at the German Reutlingen University (Web (a)). The following project objectives were announced:

- explore the impact of flash-based SSD on the architecture and performance of existing database systems, relational data warehouses, and column store systems;
- develop algorithms and data structures that ensure optimal use of the characteristics of SSD based on flash memory in OLTP and OLAP scenarios;
- implement a prototype system.

The list of research directions covered in the project includes database system architectures, transaction processing, multiple user access management, recovery after different failures, buffer management, indexing, query optimization, data placement. As can be seen, the general focus of the project is consistent with the above considerations. Apparently, one of the first paper devoted to the FlashyDB project was (Petrov et al., 2015). A complete list of published papers is available on the project website (Web (a)). As this list shows, the project has achieved significant results in far from all of the areas of research identified.

Perhaps the lack of activity of researchers to build true SSD-oriented DBMS is due to the fact that until recently the maximum capacity of storage devices in flash memory was limited to one terabyte. However, the technology is developing rapidly, and as early as 2016 Samsung introduced a 32 TB SSD and promises to bring its SSD capacity up to 100 TB. Seagate showed a SSD with a capacity of 60 TB. I think this will "spur" the database community.

2.2. Two-level SSD-based cache

In the time when the capacity of SSD was relatively small, the idea of using SSD as part of a hierarchical two-level buffer in a traditional HDD-oriented DBMS was rather popular (Kuznetsov and Prokhorov, 2012). The essence of the idea is simple enough. If for some reason we want to continue to use HDD to store databases, but at the same time receive sufficient benefit from SSD use, why don’t we temporarily store some of the database blocks that can be probably needed in this moment of time in flash memory.

To implement this idea, it is enough to change only one component of the traditional HDD-based DBMS – the main memory buffer manager. The buffer becomes two-level: the first-level cache is located in the main memory, and the second-level cache is in the flash-based SSD. The database blocks required to perform operations on the database are read from the HDD-based external memory to the buffer pages of the first-level cache. If there is a lack of memory in the first-level cache, some buffer page is replaced. If the content of chosen page changed after reading from external memory, then the page moves to the second-level cache (taking into account the notes on buffer managing from subsection 2.1). If there is not enough memory in the second-level cache, the replaced block is moved to the external HDD-based memory.

In (Kuznetsov and Prokhorov, 2012), an overview of algorithms for management of such two-level buffer pool is given. All known algorithms are complex and resource intensive. I do not know of any DBMS in which these algorithms were applied. Nevertheless, it seems that the introduction of a two-level cache with SSD in a disk-oriented DBMS is the cheapest way to modify a DBMS to improve its performance by using SSD technology.

In this case, the most frequently used blocks of the database gradually fall into the cache of the second level, access to which then occurs with the speed of SSD. In addition, since the flash memory is non-volatile, there is no need to push pages out of the SSD-based memory into the HDD-based memory in any case except for lack of space.

However, this approach does not eliminate the need to develop true SSD-oriented DBMSs, in which the storage system characteristics are taken into account in all components.

2.3. Hybrid drives

The easiest way to get some gain in DBMS performance from using SSD technology is to simply replace HDD hardware with SSD hardware without any DBMS changes. As I mentioned in subsection 2.1, database operations after this will not become slower, and most likely they will be executed on average faster.

If there are large databases, to change the hardware of data storage will be quite expensive, and vague promises of a better life (at a qualitative level) can hardly encourage managers to allow such expenses. In hybrid devices of data storage on hard disks (solid-state hybrid drive, SSHD) technologies SSD and HDD are jointly used.

Within SSHD, SSD is used to cache the contents of HDD blocks, which are most often accessed. As a result, SSHD often runs at SSD speed at a cost close to the cost of HDD. It is not so expensive to try to improve DBMS performance due to the transition from using HDD to SSHD, although, of course, this solution does not rely on any technological arguments and remains risky.

3. Storage-Class memory: prospects for DBMS

At present, three technologies can provide real SCM solutions: Phase-Change Memory (PCRAM) (Raoux et al., 2008), Resistive Random-Access Memory (RRAM) (Strukov et al., 2008) and Magnetoresistive Random-Access Memory (MRAM) (Chi et al., 2016).

PCRAM is based on the behavior of chalcogenide[1], which on heating can "switch" between two states: crystalline and amorphous. The crystalline and amorphous states of the chalcogenide are fundamentally different in electrical resistance. An amorphous state with a high resistance is used to represent a binary 0, and a crystalline state with a low resistance level represents 1.

The main idea of RRAM is that dielectrics, which in the normal state have very high resistance, after applying a sufficiently high voltage can form inside themselves conductive low-resistance wires, and in fact turn from a dielectric into a conductor. By applying the appropriate voltage levels, the conductive wires can be destroyed (and the material will again become a dielectric) and formed again (and the material will again become a conductor). There are several state switching effects. One of them requires one voltage polarity for switching operations from low to high resistance level (bit clearing operation), and the opposite polarity for switching from high to low resistance (bit setting operation).

[1] Binary chemical compounds of chalcogenes (elements of the 16th group of the periodic system, which include oxygen, sulfur, selenium, tellurium, polonium and livermorium) with metals.

Data in the MRAM is stored in magnetic memory elements. Magnetic elements are formed of two ferromagnetic layers separated by a thin layer of dielectric. One of the layers is a permanent magnet magnetized in a certain direction, and the magnetization of the other layer changes under the action of an external field. The memory device is organized on the principle of a grid consisting of separate «cells», each of which contain a memory element and a transistor. The technology of recording information and reading is based on a change in the magnitude of the magnetic field.

I will not dwell on which computer companies prefer this or that SCM technology. For a couple of years, various large companies have promised to start producing corresponding chips in the near future. Recently SSDs based on SCM with block exchanges (not on flash memory) have appeared on the market. I think that the corresponding RAM will appear no later than 2018.

It is interesting that in 2011 Russian state corporation Rosnano signed an agreement with the French company Crocus on setting up in Russia «the production of medium and high density MRAM memory based on the 90 and 65 nm manufacturing processes» (Web (b)). For fairness, it should be noted that Samsung plans to begin mass production of such memory based on 28-nanometer technology (Lin and Shen, 2017).

Nevertheless, the choice to manufacture MRAM memory in Russia seems to be justified, since the MRAM's expected read and write time is about 20 ns (less than today's DRAM) with endurance commensurate with the endurance of DRAM and HDD, and read time of PCRAM and RRAM is several times larger (and writing is slower than reading), and the endurance is much smaller (Arulraj and Pavlo, 2017).

Of course, before the appearance of different types of SCM in the market it is impossible to reliably compare their characteristics, but there is a hope that MRAM with the promised characteristics will indeed appear, and I rely on it further on in this article.

It should also be noted that nonvolatile random access memory will be used in computers whose processors are equipped with fully volatile caches. To ensure that transactions can be committed in SCM, two commands have been added to the Intel processor instruction set – CLWB and CLFLUSH (Web (c)). Both commands are designed to push data from caches of all levels into SCM, but the first command saves the data being ejected from the cache, and the second command forces to read data from the SCM during next access.

3.1. SQL-oriented DBMSs based on SCM

At first glance, it would be prudent to use any available in-memory DBMS as the base for developing a database system that uses only SCM to store data (and doesn't use any external storage at all). Indeed, the in-memory DBMS, like the SCM-based database system, stores the entire database in the main memory. This determines the choice of main data structures and algorithms for performing operations, and the design of the query optimizer.

However, there is a fundamental difference between an in-memory DBMS and a DBMS based on SCM, which does not allow simple reuse of existing solutions: in-memory DBMSs are designed to use traditional volatile main memory, and SCM-based DBMSs use non-volatile main memory. To support the durability of transactions in in-memory DBMSs, external memory is used (HDD or SSD – is not important here), that is, *as in disk DBMSs*, a *two-level* memory hierarchy is used, the first level of which contains the volatile main memory, and on the second – non-volatile external memory. *Unlike disk DBMS*, in this case the main memory stores the entire database (and does not serve as a cache), and the external memory serves to support transaction durability[2].

When developing a DBMS based on SCM, we are dealing with a fundamentally *one-level* database storage environment with byte addressing available. In this case, generally speaking, we can completely abandon the block structure and start allocating memory (for all purposes related to database support) in portions of arbitrary size. It is worthwhile to think about whether this can be useful and, if so, reflect on the non-volatile main memory allocation by fragments of arbitrary size:

- how to deal with external fragmentation?
- is data shuffling permissible?
- is it worth using some kind of buddy system (for example, Fibonacci buddy system (Aho et al., 1983))? etc.

It seems that, if there is no memory with block structure, there is also any no reason to use B-trees for organizing indexes[3]. Then new questions arise:

- what is possible to use instead of B-trees?
- is it worth to use some method of searching in main memory based on trees (basically, binary trees are used in these methods) (Kuznetsov, 2003)?
- whether is it better to use some hash-based search method (Kuznetsov, 2003)?
- or is it better to look for or come up with something new?

Transaction management should be deeply rethought. In particular, following questions should be resolved:

- how to support transaction serialization in transactional systems?
- should we use versioning algorithms and what should they be in this case?
- is it worth saving on garbage collection in DBMS based on SCM, the need for which arises if we do not limit the number of versions of database objects?
- how to manage logging in SCM?
- do we need logical and physical logs?
- what should be an elementary entry of the physical log?

Finally, how to optimize queries? Query optimization should be fast and precise.

- How to resolve this contradiction?
- Should we continue to use cost-based optimization?
- How to build cost formulas?

There are a lot of questions, and all of them need to be answered correctly in order to obtain real benefits from the development of DBMS based on SCM. Unfortunately, although the need for a non-volatile main memory was noted back in 1987 by Michael Stonebraker during the development of Postgres (Stonebraker, 1987), currently there are practically no projects for full-scale development of SCM-based DBMS. This is in particular confirmed by the fact that at the SIGMOD conference in 2017 the tutorial "How to build a database management system in the main nonvolatile memory" (Arulraj and

[2] As noted in Section 1, a special case is represented by the DBMS VoltDB that basically works in the shared nothing mode in a massively parallel environment.
[3] In general, it seems strange to use B-trees in an in-memory DBMS - after all, by its nature B-tree is a disk structure of memory

Pavlo, 2017) was presented by Joy Arulraj and Andrew Pavlo from the Carnegie Mellon University who are leaders of the Peloton project (Web (d)).

The list of the main characteristics of the project includes native support for data storage technology based on the main nonvolatile memory. Unfortunately, as the name of the project shows, this project goal is not the main one. The main goal is the integration of artificial intelligence components to provide the possibility of autonomous (self) system optimizations depending on the current workload (Pavlo et al., 2017).

Nevertheless, at present the project participants (Web (d)) seem to have the most extensive experience in developing DBMS based on SCM. It is absolutely necessary to start new projects, actively explore possible approaches, to hold special seminars and conferences to exchange ideas and experiences.

To conclude this subsection it should be noted that potential advantages of SCM-based DBMS approach for transactional applications are obvious. The speed of processing transactions can be achieved almost equal the speed of main memory. This is a fundamentally new quality. As a hardware platform for SCM-based DBMSs computers are suitable whose processors have multi-core and / or multi-threaded organization, including powerful graphics accelerators.

Unfortunately, it is hard to find a scenario, in which the use of SCM can provide significant advantages for analytical applications. It is a common vision that horizontally scalable analytical databases should be based on the use of massively parallel architectures and the principle of shared nothing (Kuznetsov, 2012b). Modern analytical databases are so large the database can completely place only in a cluster, whose nodes have very large storage capacities. The overhead of data transferring over the network can be unacceptable even with the use of disk memory. If SCM is used in the nodes, network overheads can negate all the benefits of SCM.

3.2. SCM for object-oriented and XML-oriented DBMSs

In the 21st century, object-oriented databases almost lost users. At the same time, various means of object-relational mapping (ORM) are actively used, which allow object-oriented applications in an object-like manner to interact with SQL-oriented databases (Neward, 2006). In principle, it would be better to use object-oriented DBMS (OODBMS) for storage of objects, rather than ORM[4].

It seems that the prevalence of OODBMS was largely limited by the problem that is partly related to the object-oriented data model (Cattel and Barry, 2000). As widely known, one of the basic concepts of this data model is the Object Identifier (OID) that is automatically generated by the system when creating any object, uniquely distinguishes this object from all other objects of any object type, and serves as a kind of abstract pointer to the object. In particular, in the ODMG model, relationships between objects are formed with use of OIDs.

When OODBMS uses block external memory to store databases, it is difficult to explicitly use ordinary pointers as OIDs. In addition, the problem of converting OIDs to regular pointers (swizzling) when moving objects from a database to an object-oriented

[4] As (Kuznetsov, 2015) demonstrates, it is possible to use, with equal success, object capabilities of the SQL language itself, but this idea is not widely used.

environment of client applications has long been known (Kemper and Kossmann, 1995). If you base OODBMS on SCM, both problems seem to be greatly simplified, and the navigational nature of OODBMSs will not severely hamper its operation, since the costs of dereferencing OIDs can be reduced to almost zero.

Likewise, the use of SCM can revive interest in XML-oriented DBMS, in which it is necessary to maintain a lot of links to support path expressions, etc., and to use more sophisticated storage schemes to ensure somewhat acceptable efficiency (Taranov, et al., 2010). Obviously, with 64-bit addressing and a sufficient amount of basic non-volatile memory, XML-oriented DBMS can be dramatically simplified and accelerated.

4. Conclusion

As you can see, scenarios in which SCM can significantly improve the efficiency of DBMS and simplify their organization are more than enough. It is necessary to continue to analyze different branches of the discipline of data management, so as not to miss other favorable opportunities for SCM application. In particular, it would be very interesting to find ways to use SCM in analytical DBMS. And of course, a large number of research projects are required to find the right ways to develop a DBMS based on SCM.

This paper is slightly modified English version of (Kuznetsov, 2017) presented at the APSSE'2017 International Conference.

References

Aho A., Ullman J., Hopcroft J. (1983). *Data Structures and Algorithms*. Pearson, 427 pp.

Arulraj J., Pavlo A. (2017). How to Build a Non-Volatile Memory Database Management System. *Proceedings of the 2017 ACM International Conference on Management of Data*, 1753–1758.

Bayer R., McCreight E. (1972). Organization and Maintenance of Large Ordered Indexes, *Acta Informatica*, vol. 1, issue 3, 173–189.

Cattel R.G.G., Barry D.K., eds. (2000). *The Object Data Standard: ODMG 3.0*. Morgan Kauffmann Publishers, 280 p.

Cheung S., Law S., Bloom J., Yang J. (2013). *Oracle TimesTen In-Memory Database High Availability. Using TimesTen Replication to deliver High Availability and Disaster Recovery.* Oracle White Paper, *http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/wp-timesten-ha-2735640.pdf*. Accessed 23 December 2017.

Chi P., Li S., Cheng Yu., Lu Yu, Kang S.H., Xie Yu. (2016). Architecture Design with STT-RAM: Opportunities and Challenges. *Proc. of the 21st Asia and South Pacific Design Automation Conference*, 109–114.

DeWitt D.J., Gray J. (1992). Parallel database systems: the future of high performance database systems. *Communications of the ACM*, vol. 35, Issue 6, June 1992, pp. 85–98.

DeWitt D.J., Hawthorn P.B. (1981). A Performance Evaluation of Data Base Machine Architectures (Invited Paper). *Proceedings of the 7th International. Conference on Very Large Data Bases*, pp. 199–214.

DeWitt D.J., Katz R.H., Olken F, Shapiro L.D., Stonebraker M.R., Wood D.A. (1984). Implementation techniques for main memory database systems. *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, 1–8.

Hellerstein J.M., Stonebraker M. (2005). Anatomy of a Database System. *Readings in Database Systems*, 4th Edition. MIT Press, 42–95.

Kemper A., Kossmann D. (1995). Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis. *The VLDB Journal*, vol. 4, issue 3, 519–566.

Kuznetsov S.D. (2003). Sort and search methods. *http://citforum.ru/programming/theory/sorting/sorting2.shtml*. Accessed 10 October 2017 (in Russian).

Kuznetsov S.D. (2011). Transactional Massive-Parallel DBMSs: A New Wave. *Trudy ISP RAN/Proc. ISP RAS*, v. 20, 189–251 (in Russian).

Kuznetsov S.D. (2012a). *Databases*. Academia, series: University textbook, 496 pp. (in Russian).

Kuznetsov S.D. (2012b). To the freedom of the big data problem. *Open Systems*, issue 2, 2012, 22–24 (in Russian).

Kuznetsov S.D. (2015). ODMG and SQL object models ten years later: there are no contradictions. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 1, 2015, 173–192. DOI: 10.15514/ISPRAS-2015-27(1)-9 (in Russian)

Kuznetsov S.D. (2017). Prospects and problems of using nonvolatile memory. *Proceedings of the 5th International Conference on Actual Problems of System and Software Engineering*, CEUR Workshop Proceedings, vol. 1989, 7–21 (in Russian)

Kuznetsov S.D., Prokhorov A.A. (2012). Flash-based algorithms of database buffer management. *Trudy ISP RAN/Proc. ISP RAS*, v. 23, 173–194. DOI: 10.15514/ISPRAS-2012-23-11 (in Russian).

Lahiri T., Neimat M.-A., Folkman S. (2013). Oracle TimesTen: An In-Memory Database for Enterprise Applications. *Bulletin of the Technical Committee on Data Engineering*, vol. 36, no. 2, 6–13.

Lin Y., Shen J. (2017). Samsung ready to mass produce MRAM chips using 28nm FD-SOI process. *https://digitimes.com/news/a20170925PD206.html*. Accessed 10 October 2017.

Lindström J., Raatikka V., Ruuth J., Soini P., Vakkila K. (2013). IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability. *Bulletin of the Technical Committee on Data Engineering*, vol. 36, no. 2, 14–20.

Neward T. (2006). The Vietnam of Computer Science. *Ted Neward's Blog*, Jun 26, 2006. Accessed 10 October 2017.

Novotný R., Kadlec J. and Kuchta R. (2015). NAND Flash Memory Organization and Operations. *Journal of Information Technology & Software Engineering*, vol. 5, issue 1, 8 p.

Pavlo A., Angulo G., Arulraj J., Lin H., Lin J., Ma L., Menon P., Mowry T.C., Perron M., Quah I., Santurkar S., Tomasic A., Toor S., Van Aken D., Wang Z., Wu Y., Xian R., Zhang T. (2017). Self-Driving Database Management Systems. *Proceedings of the 8th Biennial Conference on Innovative Data Systems Research* (CIDR '17), Online Proceedings, 6 p.

Petrov I., Gottstein R., Hardock S. (2015). DBMS on modern storage hardware. *Proceedings of the 31st International Conference on Data Engineering* (ICDE), 2015, 1545–1548.

Raoux S., Burr G. W., Breitwisch M. J., Rettner C. T., Chen Y.-C., Shelby R. M., Salinga M., Krebs D., Chen S.-H., Lung H.-L., Lam C. H. (2008). Phase-change random access memory: A scalable technology. *Journal of Research and Development*, vol. 52, no. 4/5, 465–479.

Salkosuo S (2010). Introducing IBM solidDB High Availability and Transparent Connectivity. IBM developerWorks, *https://www.ibm.com/developerworks/data/library/techarticle/dm-1003soliddbha/dm-1003soliddbha-pdf.pdf*. Accessed 23 December 2017.

Selinger P. G., Astrahan M.M., Chamberlin D.D., Lorie R.A., Price T.G. (1979). Access Path Selection in a Relational Database Management System. *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, 23–34.

Shaporenkov D.A. (2006). *Efficient methods of data indexing and query execution in in-memory database management systems*. PhD Thesis. Saint-Petersburg State University (in Russian).

Stonebraker M. (1987). The Design of the POSTGRES Storage System. *Proceedings of 13th International Conference on Very Large Data Bases*, 289–300.

Stonebraker M., Weisberg A. (2013). The VoltDB Main Memory DBMS. *Bulletin of the Technical Committee on Data Engineering*, vol. 36, no. 2, June 2013, 21–27.

Strukov D.B., Snider G.S., Stewart D.R., Williams R.S. (2008). The missing memristor found. *Nature*, 453, 80–83.

Taranov I., Shcheklein I., Kalinin A., Novak L., Kuznetsov S., Pastukhov R., Boldakov A., Turdakov D., Antipin K., Fomichev A., Pleshachkov P., Velikhov P., Zavaritski N., Grinev M., Grineva M., Lizorkin D. (2010). Sedna: Native XML Database Management System (Internals Overview). *Proceedings of the 2010 International Conference on Management of Data*, 1037–1046.

Web (a). *FlashyDB project website*, `http://dblab.reutlingen-university.de/FDB.html`. Data Management Lab, Reutlingen University, Germany. Accessed 10 October 2017.

Web (b). *MRAM: Organization of manufacturing of magnetoresistive RAM in Russia*, `http://www.rusnano.com/projects/portfolio/crocus-technology`. Accessed 10 October 2017 (in Russian).

Web (c). *Intel 64 and IA-32 Architectures. Software Developer's Manual. Documentation Changes*. July 2017. `https://software.intel.com/sites/default/files/managed/3e/79/252046-sdm-change-document.pdf`. Accessed 10 October 2017.

Web (d). *Peloton project website: The Self-Driving Database Management System*, `http://pelotondb.io/`. Database Group, Carnegie Mellon University. Accessed 10 October 2017.