DISTRIBUTED DATABASES IN A HETEROGENEOUS COMPUTING ENVIRONMENT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/110

7 September 1989

cover page

DISTRIBUTED DATABASES IN A HETEROGENEOUS COMPUTING ENVIRONMENT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/110

7 September 1989

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DISTRIBUTED DATABASES IN A HETEROGENEOUS COMPUTING ENVIRONMENT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M89/110

7 September 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

To appear: AUTOFACT '89

Distributed Databases in a Heterogeneous Computing Environment[†]

Lawrence A. Rowe

Computer Science Division-EECS
University of California
Berkeley, CA 94720

Abstract

This paper describes alternative software architectures for database applications. The ANSI SQL standard data model and the database vendors commitment to tools and systems that support portable applications will allow organizations to change hardware platforms and reconfigure their computing environment more easily. The features of a full-function distributed database system and how they can be used to solve some common problems in a large manufacturing organization are also described. Lastly, database gateways are described that interface existing databases stored in older database systems (e.g., hierarchical and network) to applications that use the relational model.

1. Introduction

Engineering design and manufacturing environments are heterogeneous computing environments. A typical environment will have many different types of computers (e.g., embedded computers, personal computers, workstation computers, minicomputers, and mainframe computers) that are interconnected by a network. Many different applications including engineering design, real-time factory control, and corporate business applications are run on this collection of computers.

These applications manage databases that contain relevant data. For example, engineering design applications maintain documents that specify a product's requirements and design, geometric models that describe a product, and component lists that specify the parts that makeup a product. A factory control system keeps track of the work-in-progress (WIP), status of equipment in the factory, and test data collected during the manufacturing process. Finally, corporate business applications manage data about orders, personnel, and inventory. These information systems (i.e., the application programs, the databases, and the operational procedures) are an important corporate assest.

Computing environments of the past were dominated by a few large mainframes. The proliferation of low cost microprocessors has lead to the introduction of hundreds, and in some cases thousands, of different types of computers into the engineering design and manufacturing environment. Application programs must run on these different computers. At the same time, they must access a shared, integrated database created by merging

† This research was supported by National Science Foundation Grant MIP-8715557.

the autonomous application-specific databases. A shared, integrated database is required for two reasons. First, applications will have to access data in more than one database to meet future business requirements. And second, existing applications that currently run on a single large computer will be distributed to different computers in the environment.

An important trend in the development of application software is to build portable, reconfigurable applications. A portable application can be run on different hardware platforms and different software systems (e.g., operating systems and database systems). A reconfigurable application can be distributed across different computers in different ways. Vendor independence is important because it allows an organization to change computers without having to rewrite the application software. Consequently, the organization can migrate to more cost effective computer systems with less difficulty. Application reconfiguration is important because the distribution of the software across different computers may change due to changes in the price performance of hardware.

This paper surveys the current state of database systems and database application architectures. Alternative software architectures and database system support for those architectures will be presented. We will attempt to show that a standard data model and a vendor commitment to portable applications allows an organization to build portable, reconfigurable applications.

The software environment we envision is applications written in a 4th Generation Language (4GL) or a 3rd Generation Language (3GL) that access a relational database using the SQL query language [Dat81]. These applications may interact with the user through an alphanumeric or graphic terminal and they may run on different operating systems.[1]

The remainder of the paper is organized as follows. Section 2 discusses alternative software architectures for applications and the DBMS. Section 3 presents the features that a full-function distributed database system should provide. And, section 4 describes database system functionality that is required by engineering design and manufacturing applications.

2. Software Architectures

This section describes software architectures for applications and database management systems (DBMS). These architectures can be used in different ways in an engineering design and manufacturing environment. Some examples are discussed at the end of this section.

The application program must run in a separate process from the DBMS as shown in figure 1. The application process contains code specific to the application and code to interact with the user through a terminal or workstation interface. The DBMS process

[1] The goal of portable applications will be easier to achieve if a UNIX operating system is used. However, many 4GL's and 3GL's are designed to run on different operating systems.

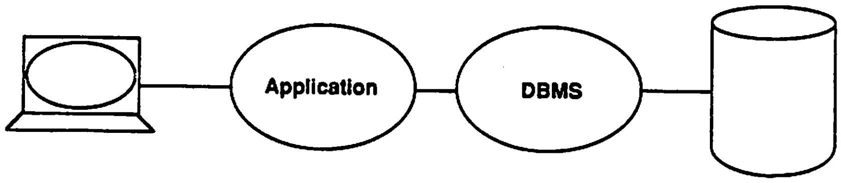

Figure 1: Separate application and DBMS process architecture.

contains the database system code. The DBMS is run in a separate process so that it can run in a different protection domain. That way, the application program can access only the data that it is supposed to access.

Another reason the DBMS is run in a separate process is so that it can be run on a different computer. This configuration is sometimes called *remote database access*. The advantage of this configuration is that the workload can be spread across two computers rather than just one computer. If the computer that runs the DBMS process is dedicated to that function, it is called a *database server*. This configuration is a natural one when users are running applications on a workstation but still need to share a common database. The applications access the same database managed by the DBMS process on the database server. This configuration has the added benefit of putting the DBMS code close to the disk that contains the data and the interface code close to the user. Because the application issues high-level relational queries to the database server, only the data required by the application must be sent to the workstation. In contrast, if the DBMS process ran on the workstation and accessed the data through a network file system, all disk pages that the DBMS read would have to be sent to the workstation. A similar argument can be made that the application process should be located close to the user to minimize the messages that must be sent between the interface devices and application program.

In a multiuser environment, each application might have a separate DBMS process as shown in figure 2. The DBMS processes run in "shared instruction" mode so that only one copy of the DBMS code must be kept in main memory. However, this architecture will be less efficient than the server architecture shown in figure 3. The server DBMS process handles requests from more than one application.

The server DBMS is a mini operating system that manages and schedules "tasks" (i.e., the application queries), issues I/O requests to read and write disk pages, and manages a buffer pool of recently accessed pages.

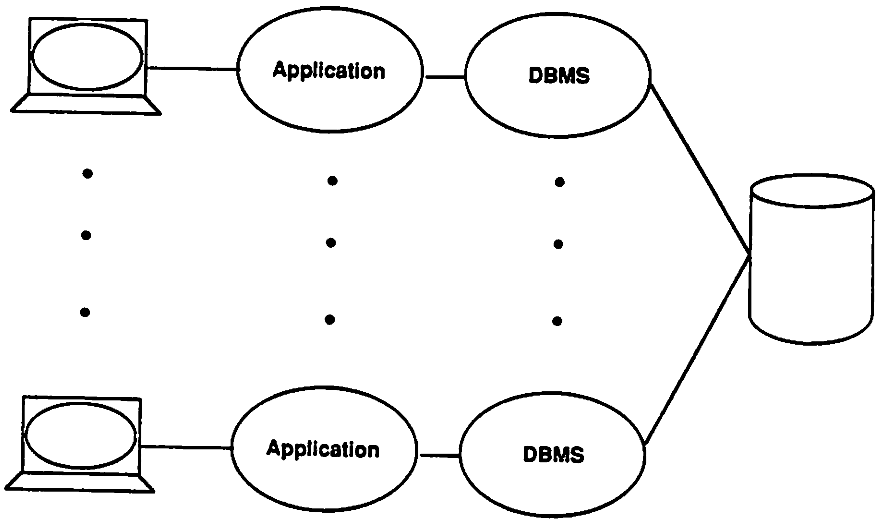

Figure 2: Process-per-user configuration.

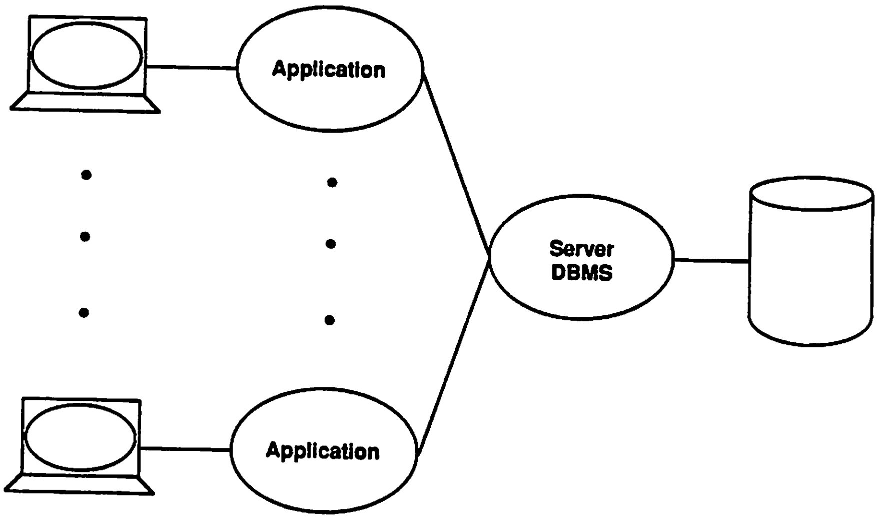

Figure 3: Server DBMS architecture.

The primary advantage of a server DBMS over a process-per-user DBMS is that the server can be customized to the database tasks that it is executing. A customize operating system is more efficient than a general-purpose operating system because it knows more about the tasks that it is running [Sto81]. For example, the server has a good idea of how much CPU time and how many I/O's each task will perform because the query optimizer estimates these numbers when it selects a query execution plan for the task. Most on-line transaction processing systems use a server architecture because it is the most cost effective architecture.

A server architecture is a good architecture, but it does not take advantage of the tightly-coupled parallel processors that are currently popular. The appropriate configuration in that environment is a multi-server architecture. Figure 4 shows a multi-server architecture. Each server runs in parallel on a different processor. The advantage of this architecture is that both response time and thruput can be improved because the system can execute many tasks in parallel. More work will be completed by the multi-server architecture that exploits parallelism than by a server architecture that uses interleaved execution. Another advantage of the multi-server architecture is that DBMS cycles can be increased by adding another processor if the application load grows to where it cannot be

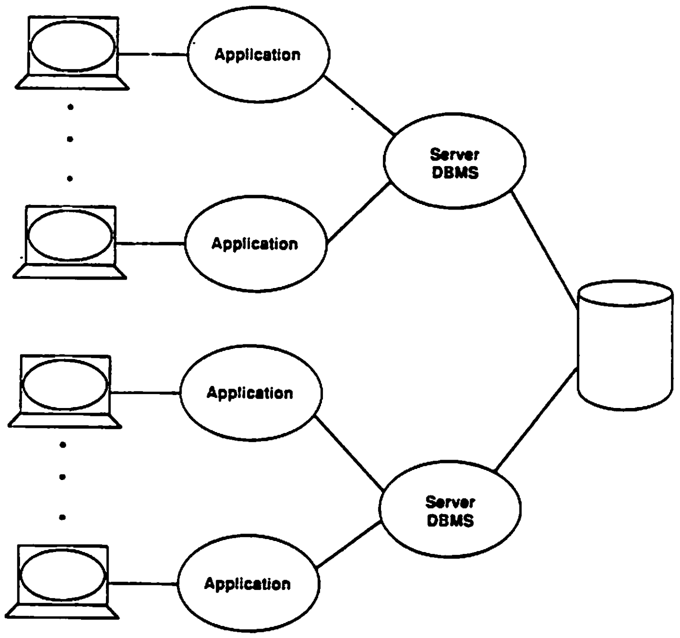

Figure 4: Multi-server architecture.

handled by an existing computer system.

Parallel processors are most frequently used to support a larger application workload. However, some vendors are developing query executers that will use several processors in parallel to speed up query execution. Parallel execution is also possible in a distributed database system as discussed below.

Thus far, we have focused on applying the idea of servers to the DBMS program. The same idea can be applied to the application program. Figure 5 shows an application server process that is connected to several database servers. The application server can send a request to the server process that is least busy in order to balance the load across the different processors. A disadvantage of this architecture is that the application server can become a bottleneck which will limit the system performance. Consequently, it is necessary to support multiple application servers. Although some systems use this architecture, it conflicts with the trend of moving applications to a user's workstation.

Some organizations cannot justify the cost to put a high-priced graphic interface workstation on every person's work space. Some window systems, notably the X Window System [ScG86] and Sun's NeWS System [SUN89], actually run a server process that per-

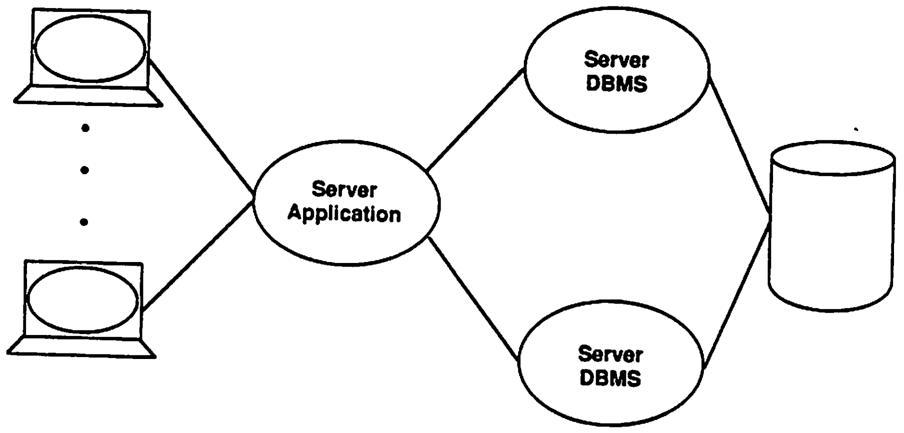

Figure 5: Application server architecture.

forms screen output and handles keyboard and mouse input.[2] The application process architecture in the workstation is shown in figure 6. Each application interacts with multiple windows that are managed by the window system server. A new generation of intelligent terminals that execute just the X Window System server are being developed that will cost roughly $1,000 and run exactly the same applications with the same interface "look and feel" as a more expensive workstation. The advantage these devices offer is that a low cost-per-user graphic terminal can be used that will run exactly the same applications as a high cost-per-user workstation. The X terminal solution may run slower than the workstation for some applications because the application runs in a shared compute server rather than in the local workstation.

The last database system architecture that will be discussed evolved from the desire to access data stored on physically separate computers. This architecture, called a *distributed database system*, is shown in figure 7 [CeP84]. The FE application sends exactly the same commands to the distributed DBMS that it sends to a single-site DBMS. The distributed DBMS sends commands to the local DBMS's to implement the application query. Typically, the application process and the distributed DBMS process run on the same computer and the local DBMS's run on the computers that contain the data. This architecture has three advantages. First, data can be accessed transparently. That is, an

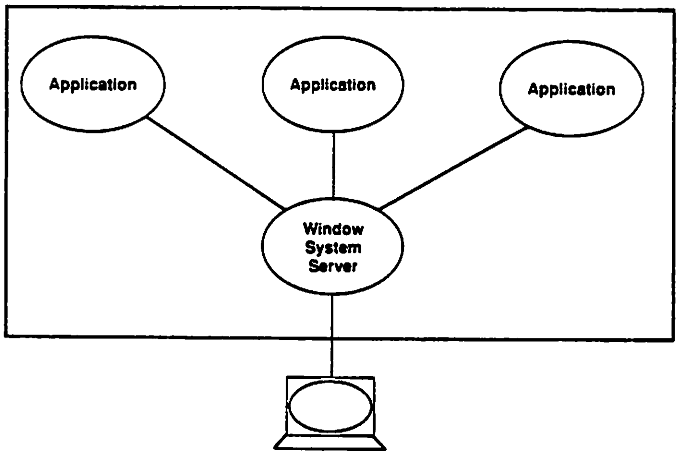

Figure 6: Workstation window server architecture.

[2] A protocol could be defined to run the other major window systems (Microsoft's Presentation Manager and the Macintosh Toolbox) as servers, but they do not inherently support a window server.

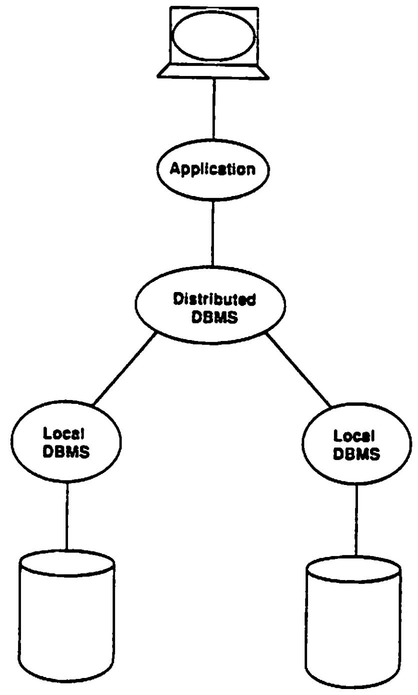

Figure 7: Distributed database system architecture.

application program can access data stored on different computers without having to know where it is stored. Second, queries can be executed in parallel to improve performance. And third, computers can be incrementally added to or removed from the system without requiring changes in the application programs. The next section describes in more detail the features of a distributed database system.

Figure 8 shows a typical hardware configuration in a factory. A local area network connects together a factory computer, a database server, and a collection of workstation and cell computers. In addition, terminals are connected to the factory computer and to some cell computers. Notice that the network has gateways to the corporate and engineering design computers too. The data for the WIP system that controls the manufacturing process will be distributed to the database server and the cell computers. Most of the time the applications that run locally on the cell computers will access data on the local database and occasionally they will access data on the database server. A server DBMS should be run on the cell computer that can handle the local applications and the commands sent to it by a distributed database process that runs on the factory computer or one of the workstations. An engineer trouble shooting a problem in the factory might

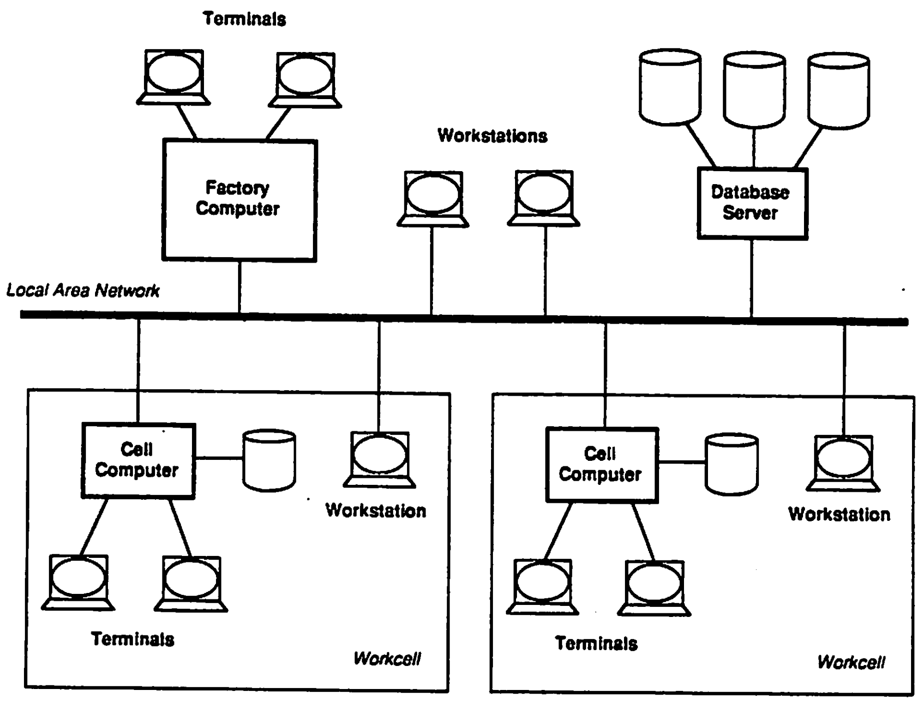

Figure 8: Typical computing environment at a factory.

query data in the database server or a local database on a cell computer. However, all these applications should be written in an SQL compatible language so that they can be easily ported to run on the different databases.

This section described alternative architectures for database and application systems. A typical factory computing environment was described and ways to use the alternative database architectures in that environment presented. Most factories have heterogeneous hardware so it is crucial that the application software run on this different hardware without changes.

3. Distributed Database Systems

Distributed DBMS's have only recently been introduced to the commercial marketplace. Consequently, most products are primitive. This section describes the features that a full-function distributed DBMS should eventually support. Current products will be enhanced to provide most, if not all, of these functions over the next 5 years.

First and foremost, a distributed DBMS must be functionally equivalent to a single-site DBMS. It must support ad hoc and program query access to the database, transactions (i.e., multiple user access and crash recovery), integrity and protection, and other single-site DBMS services. These functions must transparently operate on data located at different sites.

A distributed DBMS can provide more function than a single-site DBMS because it is distributed. The natural partitioning of a database is to place different tables at different sites. Queries that involve one table can be executed at one site. Queries that involve tables at more than one site can either move all the data to one site and execute the query there or move subsets of the data to several sites and execute the query at those sites. The distributed query optimizer is responsible for picking an efficient query execution plan given a query, a particular data distribution, and information about the cost to move data and execute local queries.

A table can be distributed to more than one site by either vertical or horizontal partitioning. A vertical partition stores different columns of the table at different sites. For example, given the employee table

EMP(Name, Address, Dept, Picture)

The *Name*, *Address*, and *Dept* columns could be stored at one site and the *Picture* column, presumably a large pixmap, could be stored at a different site. A horizontal partition stores rows of the table at different sites. Each partition is called a *fragment*. The *EMP* table above could be partitioned based on the employee's department. For example, administrative employees could be stored at headquarters on the corporate computer and manufacturing employees could be stored in the factory computer in the manufacturing plant. Regardless of how the data is partitioned, the following query should return the same results

```
select *
from EMP
where Name = 'John Smith'
```

The data partition is chosen to optimize a particular set of queries. For example, if employee pictures are infrequently accessed they can be stored on a slow optical disk in the corporate data center. This example uses vertical partitioning to store infrequently used data on slower devices. On the other hand, suppose there is a very large table, say 10 gigabytes, of historical sales data that a marketing person is analyzing for trends. This table can be horizontally partitioned and distributed to different computers so that the ad hoc queries run by the marketing person can be executed in parallel to improve response time.[3]

A distributed database can be accessed from geographically dispersed places. For example, corporate headquarters might be in Detroit but a manufacturing plant might be in

[3] A simple query to compute a mean that takes 48 hours to run on a single processor can be run in 60 minutes on a parallel processor.

Mexico. Trouble shooters in the manufacturing plant might need to access a part database stored at headquarters. If many queries against the part database are being run, the data might be moved many times to Mexico. It may be impractical to move the part database to Mexico permanently because it is also being accessed by other plants in the US. The solution might be to keep a copy of the data in Detroit and Mexico. The distributed DBMS will manage the copy. In other words, the distributed DBMS will choose the least expensive copy to access and it will propagate updates to all copies to maintain database consistency.

Copies are a good idea if you have data that does not change frequently. Otherwise, the cost of maintaining the copies may out weight the advantage of keeping it. Another possibility is to maintain a snapshot of the part table in Mexico. A snapshot is just a copy of the table at a particular time. Then, the snapshot can be updated with changes at regular intervals (e.g., once a day, week, or month). A snapshot improves the execution of queries in Mexico, but the answers may be out of date. This solution might be perfectly acceptable if the parts table does not change very often. The distributed DBMS should manage the periodic updating of the snapshots.

Another feature of a full-function distributed DBMS is replicated catalogs. A catalog is a table maintained by the DBMS to keep track of the database itself. For example, a catalog exists that describes each table or fragment in the database and the site at which it is stored. A single catalog leaves the distributed DBMS vulnerable to a single failure. The entire database will be unavailable if the site that holds the catalog is down. The solution is to replicate the catalogs at several sites. That way, if one site goes down, the distributed DBMS can access a copy of the catalog at a different site. Replicated catalogs are not free. Schema changes (e.g., adding or removing indices, adding columns to a table, or adding or removing tables) will require that all copies of the catalogs be updated. Consequently, replicated catalogs will be useful for stable production environments that need high availability.

The last feature of a distributed DBMS is a distributed transaction log. Recall that log records have to be written before a transaction can commit so that the database can be restored to a consistent state should the system crash before the updated data pages are written to disk. A limited function distributed DBMS will have a log at one site. This solution is acceptable if the overhead to write the log is small as will likely be the case if the log is at the same geographical site. However, if the log is at a remote site, it might take a long time to write the log. The solution to this problem is to have many logs distributed at different sites. Distributed logs do not cause a problem for the applications that run on a distributed DBMS, but they do introduce considerable code complexity into the distributed DBMS itself.

4. Heterogeneous Distributed Database Systems

Distributed database systems solve the problem of building integrated information systems that use geographically disperse databases and parallel processing. They do not, however, solve the problem of managing the transition from existing applications and databases to these new systems nor the problem of integrating disparate data that is not stored in a database (e.g., geometric models stored in a file). A distributed database system can be used to solve these problems if we build *gateways* from the distributed system to the older systems.

Figure 9 shows a heterogeneous distributed database system that interfaces to an IBM IMS database and a DEC RMS file. The IMS and RMS gateways translate the commands sent by the distributed DBMS process to a local database system into commands on the foreign database or file system. The results of executing these commands are translated into responses to the DBMS process which passes them back to the application program.

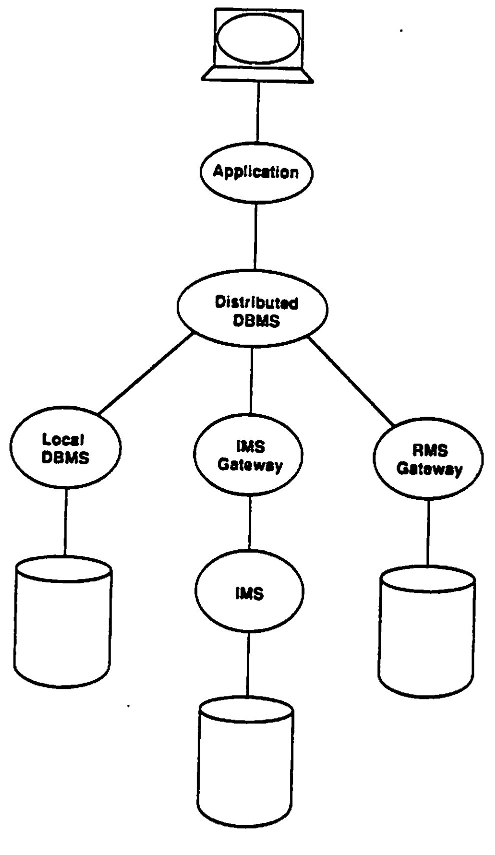

Figure 9: Heterogeneous distributed database system.

The majority of relational queries can be translated to a foreign database. However, some queries, usually update commands, cannot be translated. For example, a relational database with *EMP* and *DEPT* tables that represent employees and departments can be updated so that employees exist who are not in any department.[4] This update cannot be mapped to a CODASYL system that represents the departments by *DEPT* records and places the employees in owner-coupled sets.

Other problems must be solved to make these systems practical. For example, a common dialect of SQL must adopted so that gateways between different relational databases can be implemented. Common data types, catalogs, and error messages must also be specified so the systems can work together smoothly. Nevertheless, heterogeneous distributed databases can be built and they will solve an important problem.

5. Summary

Single site relational database systems are approaching maturity. Functions can still be added to the leading products, but most products already have a rich set of functions. Distributed database systems are an important new technology that will allow data stored at different sites to be accessed and used as though it were at a single site. Distributed DBMS's are immature products. However, they offer real promise for several problems. First, they will allow applications written against a relational database to access data stored in different physical databases. The data can be stored at different sites and in different DBMS's (e.g., hierarchical, network, or relational).

Second, they will take advantage of the proliferation of low cost computers. Parallel processing can be exploited to handle changing workloads and to improve response time for selected queries.

These capabilities will be available without changes to application programs because the a standard program interface to a database is used and because database vendors recognize that customers want portable applications that can access heterogeneous databases.

It may take 5 years for the vendors to complete the network protocols, database gateways, and distributed database systems. Nevertheless, the direction for the future is portable, reconfigurable applications that run on heterogeneous computing environments.

References

[CeP84] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, NY, 1984.

[Dat81] C. J. Date, *An Introduction to Database Systems*, Addison-Wesley, Reading, MA, 1981.

[4] This example assumes that the database does not have a referential integrity constraint that disallows this situation.

[ScG86] R. W. Scheifler and J. Gettys, ''The X Window System'', *ACM Trans. on Graphics 5*, 2 (Apr. 1986).

[Sto81] M. Stonebraker, ''Operating System Support for Database Management'', *Comm. of the ACM*, JULY 1981.

[SUN89] *NeWS Programmer Guide*, Sun Microsystems, Inc., Mar. 1989.