TASK QUARTERLY 2 No 1 (1998), 107-119

THE ARCHITECTURE OF MODERN DATABASE SYSTEMS

KRZYSZTOF GOCZYŁA

Department of Applied Informatics
Technical University of Gdańsk
Narutowicza 11/12, 80-952 Gdansk, Poland
kris@pg.gda.pl

Abstract: The paper presents major trends in the modern architecture of large database systems. Two types of architecture are described in detail: the parallel architecture and the distributed architecture. It has been widely recognised that centralised, single processor computing systems and centralised database systems in particular are approaching their theoretical limits of performance. Hence we can observe a growing interest among researchers and developers in the design and implementation of highly efficient distributed architecture. The paper focuses on different types of client-server architecture, which nowadays is becoming very popular in data processing systems.

1. What is a database system architecture?

The concept of "system architecture" is not precisely defined in the information technology. Depending on contexts and applications, system architecture may be understood as a hardware configuration, a software configuration or a set of communication protocols between different components of a system. If we consider a system as a whole, the "system architecture" includes all those elements — hardware, software and communication protocols. In the case of a database system, the architecture of a system can have two facets: *logical* architecture and *physical* architecture. This paper focuses mainly on different types of physical architecture of modern database systems. To help to understand the problems of physical architecture, the paper first gives a brief overview of the underlying concepts of logical architecture of a database system.

The first standards of logical architecture of database systems were introduced in the 1970s by a special workgroup called "Study Group on DBMS" within the framework of ANSI/SPARC. The ANSI/SPARC model requires the database systems to be constructed according to three-level logical architecture (Figure 1). The three levels of the architecture are:

- the *external* (or *user*) level,
- the *conceptual* (or *logical*) level,
- the *internal* (or *physical*) level.

The external level is a level from which a user communicates with a database system. This level separates the user from the technical details of the system. At the external level the user can access the database by means of a high-level query language and can develop applications using special development tools. The user does not have to know anything about physical representation of the data stored in the database. Moreover, this level usually restricts privileges that the user has for the access to the data, which provides data security. The data security is obtained by means of user *identification*, *authentication* and by *authorisation* protocols. The security mechanisms can be applied to a single user as well as to whole groups of users. As a result, each user is allowed to operate on some specific portion of data (visible through *views*) in a specific, permissible way. The authorisation is performed during an interactive access to data (e.g. when the user queries a database in an *ad hoc* manner) and during execution of a database application, where database queries are embedded into a high-level language program.

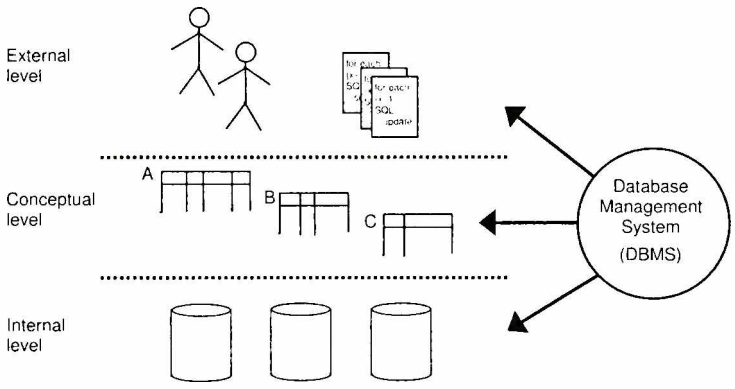

Figure 1. *The three-level logical architecture of a database system*

The conceptual level of a database system defines a *schema* of a database. The database schema is defined according to a *data model* that is common to all the databases in a database system. The most popular *relational* data model defines a database as a set of logically related tables (*relations*) composed of rows (*tuples*) representing elementary data entities. The traditional relational model is presently migrating towards other models specific to so-called *next-generation* database systems. A most prominent example of such new models is the *object-oriented* data model. This model describes a database schema as a set of *classes*, each class representing a collection of similar *objects*. An object is a database entity that encapsulates not only its data, but also its behaviour (functions). The specific data model applied in a given database system is of great importance to the user, as the model strongly influences the query language of the system and other programming languages that are used to develop database applications.

The main task of the internal level of a database system is to store physical data in a reliable and persistent way. At this level there are external storage devices of different kinds, as well as methods of structuring and accessing data files. The patterns of data storage at this level usually differ much from the data representation at the conceptual level and from their formulation in the query (or other high-level) languages. However, differences in data representations do not cause problems to the user who can operate on the database in a convenient way. Due to the postulated isolation of the three levels, it is possible to modify the techniques (devices and access methods) applied at the internal level without changing anything in the database schema and database application programs. This feature of database systems is called *physical data independence* (or just *data independence*). Data independence is probably the most useful feature of modern database systems, and preserving this feature as close as possible is one of the main goals that database systems developers are seeking.

The operation of a whole database system at the three levels is performed by software, called *Database Management System* (DBMS). The main tasks of a DBMS are:

- carrying out the operations submitted at the three levels,
- mapping data structures between the three levels — which is necessary to support data independence,
- providing data security by granting or revoking user privileges and monitoring user operations,
- resolving conflicts during concurrent access to the same pieces of data,
- maintaining the integrity (logical consistency) of data,
- performing recovery after failures.

Commercial DBMS for large database systems are among the most complicated (and expensive) software packages in today's information technology.

Logical architecture of a database system is implemented by physical architecture of the system. Physical architecture of a database system is understood here as implementation of the three levels of logical architecture by different physical components of a computer system. These components are: processors, memory chips, storage devices, communication media etc. The following sections of the paper discuss types of physical architecture whose dominating feature is decentralisation of processing and data. This feature enables database systems to achieve high efficiency, reliability and availability that is required for demanding, mission-critical applications.

2. Parallel database systems

Database systems that exploit parallelism have started to replace traditional centralised systems based on mainframes. This is particularly true in the case of large databases (with capacity of terabytes) and heavy workloads that demand processing

many thousands of on-line transactions per second. In the early 80s, however, the future of parallel database systems seemed rather questionable. Database technology concentrated on the development of highly specialised and highly efficient (but also very expensive) hardware configured into *database machines*. The special hardware devices were, for instance, bubble and CCD memory modules and fixed-head disks of large capacities and short access times. However, the database machines did not fulfil their promises and their development was given up. Such traditional (commodity) devices as semiconductor memory chips, conventional processors and moving-head disks dominate in today's database systems (also parallel systems, see Figure 2).

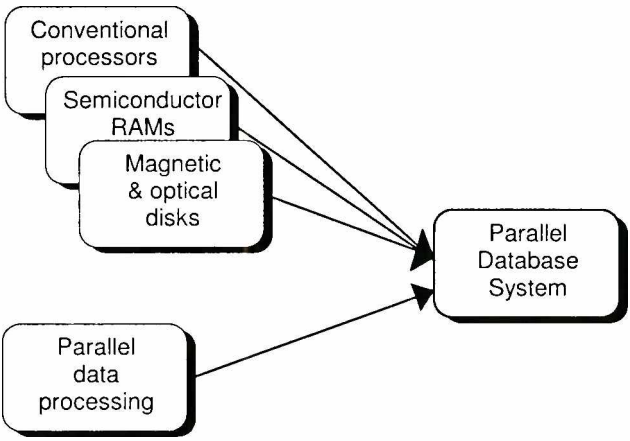

Figure 2. *The components of a parallel database system*

In the mid-80s database systems developers started intensive research in the area of employing parallelism (i.e. "physical concurrency") in the processing of voluminous data. The reasons were two-fold: economical (prices of typical memory devices started to decrease radically, with steadily growing bandwidths) and experimental (the success of some prototypes, like Bubba or Prisma). Soon, first commercial products appeared on the market. The leaders in the development of parallel DBMSs were Teradata and Tandem. There was also a dynamic progress in the development of multiprocessor computers (Intel, nCUBE, NCR etc.). Parallel machines began to achieve performance comparable to (or even higher than) large centralised systems. The main advantages of the parallel systems in comparison with the centralised ones are: lower price and the feature of scaleability (which means that they are able to increase their performance proportionally to the price). The scaleability is a very desired feature as it enables the parallel system to "grow with its owner": the owner of the system can gradually extend the system according to his/her growing needs and financial capabilities.

Nowadays the major database systems manufacturers make their DBMSs capable of operating in a parallel mode on parallel machines. Some popular and commer-

cially available parallel database systems include Oracle versions 7.x or later and Informix-Online versions 7.x or later.

A *parallel database* system can be defined as a system composed of a (usually large) number of simple components that operate in parallel (concurrently) on data partitioned among them. It is essential that the simple components are not self-contained database systems. To compose a fully functional database system, they have to tightly co-operate. That is the basic difference between parallel and distributed database systems. In distributed database systems each component can operate autonomously as a self-contained, fully functional database system, and in this sense each component is independent. In a parallel database system, a single component (a node) performs one strictly defined role in data processing. For example, nodes of a parallel system, called *virtual processors*, can be divided into the following functional groups:

- interface processors — responsible for interaction with a user and for co-ordination of transactions,
- transaction processors that perform the operations on data,
- recovery processors that maintain transaction logs and perform recovery after failures.

Usually, the number of virtual processors can differ from the number of physical processors in a computer. This enables the system to be set up in a flexible way, so that no major configuration changes are necessary when the number of physical processors changes.

It is clear that the quick growth of parallel database systems is related to the popularity of the relational data model. Due to the closure property of the relational algebra, the relational model of data processing can be easily adapted to exploit parallelism. A result of a relational operator is always a relation, so operations performed on a relational database can be mapped onto dataflow graphs with vertices representing relational operators and arcs representing flows of relations. The dataflow graphs can be decomposed into subgraphs, one subgraph per one node of a parallel system (i.e. per one virtual or physical processor). New operators necessary to perform the decomposition are: SPLIT and MERGE. SPLIT performs a split of a single relation, which enables the data of the relation to be partitioned among different nodes; MERGE performs a merge of a partitioned relation, which enables the system to collect the result of a query into one relation, if necessary. Figure 3 illustrates an example of a decomposition of a SELECT (selection of tuples from a relation) followed by a SORT (ordering the result). The exemplary query is:

```
select *
  from data
  where ...
  order by ...
```

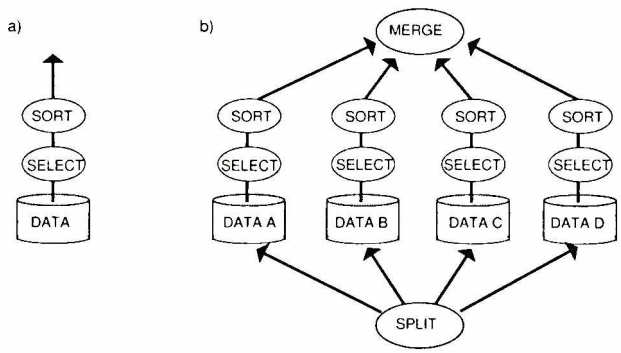

Figure 3. *Data flow graph for an exemplary query: a) no parallelism, b) with parallelism*

The efficiency of the parallel processing in relational database systems depends, among other factors, on how uniformly the data is partitioned among the nodes that participate in a given transaction. The poorer the data uniformity is (or the higher the data *skew* is), the less advantage is gained from the parallelism in a transaction. In an extreme case (at a high data skew), a parallel execution of a query can be slower than a sequential execution of the same query — due to necessary co-ordination costs. Researchers are conducting intensive investigation in the area of the optimal data partition for a given relational query. Also, possibilities of exploiting the parallelism in next-generation database systems (extended relational and object-oriented) is under close investigation.

The most common architecture of parallel database systems is *shared-nothing* architecture (Figure 4). In this kind of architecture each processor is equipped with its own main memory and disk storage. Such a solution minimises both the interference between processors and the traffic volume in the communication medium (mainly, only queries and query results are transmitted over the medium). As a result, *shared-nothing* parallel systems scale up well to hundreds (or even thousands) of processors. The development of *shared-nothing* architecture was possible due to reduction of prices of RAM chips and discs units, accompanied by growing memory densities.

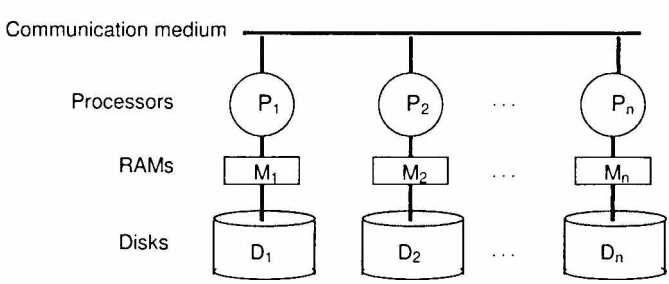

Figure 4. *The shared-nothing parallel architecture*

3. Distributed database systems

Distribution is one of the outstanding features of modern information systems. Distribution in general can be accomplished by: distribution of processing, where many computers work together on behalf of one application; distribution of data, where data to be processed by one application are stored on many computers; and a combination of the two. The development of distribution technology was possible due to the rapid development of network technologies, which enable remote systems to communicate with each other efficiently and reliably. A *distributed database* system is a set of computer nodes interconnected through a communication network. The nodes have the following properties:

1. Each node is a self-contained (autonomous) database system.
2. The nodes established rules of co-operation, so that any user working at one (local) node can access data located at other (*remote*) nodes as if the data were located at one, local node.

A single node of a distributed system usually consists of a computer called *database server* that runs DBMS and stores data, and many *workstations* for users to run their applications and to issue their queries and commands. From the user's point of view, the basic requirement that a distributed database system should meet is *transparency*. The system satisfies this requirement if it enables a user to work with the distributed system in exactly the same way as he/she worked with a centralised system. In other words, the features of distribution should be transparent (hidden) to the user. It is clear that more transparent systems are more convenient for the users as the users do not have to bother about where their data is located and which computer is able to perform their operations. The transparency of distribution manifests itself in several aspects. The most important ones are discussed below.

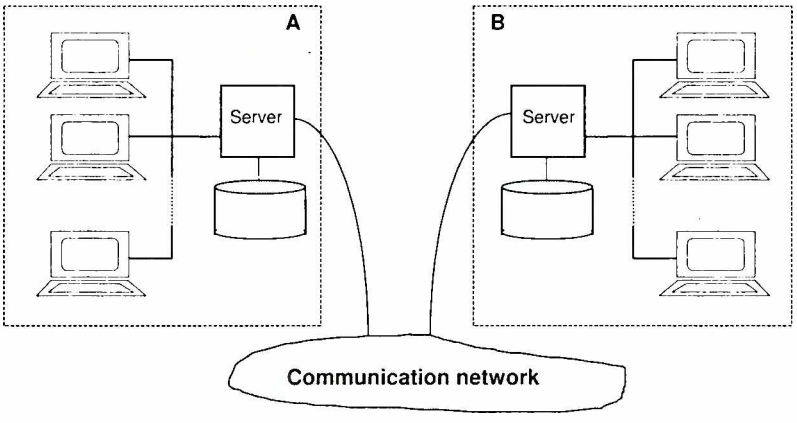

Figure 5. *The layout of a distributed database system*

Location transparency. A user should not need to know which node of a distributed system actually stores the data he/she wants to operate on. The system should be able to locate the data, communicate with the appropriate node(s) and perform necessary transmissions to accomplish the user's request.

Fragmentation transparency. For efficiency reasons, one logical database object (e.g. one relation) may be partitioned into physical parts (called *fragments*) that are stored at different nodes. For example, in a distributed banking system it may be reasonable to partition the accounting database in such a way that each account is stored at the node where it was first created. The fragmentation transparency requires that the user should be able to operate on fragmented data as if the data were not fragmented at all.

Replication transparency. For efficiency reasons, one logical database object may be maintained at the physical level in many copies (*replicas*), each replica being stored at a separate node of a distributed system. The advantage of this solution is that all "read" operations can be directed to the nearest node, which reduces the network traffic and the access time. The main disadvantage of replication is that all updates made to the replicated data must be propagated to all replicas; otherwise the distributed database system looses its logical integrity (enters an inconsistent state). The replication transparency requires that the replication should be invisible to users; in other words, it is the database system, not the user application, that is responsible for propagating updates to all appropriate replicas of modified data.

Let us consider two basic methods of support for data replication, as replication management is one of the most important problems in distributed database systems management. It is clear that the crucial issue is the reliability of the nodes and of the communication network. In case of a serious failure at a node participating in the replication or a failure in the communication network, the immediate propagation of updates may become impossible and the database system as a whole loses (at least temporarily) its logical integrity. Although this problem complicates the implementation and management of the replicas, advanced distributed database systems do support different schemes of data replication. Below two completely different schemes are described; in real-life they can be mixed together into different configurations and variants.

The simplest scheme of data replication is a *read-only* replication (Fig. 6). In this scheme there is one server (a *primary* server) that is responsible for propagating updates to the rest of a distributed system. The primary server sends updated data to *target* servers (i.e. the other servers participating in a replication) where they can be accessed by users in a form of *snapshots*. The snapshots of the replicated data may be updated periodically, continuously or on demand. This scheme is simple because there is always only one source of updates (the primary server) and in case of a node

or the network failure, it is comparatively easy to restore the system integrity.

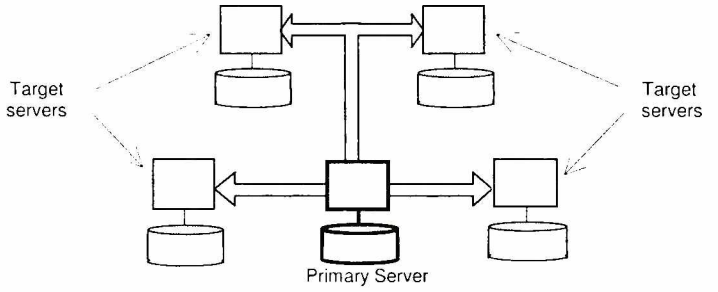

Figure 6. *The read-only replication scheme*

The other replication scheme, much more sophisticated, is an *update-anywhere* (or *symmetric*) replication (Figure 7). In this scheme there is no primary server; all the servers participating in a replication are able to perform any updates of the replicated data and are responsible for propagating the updates to all the replicas residing on other servers in a replication group. One server in each group stores the definition of the group that contains information on the servers participating in a replication and on the data being replicated. Implementation and proper management of an update-anywhere replication scheme is much more complicated than a read-only scheme, so DBMS vendors recommend using a read-only scheme whenever possible. However, some specialised applications may require employing an update-anywhere scheme, particularly when replicated data is updated frequently and when there are many sources of updates. In such workloads, a high overhead caused by heavy communication to a primary server may justify introducing a more complicated, but also a more powerful and flexible update-anywhere scheme.

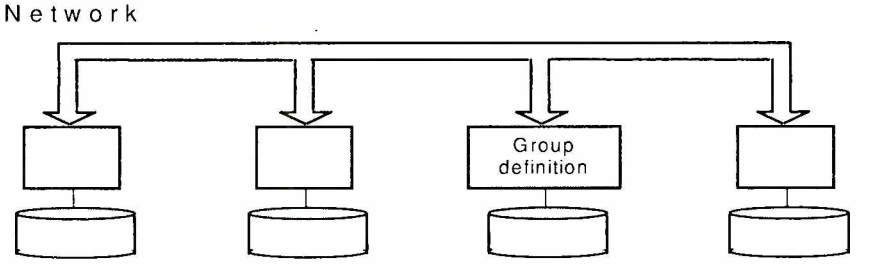

Figure 7. *The update-anywhere replication scheme*

4. The architecture of autonomous database systems

This section describes different kinds of architecture of a database system that is a single node of a distributed system or operates autonomously (is a centralised database

system). The idea of distribution of processing and data can also be exploited in an autonomous database system. The distribution of processing can be obtained in a straightforward manner by assigning DBMS tasks and application tasks to separate computers — a database server and workstations. This configuration is called a *client-server* configuration (Figure 8). The server, running a DBMS, is responsible for the management of shared network resources, including information resources (data from databases stored on the server). To access any shared resource, a client (a workstation) must request an appropriate service from the server. Resources that are local to a workstation (peripherals, storage devices) are inaccessible to other workstations. Advantages of client-server configurations are the following:

- Simplicity in shared resources management and in controlling data security.
- Possibility of centralised optimisation of operations performed on the server.
- The database server is released from performing such cumbersome and time-consuming operations like graphical data presentation and heavy computations; these operations are performed on workstations.

A disadvantage of client-server configurations is that workstation resources are usually poorly utilised. Also, reliability and efficiency requirements for the server are very high, so the server itself must be usually a powerful, multiprocessor machine.

An application that runs in a client-server environment may be decomposed into two separate components: a *front-end*, that operates on the client, and a *back-end*, that runs on the server. The front-end component interacts with the user, sends requests to the server, receives results from the server and displays them to the user. The back-end component performs operations on the data and accesses shared resources. Some vendors (e.g. Oracle, Informix, Progress), together with their DBMSs, provide also user-friendly tools for development of the front-end components of database applications.

Newer (*second-generation*) client-server configurations can accommodate several database servers (run under control of different DBMSs) in one system. It is achieved by conforming to the idea of *open database systems*: i.e. following the ODBC (*Open Database Connectivity*) standards of communication between front-ends and back-ends. Moreover, some parts of application processing (e.g. user-defined procedures) can be delegated for execution from a workstation to a server, so that the traffic in the local network is reduced. In that way the distribution of processing and data is achieved in one autonomous database system. It is expected that increasingly sophisticated, flexible and powerful client-server configurations will dominate the local processing configurations in the nearest future.

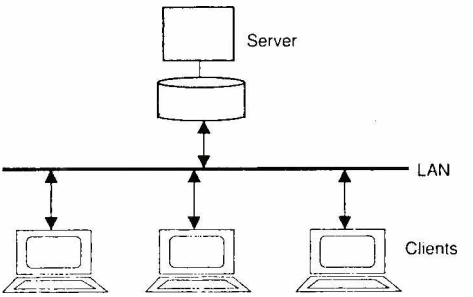

Figure 8. *The client-server configuration*

A workstation can gain access to server resources through Internet. Figure 9 illustrates such a configuration. A workstation accesses data located at a Web site through a Web browser. Vendors of DBMSs provide ready-to-use Web servers (or tools for development of specialised Web servers) that enhance the functionality of their database systems so that they can perform services through Internet. The Web server plays the role of a broker between workstation applications (in this case – – Web browsers) and database servers. It submits data requests to an appropriate server, receives results from the server, formats them into HTML pages and transmits them to the requesting browser. A Web server can reside on the same computer as the database server, or it can be installed as a separate Web site to communicate with many different servers in many remote database systems.

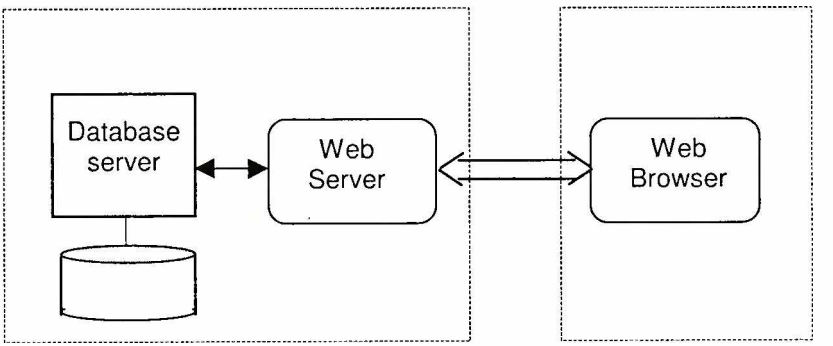

Figure 9. *The client-server configuration on the Web*

Another configuration of an autonomous local database node is a *peer-to-peer* configuration (Figure 10). There is no separate server in this configuration. Each computer can play the role of a server for other computers and the role of a workstation. So, server (back-end) processes and application (front-end) processes can be simultaneously run on one computer. Information stored at one computer is accessible to other clients, provided the clients have appropriate permissions. Such configuration can form a "locally distributed", simple database system, though without typical fea-

tures of distributed database systems such as distributed transactions management or support for data replication. An advantage of a peer-to-peer configuration is its flexibility and simplicity in sharing resources among all computers in the network. The main disadvantage is that it is difficult to provide sufficient data security (because of the lack of sophisticated authorisation and recovery mechanisms) and to achieve global optimisation of the system. As a matter of fact, a peer-to-peer configuration is not a proper environment for fully functional database systems, but rather for locally distributed information systems, suitable for small departments and not for very demanding applications.

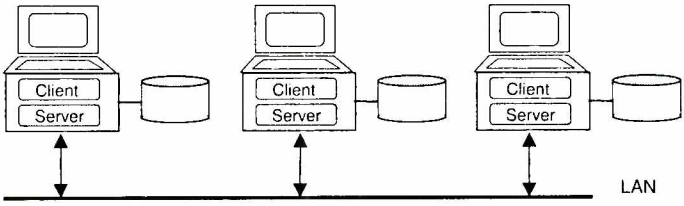

Figure 10. *The peer-to-peer configuration*

5. The future

Basic physical kinds of architecture employed in contemporary database systems have been sketched here. It must be stressed, however, that nowadays the boundaries between different configurations become blurred, mainly due to the rapid technology progress. Also mechanisms of information management become more "global" in the sense that there is a strong tendency to make increasingly large information volumes available to the increasing number of users through a world-wide, easy-to-access communication medium. An example of such global configuration, proposed independently by different vendors (Sun, Oracle), is the *Network Computing Architecture* (Figure 11). The skeleton of the architecture is a global bus with a large number of "sockets" for functional modules to be plugged-in.

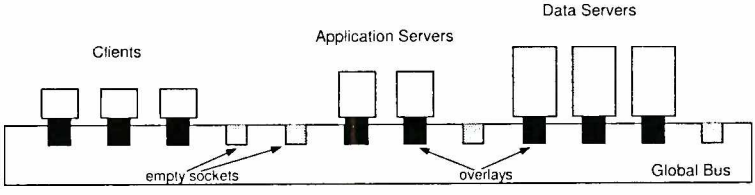

Figure 11. *The Network Computing Architecture (NCA)*

The functional modules can be:

- Clients — simple *network computers* equipped with huge main memory and multifunctional network browsers.

- Application servers — computers that store and make available applications to be run on clients.
- Data servers — computers that store data and make it available; they can be simple file servers, or sophisticated database servers running DBMSs.

The modules are plugged into the sockets through *overlays* that are able to filter out some functionality of modules. For instance, the same data server can have different functionality if it is plugged into different sockets. The overlays can also be responsible for checking user permissions and performing other supplementary tasks. It is assumed that the applications offered by application servers can be executed on any client machine without any modification (they are perfectly portable throughout the whole global bus). The emerging technology that could be applied in the NCA is Java introduced by Sun Microsystems. Applications written in Java (called *applets*) can be run on any client equipped with a browser that can interpret Java. An applet can access data stored in data servers through *Java Database Connectivity* (JDBC) interface, which is a layer over the standard ODBC protocol.

The NCA is a good example of a general tendency in data processing and in database technology in particular — the distribution of processing and data, and a universal connectivity between different systems. It may be expected that regardless of specific configurations, in the nearest future the trends towards distribution of database systems in a macro (corporate and world-wide) scale and distribution and parallelism in a micro (departmental or local) scale will dominate and will be further developed.

References:

[1] Date C. J., *An Introduction to Database Systems, Vol. I, 5th Ed.* Addison-Wesley Pub. Co, 1990

[2] DeWitt, Gray J., *Parallel Database Systems: The Future of High Performance Database Systems*, Communications of the ACM, Vol. 35, No 6, pp. 85-98, 1992

[3] Olsen, *Parallel Systems Management with Oracle 7 and IBM S/390 Parallel Transaction Server*, Oracle Magazine, Fall 1994, pp. 97-100, 1994

[4] *Parallel Database Systems*, Proc. of PRISMA Workshop, Noordwijk, The Netherlands, Sept. 24-26, 1990, Springer-Verlag, 1990

[5] Bell D., Grimson J., *Distributed Database Systems*, Addison-Wesley Pub.Co., 1992

[6] Khoshafian S., *Object–Oriented Databases*, J.Wiley & Sons Inc., 1993

[7] Bobrowski S., *Implementing Data Replication. Part I & II*, Oracle Magazine, May/June (pp. 93-96), July/August (pp. 97-102), 1996

[8] *Informix–Online Workgroup Server* (Manual), Informix Software Inc., 1996

[9] Hurwitz J., *Second–Generation Client/Server Computing*. Hurwitz Consulting Group, Inc., 1994