

Survey of Architectures of Parallel Database Systems

L. B. Sokolinsky

South Ural State University,
pr. im. V.I. Lenina 46, Chelyabinsk, 454080 Russia
e-mail: sokolinsky@acm.org

Received February 26, 2004

Abstract—The paper is devoted to the classification, design, and analysis of architectures of parallel database systems. A formalization of the notion “parallel database system” is suggested, which relies on a concept of a virtual machine. Based on this formalization, a new approach to the classification of architectures of parallel database systems is suggested. Requirements to parallel database systems are formulated, which serve as criteria for comparing various architectures. Various classes of architectures of parallel database systems are considered and compared.

1. INTRODUCTION

A *parallel database system* is a database management system (DBMS) implemented on a multiprocessor system with high-degree connectivity. A multiprocessor system with high-degree connectivity is a system containing many processors and disks in which the processors are connected with each other by means of a communication network such that the network data exchange time is comparable with the time of the data exchange with a disk. This definition excludes from the consideration distributed DBMSs implemented on several independent computers connected through a local or/global networks. The latter systems possess their own specific features (such as those associated with different geographic locations of the computers, local autonomy, and software and hardware heterogeneity [1]), and the problems associated with the large number of processor nodes are usually not considered for such systems. However, there exist a wide spectrum of systems—starting from traditional one-processor DBMSs ported to symmetric multiprocessor systems that use only intertransaction parallelism and ending with complex parallel systems implemented on clusters or multiprocessors with massive parallelism that use partitioned parallelism [2]—that meet the above definition.

Currently, there exist several approaches to classifying parallel computation systems. A good survey of the existing methods of the description and classification of architectures of computational systems can be found in the book [3]. However, the existing classifications of architectures of multiprocessor systems are either too general from the DBMS standpoint (first of all, this refers to the Flynn classification [4]), or too complicated (for example, a taxonomic system suggested in [5]), or not quite adequate (this refers both to the Flynn classification and the structural–functional classification from [6]). The Stonebraker classification [7] has purposely been developed for parallel database sys-

tems, but, currently, it is not quite adequate either [8]. This is explained by the fact that the existing classification approaches are based on mapping the parallel database system architecture directly to the hardware architecture of the multiprocessor system, as shown in Fig. 1.

The classification problem specified above can be solved by introducing some additional abstraction level based on the notion of a *virtual parallel database machine*. The architecture of a parallel database system is mapped onto the architecture of the virtual parallel database machine, which, in turn, is mapped onto one or another hardware architecture of a multiprocessor system (Fig. 2). We used this approach for the classification of modern parallel database architectures. On the

@

	SN	CE	CD	CDN
Scalability	2	3	3	3
Data availability	2	1	3	3
Load balancing	0	2	1	1
Interprocessor communications	0	2	1	1
Cache coherence	3	2	0	3
Concurrency control	3	2	0	3
Sum of points	10	12	8	14

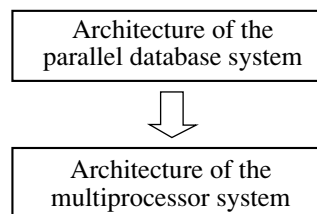


Fig. 1. Traditional approach to the classification of architectures of parallel database systems based on the classification of the hardware architecture.

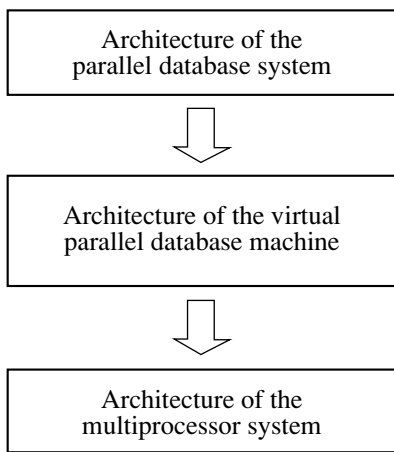


Fig. 2. Classification based on the notion of a virtual parallel database machine.

basis of the classification constructed, a qualitative comparative analysis of various architectures is carried out.

The remaining part of the paper is organized as follows. In Section 2, a notion of a virtual parallel database machine is defined. Based on this notion, in Section 3, a classification of architectures of modern parallel database systems is given. Section 4 is devoted to the comparative analysis of various parallel architectures of database systems, which relies on a certain set of requirements. The last section gives summary of the basic results obtained and conclusions, as well as discusses directions of future researches.

2. VIRTUAL PARALLEL DATABASE MACHINES

A *virtual parallel database machine* is constructed from the following *virtual devices*: virtual processors, virtual memory modules, virtual disks, and virtual communication network.

A *virtual processor* is a virtual device used for performing a separate task defined as a database process. A typical example of a database process is a query or a query agent (if a partitioned parallelism is used). In a real system, a virtual processor may be represented by

a microprocessor or a processor module. If several database processes are executed on one physical processor in the time-sharing mode, this physical processor is said to implement several virtual processors.

A *virtual memory module* is a virtual device used for buffering database objects. A typical example of an object of a relational database is a relation or its fragment (if partitioned parallelism is used). A virtual processor can access an object of a database only through its image loaded into some virtual memory module accessible to this processor. In accordance with this, the number of virtual memory modules in a virtual parallel database machine cannot exceed the number of virtual processors. In a real system, virtual memory modules are usually implemented as physical modules of operative memory. Note that one physical memory module may be represented by several virtual memory modules (see, for example, [9]), and, vice versa, several physical memory modules can be considered as one virtual memory module (see, for example, [10]).

A *virtual disk* is a virtual device used for storing database objects. In a real system, a virtual disk is usually implemented as a physical disk device or an array of disks [11].

A *virtual communication network* is a virtual device providing data transfers from one virtual memory module to another. The transfers are implemented only by means of communicative actions of the corresponding virtual processors. Without loss of generality, we may assume that a virtual parallel database machine has not more than one virtual communication network. Note that, if a virtual machine has only one memory module, than this machine has no virtual network.

A *virtual parallel database machine* is defined as a connected graph whose nodes correspond to various virtual devices and edges, to dataflows. An example of a virtual parallel database machine configuration is shown in Fig. 3.

It should be noted that, in the given context, the virtual database machine is just some abstraction level in the system hierarchy of program modules of the DBMS and operating system implementing the database system on a particular hardware platform. Note that, in the

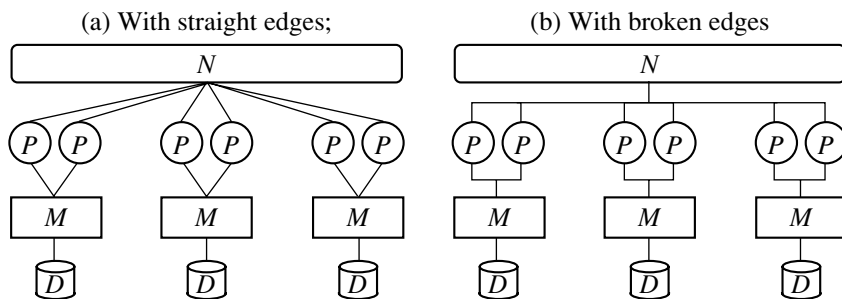


Fig. 3. An example of a configuration of the virtual parallel database machine. Here *P* is a virtual processor, *M* is a virtual memory module, *D* is a virtual disk, and *N* is a virtual communication network.

framework of one system, we can consider several abstraction levels of this kind. The virtual machine of each current level is implemented on the basis of functions and services provided by the virtual machine of the previous level. To better explain this, we consider an illustrative example shown in Fig. 4. Here, the system hierarchy is based on the virtual machine V_0 that has a certain hardware–software implementation. The hardware platform of V_0 is actually a union of several separate computers. In this case, a UNIX/Linux system loaded in each node might serve as the operating system, and MPI, as the system integration means.

Using the virtual machine V_0 , we can create a program complex I_1 implementing a virtual disk with the storage equal to the free disk space of all physical system disks. In this case, I_1 is said to implement a virtual machine V_1 in which each processor has its own private memory, but all processors share common disk space. In other words, I_1 defines a mapping of V_0 onto V_1 . Similarly, in the environment V_1 , we can create a program complex I_2 implementing common virtual memory of a size equal to the total free address space of all physical memory modules. As a result, we construct a mapping from the configuration V_1 onto the configuration V_2 in which all processors share common memory and common disk space. Having this done, we can complete the DBMS implementation in the context of the configuration V_2 . The database system obtained has a hybrid architecture in the following sense: on a physical (zero) level, it is a system without shared resources; on a logical (first) level, it is a system with shared disks; and, on the virtual (second) level, we have a system with shared memory and disks. Of course, the configuration depicted in Fig. 4 is not viable because of difficulties associated with the inefficiency of accessing such a virtual memory and virtual disk. However, in what follows, we give examples of hybrid configurations that can be used in real parallel database systems.

In conclusion, it should be noted that we use the term “virtual machine” together with the term “database” with the only purpose to restrict all variety of possible combinations and to exclude from the consideration the architectures that are not adequate from the database standpoint. These architectures, however, may occur quite adequate for other problems.

3. CLASSIFICATION OF ARCHITECTURES OF PARALLEL DATABASE SYSTEMS

The classification of architectures of parallel database systems serves as a methodological basis for many studies related to databases. Until recently, the taxonomic approach suggested by M. Stonebraker was used for these purposes. In this section, we revise the Stonebraker taxonomy and describe some extensions of this classification, which are based on a concept of virtual database machines.

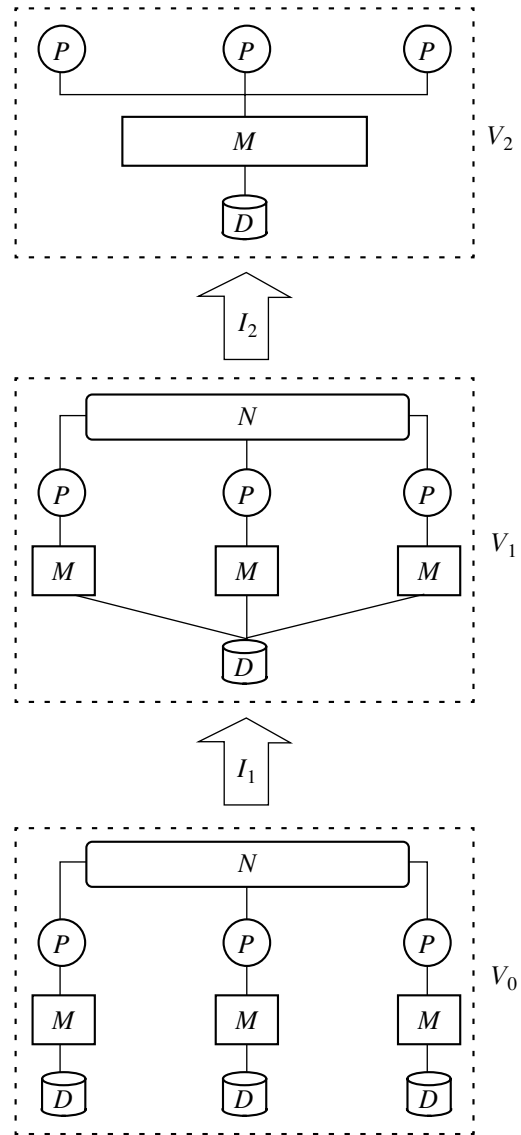


Fig. 4. Example illustrating the hierarchy of virtual database machines.

3.1. Stonebraker Classification

The most popular approach to classifying parallel database systems is that suggested by M. Stonebraker [7]. The Stonebraker classification is shown schematically in Fig. 5. Here, P denotes processors, M stands for the operative memory module, D is a disk device, and N is the communication network.

According to the Stonebraker classification, parallel database systems can be divided into the following three basic classes depending on the way the hardware resources are shared:

- (1) *SE (shared-everything)* architectures with shared memory and disks (Fig. 5a),
- (2) *SD (shared-disks)* architectures with shared disks (Fig. 5b),

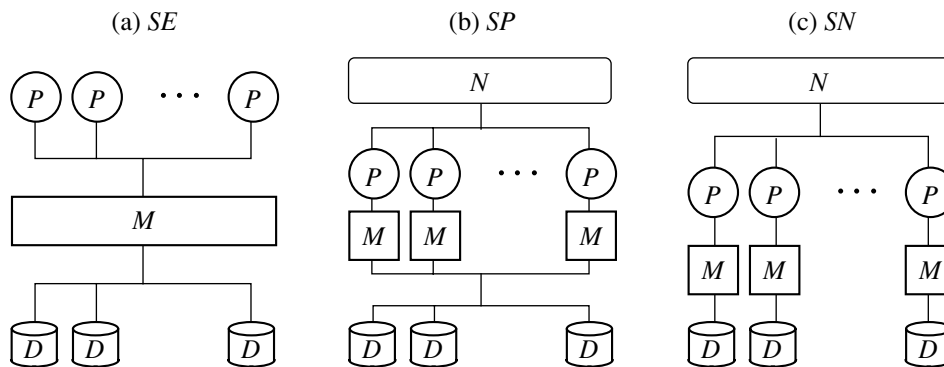


Fig. 5. The Stonebraker classification.

(3) *SN* (*shared-nothing*) architectures without shared resources (Fig. 5c).

The *SE* architecture (in [7], this architecture is referred to as the shared-memory architecture) includes database systems in which each processor has access to any disk (with the same access time) through the shared memory (Fig. 5a). The interprocessor communications in the *SE* systems are implemented through the use of the shared memory. The access to the disks in the *SE* systems is implemented usually through a common buffer pool. It should be emphasized that each processor in an *SE* system has its own cache memory.

There exist many parallel database systems with the *SE* architecture. In fact, all leading commercial DBMSs have implementations based on the *SE* architecture. One of the first examples of porting a single-processor system to the *SE* architecture is the implementation of DB2 on IBM3090 with six processors [12]. However, it should be noted that the majority of the commercial *SE* systems make use of only intertransaction parallelism (i.e., the intra-transaction parallelism is lacking). Nevertheless, several prototype *SE* systems that use the intra-transaction parallelism, such as XPRS [14], DBS3 [15], and Volcano [16], have already been created.

The *SD* architecture (Fig. 5b) includes database systems in which each processor has access to any disk; however, each processor has its own private memory [17]. The processors in such systems are connected with each other through a high-speed network to make it possible to transfer data. Examples of parallel database systems with the *SD* architecture are IBM IMS [18], Oracle Parallel Server [19] on nCUBE [20] and VAXclusters [21], IBM Parallel Sysplex [22], and others.

In the *SN* architecture (Fig. 5c), each processor has its own memory and its own disk. As in the *SD* systems, the processors are connected with each other through a high-speed network, which makes it possible to organize message exchange between the processors [2]. Currently, there exist many prototype systems and several commercial systems with the *SN* architecture, which use partitioned parallelism. For the examples of the prototype *SN* systems, we can cite the following

ones: ARBRE [23], BUBBA [24], EDS [25], GAMMA [26], KARDAMOM [27], and PRISMA [28]. Examples of the commercial systems with the *SN* architecture are NonStop SQL [29], Informix PDQ [30], NCR/Teradata DBC [31], IBM DB2 PE [32], and others.

3.2. Extension of the Stonebraker Classification

The Stonebraker classification was used in many works devoted to the analysis of architectures of parallel database systems (see, for example, [2, 32–35]). However, this classification is currently considered obsolete and inadequate. *Two arguments* questioning the adequacy of the Stonebraker classification are as follows ([8]):

(I) the Stonebraker classification does not cover all variety of the existing architectures;

(II) the classification based on sharing hardware resources is not appropriate for classifying architectures of modern parallel database systems.

The *first argument* is based on the fact that there appeared multiprocessor systems that combine features of both the *SE* and *SN* architectures [8, 34–37]. To describe such systems, Copeland and Keller [38] suggested extending the Stonebraker classification by introducing the following two additional classes of architectures of parallel database machines (Fig. 6):

- the *CE* (*clustered-everything*) architecture with *SE* clusters joined on the basis of the *SN* principle (Fig. 6d);
- the *CD* (*clustered-disk*) architecture with *SD* clusters joined on the basis of the *SN* principle (Fig. 6e) (note that the boundaries of the *SD* clusters in Fig. 6 are extended to the common (global) communication network, since they can include their own (local) communication networks),

These architectures are also referred to as *hierarchical architectures* [39]. Figure 6 shows two-level hierarchies. However, the classification approach suggested by Copeland and Keller can easily be extended to architectures with three or more hierarchy levels. An example of a three-level hierarchical architecture is CD_2 (Clustered-Disk with 2-processor modules) architec-

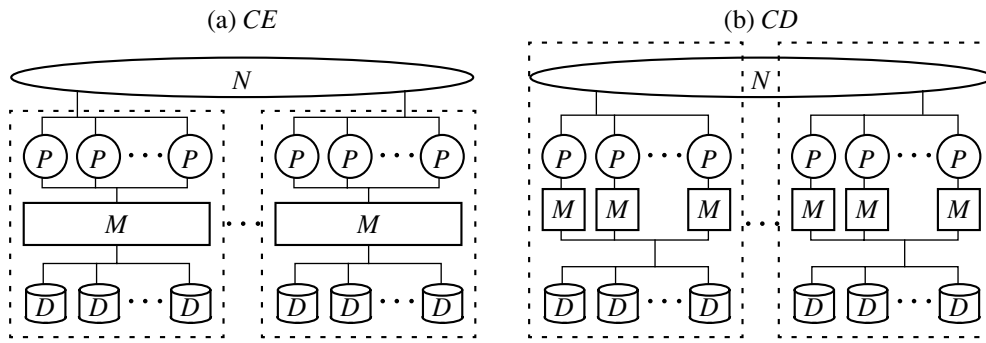


Fig. 6. Extension of the Stonebraker classification.

ture, which was used in designing the parallel database system Omega [40, 41]. Two-level hierarchical architectures were studied in a number of works (see, for example, [34, 37, 38, 42, 43]). Three-level hierarchical architectures almost have not been studied by now.

The *second argument* is related to the fact that modern multiprocessor systems, as a rule, have hardware components of complicated structure and combine properties of architectures of different classes. Examples of such systems are Russian multiprocessor systems from MBC-100/1000 series [6], multiprocessor systems SP2 [44] manufactured by IBM, computers based on the ServerNet technology by Tandem [45], and others. Indeed, the Stonebraker classification turns out inadequate as applied to parallel database systems implemented on the platforms of this kind if the classification approach shown in Fig. 1 is used. However, when using the classification approach based on the introduction of an intermediate notion of the virtual parallel database machine (Fig. 2), the criterion of the resource sharing may still serve as an adequate basis for classifying architectures of modern parallel database systems. Note that we can consider a hierarchy of virtual machines: each current machine is a platform for the implementation of the previous one. For an example of such hybrid architectures, we consider the *CDN* architecture of parallel database systems described in [40, 41]. This architecture is based on the approach suggested by E. Rahm in [46]. The *CDN* architecture is constructed as a set of one-type *SD* clusters combined on the basis of the *SN* principle. A distinctive feature of this system architecture is that the *SD* clusters on the upper levels of the system hierarchy are viewed as *SN* systems (Fig. 7). This manifests itself in that, to each processor node, a separate disk is logically assigned. Such an approach makes it possible to avoid difficulties associated with the implementation of the global blocking table and support of the cache coherence, which are typical of the *SD* systems [35], and, simultaneously, to take advantage of the possibilities of the *SD* architecture for load balancing. A similar approach has been used in the development of the parallel database system NonStop SQL/MP [29].

4. COMPARATIVE ANALYSIS OF ARCHITECTURES OF PARALLEL DATABASE SYSTEMS

In this section, we compare various parallel architectures of database systems on the basis of the taxonomic approach discussed in Section 3. To compare different architectures, it is required to formulate requirements to parallel database systems.

4.1. Requirements for Parallel Database Systems

The criteria used in the comparison of architectures of parallel database systems rely on the following set of requirements [2, 7, 35]:

- (1) good scalability,
- (2) high data availability,

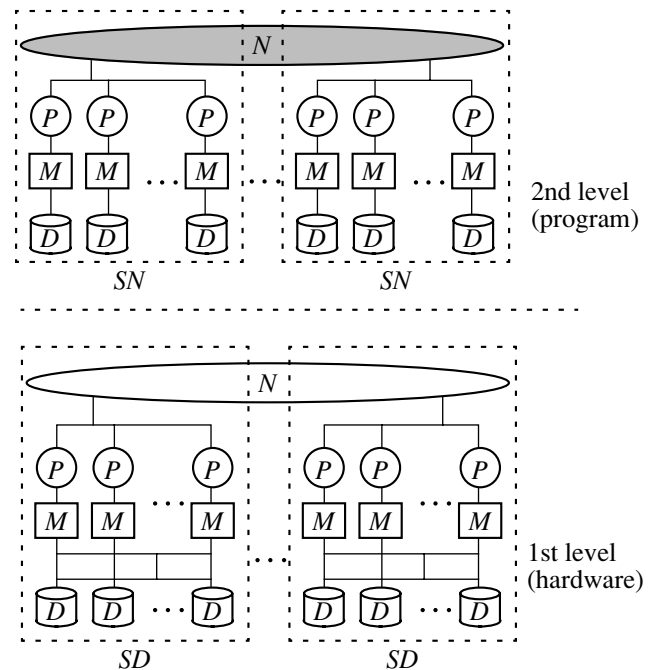


Fig. 7. Hybrid CDN architecture.

Comparison of architectures

	<i>SN</i>	<i>CE</i>	<i>CD</i>	<i>CDN</i>
Scalability	2	3	3	3
Data availability	2	1	3	3
Load balancing	0	2	1	1
Interprocessor communications	0	2	1	1
Cache coherence	3	2	0	3
Concurrency control	3	2	0	3
Sum of points	10	12	8	14

- (3) efficient load balancing,
- (4) low cost of interprocessor exchanges,
- (5) low overheads on ensuring cache coherence,
- (6) efficient organization of the concurrency control.

Let us consider the specified criteria in more detail.

Scalability. The possibility of dynamic buildup to adapt to a growing database size or increasing performance requirements is an important property of parallel platforms [35]. This feature is achieved by gradually incorporating additional processors, memory modules, disks, and other hardware components into the system. This process is referred to as system scaling. If the hardware capacity of the system doubles, its performance is expected to double as well. However, in practice, a real increase in the performance is usually much lower. For example, the scalability of the *SE* systems is limited to 20–30 processors [35]. With the further enhancement of an *SE* system, its performance grows very slowly or even starts to fall [34]. This is explained by the fact that the processors spend much time waiting for an access to the shared resources. Hence, the scalability of any multiprocessor system is determined by the parallelization efficiency.

The parallelization efficiency is described in terms of two basic qualitative characteristics: *speedup and scaleup* [2]. The architecture of a multiprocessor system is considered to be nicely scalable if it demonstrates almost linear scaleup and speedup. *Linear scaleup* implies that the time spent by the system for solving a problem is equal to that spent by a double system for solving a double problem. *Linear speedup* implies that a double system solves a problem twice as fast as the original system. The main factor that worsens the scalability of systems results from drawbacks associated with the concurrent access of shared resources by the processors.

Data availability. One of the critical characteristics of parallel database systems is the capability of the system to ensure a high degree of data availability under the condition of failures of some hardware components. The probability of a hardware failure in a one-processor system is not great. However, in a system with thousands of processor nodes, this probability increases

thousandfold. Therefore, the problem of ensuring high data availability in multiprocessor systems is of great importance.

The database *availability coefficient* can roughly be defined as the ratio of the time during which the database was available for the users to that during which the users tried to access the database. For example, if the users needed the access to the database during eight hours a day, but the database was actually available only for six hours, the availability coefficient is equal to $6/8 = 0.75$ during an 8-hour period. A highly available database system can be defined as a system accepting users' queries 24 hours a day with the availability coefficient not less than 0.99 [47].

Hardware fault-tolerance is the basic factor ensuring high data availability in parallel database systems with a large number of processor nodes. The hardware fault-tolerance is meant to be the retention of the system efficacy under single failures of hardware components, such as a processor, memory module, disk, or links [47]. In particular, a single failure of any device must not result in the loss of the database integrity, to say nothing of a physical loss of any part of the database.

Load balancing. The balancing of the processor load is one of the key problems in ensuring high efficiency of the parallel query processing. The DBMS should divide a query into parallel agents and distribute them among the processors to ensure uniform loading of all processors. The problem of load balancing is especially important in the case where partitioned parallelism is used [2]. The important factor affecting the efficiency of the parallelization of relational operations (especially, join and sort operations) is the value of the skew in data to be processed. It has been shown that, in real databases, some values of a certain attribute occur more frequently than others [48–50]. In particular, Lynch [49] notes that the values of text attributes are usually distributed in accordance with the Zipf law [51]. Such non-uniformity is said to be the attribute value skew [52]. Lakshmi and Yu [53] showed that, in the presence of data skew, the speedup of the parallel execution of the join operation may be extremely low because of an overload of some processors and underload of others.

Interprocessor communications. If the partitioned parallelism is used, the interprocessor communications in parallel database systems can generate considerable traffic [35]. This is explained by the fact that, upon parallel execution of the operation of joining two relations, we have either to dynamically fragment anew the original relations by the join attribute or to send the “alien” tuples from one processor node to another. Both actions are associated with sending considerable amounts of data through the communication network. Therefore, the cost of the interprocessor exchanges may critically affect the total system performance.

Cache coherence. When a common disk pool is shared by several processors, we face the so-called *cache coherence problem* [17]. The essence of this problem is as follows. After a transaction addresses a disk page, the image of this page remains for some time in the buffer associated with the given processor node. Hence, one processor node may revoke changes made by the other processor node. To avoid this, any time when the disk page is accessed, we need to check whether the image of this page is contained in the buffer pools (caches) of other processor nodes and, if this takes place, *coordinate* changes produced in the caches of these processor nodes.

When the common operative memory is shared by several processors, we encounter a similar problem with the content of the processor cache memory [54]. True, in the latter case, the problem is usually solved on the hardware–microprogram level. In any case, the ensuring of the cache coherence requires additional overheads, which can be considerable in database systems [17].

Concurrency control. Another series problem for database systems with shared disks is the support of the global lock table [55]. The locking is one of the basic methods used for ensuring ACID properties of the transactions [56]. If different processor nodes work concurrently with the same database objects, they must have an access to the common (*global*) lock table. The support of such global lock table in multiprocessor systems without shared memory can be associated with great overheads [55].

4.2. Comparative Analysis of Architectures of Parallel Database Systems

The comparative analysis of the *SE*, *SD*, and *SN* architectures was done by Stonebraker and can be found in the classical work [7]. This analysis showed that, from the standpoint of scaleable high-performance database systems, the *SN* architecture is most preferable among these three architectures.

In this section, we compare four different architectures of parallel database systems using the criteria summarized in the table. These criteria follow immediately from the requirements for parallel database systems formulated in Section 4.1 and are graded on a four-point basis: 0 (“unsatisfactory”), 1 (“satisfactory”), 2 (“good”), and 3 (“excellent”).

Scalability. The *SN* architecture is characterized by the good scalability (2 points). This is associated with the fact that, in the case of many processor nodes, the interprocessor communication network becomes a bottleneck [8, 17]. The *CD*, *CE*, and *CDN* architectures demonstrate better scalability (3 points) owing to the fact that most of the communications occur inside the clusters, thus unloading the intercluster network.

Data availability. The *SN* architecture is characterized again as a good one (2 points). This is explained by

the fact that the backup copies in an *SN* system should be partitioned to many nodes [57] in order that to make the backup copy of a failed disk available in the parallel mode (otherwise, there may arise a serious disbalance in the loading). The support of the coherence of the partitioned backup copies requires certain overheads associated, first of all, with sending large amounts of data through the communication network. The data availability in the *CE* architecture is classified as satisfactory (1 point) because of the low hardware fault-tolerance of the *SE* cluster. Indeed, the failure of practically any hardware component of an *SE* system leads to the failure of the whole system [54]. The *CD* and *CDN* architectures demonstrate better data availability (3 points) owing to the fact that all problems related to ensuring high data availability can efficiently be solved at the level of separate *SD* clusters [40].

Load balance for the *SN* architecture is a serious problem, since the *SN* systems are very sensitive to the data skew [53]. Therefore, the corresponding grade of the *SN* architecture is 0. The hierarchical *CE*, *CD*, and *CDN* architectures make it possible to get better load balance since the load is balanced at two—intercluster and intracluster—levels. The *SD* clusters are characterized by satisfactory load balancing owing to the fact that all disks are available for all processors. Accordingly, the *CD* and *CDN* architectures get 1 point. The best load balance among the considered architectures is achieved in *SE* clusters, since, in addition to the disks, the entire operative memory is available for all processors [35]. In accordance with this, the load balancing for the *CE* architecture is estimated as good (2 points). We do not give the *CE* architecture the highest grade because the problem of balancing the load between separate clusters remains relevant for the *CE* systems.

Interprocessor communications. The high cost of interprocessor communications is a weak point of the *SN* architecture [7, 29] (0 points). The use of the *CE* architecture makes it possible to considerably reduce the overheads associated with the interprocessor communications [34] since the interprocessor communications on the *SE* cluster level can efficiently be implemented through shared memory. Therefore, in terms of this criterion, the *CE* architecture gets 2 points. The *CD* and *CDN* architectures are behind the *CE* architecture in terms of this criterion; however, they may outperform the *SN* architecture, since, potentially, the intracluster communications can be implemented more efficiently than the intercluster communications [58, 59]. Accordingly, we give 1 point each to both *CD* and *CDN* architecture.

Cache coherence is a serious problem for the *CD* architecture, since, in an *SD* cluster, the same pages of the shared disks are buffered in the private memory modules (we give the lowest grade, 0 points). The *CE* architecture is better than the *CD* architecture in terms of this parameter, since the *SE* clusters use a common buffer pool in the shared memory. However, the *CE*

architecture is behind the *SN* architecture in terms of this criterion since, in the *SE* clusters, it is required to support data coherence in the private processor caches [54]. Hence, the *SE* architecture gets only 2 points. The *CDN* architecture is free of this disadvantage since, on the logical (program) level, it has no shared resources. Therefore, the *SN* and *CDN* architectures get the highest grade (3 points).

Concurrency control. Another serious problem inherent in the *CD* architecture is related to difficulties associated with the organization of the database object locking by the concurrent transactions accessing them. In an *SD* cluster, it is required to support a copy of the global lock table in each processor node, which may require considerable overheads [55]. Therefore, the *CD* architecture gets 0 points here as well. The *CE* architecture is, basically, free of this shortcoming, since the only copy of the global lock table for the *SE* cluster is stored in the shared operative memory (2 points). In the *SN* systems, there is no need to support the global lock table just because no resources are shared. Therefore, the *SN* architecture is the best in terms of this parameter (3 points). The *CDN* architecture fully inherits this feature from the *SN* architecture (also 3 points).

Conclusion. Based on the above analysis and taking into account the sum of the grades for different criteria shown in the table, we may conclude that the *CD* architecture in the pure form is not appropriate. The *CE* architecture looks more attractive than the *SN* architecture. However, if we take into account the entire collection of the requirements to parallel database systems considered in Section 4.1, we can see that the *CDN* architecture is the best one. We used the *CDN* architecture in the design of a prototype parallel database system Omega based on the Russian multiprocessor complex MBC-100/1000. Numerous experiments carried out with this prototype substantiate the conclusions made in this paper (see [40, 41]).

6. CONCLUSION

In this paper, we have introduced the notion of a virtual parallel database machine, which is an effective tool in designing complex hybrid parallel database systems. This tool makes it possible to define different levels of abstraction when designing system architecture. The design stages represent a hierarchy of virtual machines, such that each current machine implements the previous one.

For the sake of classification and comparison of architectures of modern parallel database systems, we have analyzed the Stonebraker classification suggested in the mid-1980s. This classification has been shown to remain the most appropriate one for database systems; however, it needs certain refinement and extension. In this paper, we have refined and extended the Stonebraker classification by way of using a virtual parallel

database machine abstraction and introducing additional classes of hierarchical cluster architectures.

Further, we have formulated and considered basic requirements to parallel database systems. Based on these requirements and on the extended Stonebraker classification, we have carried out comparative analysis of modern architectures of parallel database systems. This analysis has revealed that the hybrid *CDN* architecture, whose implementation is described in [40, 41], has the best performance. The *CDN* architecture was employed in the prototype parallel database system Omega designed for the Russian multiprocessor computing system MBC-100/1000. Experiments carried out on the basis of the Omega system substantiate the conclusions of this paper.

In terms of further studies, of most interest are the following problems:

(1) Experimental studies of different cluster configurations in systems with the *CDN* architecture. It is planned to develop a program that emulates the operation of a database system with the *CDN* architecture and, using it, to carry out computational experiments on studying efficiency of query parallelizing in the OLTP and OLAP modes for various topologies of intracluster interprocessor connections.

(2) Development of algorithms performing relational operations that take into account specific features of the *CDN* architecture in some optimal way.

(3) Development of methods for optimizing parallel queries designed for database systems with the *CDN* architecture.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, project no. 03-07-90031.

REFERENCES

1. Ozsu, M.T. and Valduriez, P., *Principles of Distributed Database System*, Englewood Cliffs: Prentice-Hall, 1991.
2. DeWitt, D.J. and Gray, J., Parallel Database Systems: The Future of High-Performance Database Systems, *Commun. ACM*, 1992, vol. 35, no. 6, pp. 85–98.
3. Voevodin, V.I. and Kapitonova, A.P., *Metody opisaniya i klassifikatsii arkhitektur vychislitel'nykh sistem*, (Methods of Description and Classification of Architectures of Computing Systems), Moscow: Mosk. Gos. Univ., 1994.
4. Flynn, M.J. and Rudd, K.W., Parallel Architectures, *ACM Computing Surv.*, 1996, vol. 28, no. 1, pp. 67–70.
5. Dasgupta, S.A., Hierarchical Taxonomic System for Computer Architectures, *IEEE Comput.*, 1990, vol. 23, no. 3, pp. 64–74.
6. Korneev, V.V., *Parallel'nye vychislitel'nye sistemy*, (Parallel Computing Systems), Moscow: Nolidzh, 1999.
7. Stonebraker, M., The Case for Shared Nothing, *Database Eng. Bull.*, 1986, vol. 9, no. 1, pp. 4–9.

8. Norman, M.G., Zurek, T., and Thanisch, P., Much Ado about Shared-Nothing, *ACM SIGMOD Record*, 1996, vol. 25, no. 3, pp. 16–21.
9. Carr, J.L. and Hennessy, J.L., WSClock—A Simple and Effective Algorithm for Virtual Memory Management, *Proc. of the Eighth Symp. on Operating System Principles*, (Asilomar Conf. Grounds, Pacific Grove, 1981), New York: ACM, 1981, pp. 87–95.
10. Amza, C. *et al.*, ThreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Comput.*, 1996, vol. 29, no. 2, pp. 18–28.
11. Patterson, D.A., Gibson, G.A., and Katz, R.H., A Case for Redundant Arrays of Inexpensive Disks (RAID), *Proc. 1988 ACM SIGMOD Int. Conf. on Management of Data* (Chicago, 1988), ACM, 1988, pp. 109–116.
12. Cheng, J.M., *et al.*, IBM Database 2 Performance: Design, Implementation, and Tuning, *IBM Systems J.*, 1984, vol. 23, no. 2, pp. 189–210.
13. Davison, W., Parallel Index Building in Informix OnLine 6.0, *Proc. of the 1992 ACM SIGMOD Int. Conf. on Management of Data* (San Diego, 1992), ACM, 1992, p. 103.
14. Stonebraker, M., Katz, R.H., and Patterson, D.A., and Ousterhout, J.K., The Design of XPRS, *Fourteenth Int. Conf. on Very Large Data Bases*, (Los Angeles, 1988), Morgan Kaufmann, 1988, pp. 318–330.
15. Bergsten, B., Couprie, M., and Lopez, M., DBS3: A Parallel Data Base System for Shared Store (Synopsis), in *Issues, Architectures, and Algorithms* (Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems (PDIS 1993), San Diego, 1993), IEEE Comput. Soc., 1993, pp. 260–262.
16. Graefe, G., Volcano—An Extensible and Parallel Query Evaluation System, *IEEE Trans. Knowledge Data Engineering*, 1994, vol. 6, no. 1, pp. 120–135.
17. Rahm, E., Parallel Query Processing in Shared Disk Database Systems, *ACM SIGMOD Record*, 1993, vol. 22, no. 4, pp. 32–37.
18. Strickland, J.P., Uhrowczik, P.P., and Watts, V.L., IMS/VS: An Evolving System, *IBM Systems J.*, 1982, vol. 21, no. 3, pp. 490–510.
19. Linder, B., Oracle Parallel RDBMS on Massively Parallel Systems, in *Issues, Architectures, and Algorithms* (Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems (PDIS 1993), San Diego, 1993), IEEE Comput. Soc., 1993, pp. 67–68.
20. Dubova, N., Supercomputers nCube, *Otkrytye sistemy*, 1995, no. 2, pp. 42–47.
21. Kronenberg, N.P., Levy, H.M., and Strecker, W.D., VAXclusters: A Closely-Coupled Distributed System, *ACM Trans. Comput. Systems*, 1986, vol. 4, no. 2, pp. 130–146.
22. Nick, J.M., Moore, B.B., Chung, J.-Y., and Bowen, N.S., S/390 Cluster Technology: Parallel Sysplex, *IBM Systems J.*, 1997, vol. 36, no. 2, pp. 172–201.
23. Lorie, R., *et al.*, Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience, *Data Engineering Bull.*, 1989, vol. 12, no. 1, pp. 2–8.
24. Boral, H., Alexander, W., Clay, L., Copeland, G., Sanforth, S., Franklin, M., Hart, B., Smith, M., and Valduriez, P., Prototyping Bubba: A Highly Parallel Database System, *IEEE Trans. Knowledge Data Eng.*, 1990, vol. 2, no. 1, pp. 4–24.
25. Skelton, C.J., *et al.*, EDS: A Parallel Computer System for Advanced Information Processing, *Lecture Notes in Computer Science* (Proc. of the 4th Int. PARLE Conf., Paris, 1992), Springer, 1992, vol. 605, pp. 1–18.
26. DeWitt, D.J., *et al.*, The Gamma Database Machine Project, *IEEE Trans. Knowledge Data Eng.*, 1990, vol. 2, no. 1, pp. 44–62.
27. Von Bultzingsloewen, G., *et al.*, KARDAMON—A Dataflow Database Machine for Real-Time Applications, *SIGMOD Record*, 1988, vol. 17, no. 1, pp. 44–50.
28. Apers, P.M.G., van den Berg, C.A., Flokstra, J., Grefen, P.W.P.J., Kersten, M.L., and Wilschut, A.N., Prisma/DB: A Parallel Main-Memory Relational DBMS, *IEEE Trans. Knowledge Data Eng.*, 1992, vol. 4, no. 6, pp. 541–554.
29. Englert, S., Glasstone, R., and Hasan, W., Parallelism and Its Price: A Case Study of NonStop SQL/MP, *ACM SIGMOD Record*, 1995, vol. 24, no. 4, pp. 61–71.
30. Clay, D., Informix Parallel Data Query (PDQ), in *Issues, Architectures, and Algorithms* (Proc. of the 2nd Int. Conf. on Parallel and Distributed Information Systems (PDIS 1993), San Diego, 1993), IEEE Comput. Soc., 1993, pp. 71–72.
31. Page, J., A Study of a Parallel Database Machine and Its Performance: The NCR/Teradata DBC/1012. Advanced Database Systems, *Lecture Notes in Computer Science* (Proc. of the 10th British Natl. Conf. on Databases. BNCOD 10, Aberdeen, 1992), Springer, 1992, vol. 618, pp. 115–137.
32. Baru, C.K., *et al.*, DB2 Parallel Edition, *IBM System J.*, 1995, vol. 34, no. 2, pp. 292–322.
33. Bergsten, B., Couprie, M., and Valduriez, P., Overview of Parallel Architectures for Databases, *Comput. J.*, 1993, vol. 36, no. 8, pp. 734–740.
34. Hua, K.A., Lee, C., and Peir, J.-K., Interconnecting Shared-Everything Systems for Efficient Parallel Query Processing, *Proc. First Int. Conf. on Parallel and Distributed Information Systems (PDIS 1991)* (Miami Beach, 1991), IEEE-CS, 1991, p. 262–270.
35. Valduriez, P., Parallel Database Systems: The Case for Shared-Something, *Proc. of the 9th Int. Conf. on Data Eng.* (Vienna, 1993), IEEE Comput. Soc., 1993, pp. 460–465.
36. Ballinger, C. and Fryer, R., Born to Be Parallel: Why Parallel Origins Give Teradata an Enduring Performance Edge, *IEEE Data Eng. Bull.*, 1997, vol. 20, no. 2, pp. 3–12.
37. Pramanik, S. and Tout, W.R., The NUMA with Clusters of Processors for Parallel Join, *IEEE Trans. Knowledge Data Eng.*, 1997, vol. 9, no. 4, pp. 653–666.
38. Copeland, G.P. and Keller, T., A Comparison of High-Availability Media Recovery Techniques, *Proc. of the 1989 ACM SIGMOD Int. Conf. on Management of Data* (Portland, 1989), ACM, 1989, pp. 98–109.
39. Graefe, G., Query Evaluation Techniques for Large Databases, *ACM Computing Surv.*, 1993, vol. 25, no. 2, pp. 73–169.
40. Sokolinsky, L.B., Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture, *Programmirovaniye*, 2001, no. 6, pp. 13–29.
41. Sokolinsky, L.B., Design and Evaluation of Database Multiprocessor Architecture with High Data Availabil-

- ity, *Proc. of the 12th Int. DEXA Workshop* (Munich, 2001), IEEE Comput. Soc., 2001, pp. 115–120.
42. Bouganim, L., Florescu, D., and Valduriez, P., Dynamic Load Balancing in Hierarchical Parallel Database Systems, *Proc. 22th Int. Conf. on Very Large Data Bases (VLDB'96)* (Mumbai, India, 1996), Morgan Kaufmann, 1996, pp. 436–447.
 43. Xu, Y. and Dandamudi, S.P., Performance Evaluation of a Two-Level Hierarchical Parallel Database System, *Proc. Int. Conf. Computers and Their Applications*, Tempe, Arizona, 1997, pp. 242–247.
 44. Shmidt, V., IBM SP2 Systems, *Otkrytye sistemy*, 1995, no. 6, pp. 53–60.
 45. Shnitman, V., Fault-Tolerant Servers ServerNet, *Otkrytye sistemy*, 1996, no. 3, pp. 5–11.
 46. Rahm, E., Framework for Workload Allocation in Distributed Transaction Processing Systems, *J. Systems Software*, 1992, vol. 18, pp. 171–190.
 47. Kim, W., Highly Available Systems for Database Applications, *ACM Computing Surv.*, 1984, vol. 16, no. 1, pp. 71–98.
 48. Christodoulakis, S., Estimating Record Selectivities, *Information Systems*, 1983, vol. 8, no. 2, pp. 105–115.
 49. Lynch, C.A., Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distribution of Column Values, *Proc. of the Fourteenth Int. Conf. on Very Large Data Bases*, (Los Angeles, 1988), Morgan Kaufmann, 1998, pp. 240–251.
 50. Montgomery, A.Y., D'Souza, D.J., and Lee, S.B., The Cost of Relational Algebraic Operations on Skewed Data: Estimates and Experiments, in *Information Processing 83* (Proc. of the IFIP 9th World Comput. Congr., Paris, 1983), North-Holland, 1983, pp. 235–241.
 51. Zipf, G.K., *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Cambridge: Addison-Wesley, 1949.
 52. Walton, C.B., Dale, A.G., and Jenevein, R.M., A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins, *Proc. of the 17th Int. Conf. on Very Large Data Bases* (Barcelona, 1991), Morgan Kaufmann, 1991, pp. 537–548.
 53. Lakshmi, M.S. and Yu, P.S., Effectiveness of Parallel Joins, *IEEE Trans. Knowledge Data Eng.*, 1990, vol. 2, no. 4, pp. 410–424.
 54. Pfister, G., Sizing Up Parallel Architectures, *Database Programming Design OnLine* (<http://www.dbpd.com>), 1998, vol. 11, no. 5.
 55. Mohan, C. and Narang, I., Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment, *Lecture Notes in Computer Science*, (Proc. of the 3rd Int. Conf. on Extending Database Technol., Vienna, 1992), Springer, 1992, pp. 453–468.
 56. Gray, J. and Reuter, A., *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
 57. Hsiao, H.I. and DeWitt, D.J., A Performance Study of Three High Availability Data Replication Strategies, *Distributed Parallel Databases*, 1993, vol. 1, no. 1, pp. 53–80.
 58. Sokolinsky, L.B., Interprocessor Communication Support in the Omega Parallel Database System, *Proc. of the 1st Int. Workshop on Comput. Sci. and Information Technol. (CSIT'99)*, Moscow, 1999.
 59. Sokolinsky, L.B., Operating System Support for a Parallel DBMS with a Hierarchical Shared-Nothing Architecture, *Proc. of the Third East Eur. Conf. "Advances in Databases and Information Systems" (ADBIS'99)* (Maribor, Slovenia, 1999), Maribor: Institute of Informatics, 1999, pp. 38–45.