I. Introduction

A. Most large databases require support for accesing the database by multiple users, often at multiple physical locations (sites). There are a variety of overall system architectures that can be used to accomplish this.

B. Historically, early database systems were based on a CENTRALIZED MODEL, in which the database resides on a single computer system that allows access by remote terminals. This is still the model used by many systems today.

1. Basic characteristics:

a. All data resides on a single computer system.

b. All computations using the data are performed by this system.

c. Remote access is provided by "dumb terminals" or PC's running terminal emulation software.

Example: Gordon's old administrative computer system used this model since it was first installed in 1979. The administrative database resided on a single computer system (originally a PDP-11/70, then a VAX-11/780, then a cluster of three Alphas). Campus offices accessed this database via DEC VTxxx terminals or PC's running terminal emulation software, or later via web-based applications accessing the database through a web server running on the same system.

2. The chief advantage of this approach is simplicity - for installing and maintaing software, performing backups, and managing issues such as concurrency and recovery.

3. However, the centralized model has a number of disadvantages:

a. Typically centralized storage and manipulation of data goes hand-in-hand with centralized CONTROL of data - users have limited autonomy.

b. "Dumb terminals" support only very unsophisticater user interfaces - basically just ASCII text and text based menus.

c. A centralized system is totally vulnerable to failure of the centralized site. Thus, for example, a power failure at the site where the computer is located can shut down all access to the database, even at remote sites unaffected by the failure.

d. As the volume of transactions handled by the system grows, it can become increasingly difficult for a single system to handle the demand.

C. These problems - especially the latter - have motivated the development of alternative system architectures which move away from reliance on a single computer to perform all processing.

1. The Client-Server Model

2. Use of Parallel Processing

3. Distributed Databases

II. The Client-Server Model

A. The centralized model originated prior to the development of the microprocessor, which made PC's possible. With the replacement of dumb terminals by powerful and inexpensive PC's, it becomes possible to shift some of the burden of computation from the computer system storing the database to the individual computers serving individual users. This led to the development of the client-server model, which we have discussed earlier in the course.

B. A typical client-server application can be thought of as a system comprised of several layers.

1. The structure discussed in the text: A "front end" that manages interaction with the user, and a "back end" that manages the database.

2. A three layer architecture:

a. The user interface layer (typically a GUI).

b. The business layer (performs the actual processing specific to the application).

c. The database layer.

C. Actually, there are a couple of ways to structure the database layer that we didn't discuss when we discussed client-server architecture earlier in the course, because we had not yet discussed the overall structure of a DBMS.

1. One model is called the TRANSACTION SERVER model - the database server executes database transactions on behalf of the client, but need not incorporate any awareness of what purpose the transactions are actually serving as far as the user's application is concerned.

a. As noted in the text, when SQL is used as the medium of communication between the client and the server, it becomes very possible for the client and server software to be produced by two different vendors - and, indeed, for one server to service applications written using many different software packages, and, as well, for one client to access different servers running different DBMS's.

b. This shifts the load of application-specific computation from the server to the client, while still leaving the server responsible for all query-processing related computation (query parsing, strategy selection, performing joins, etc.) In particular, no application-specific code need reside on the server (unless stored procedures are used, as might be done to support embedded SQL), and no database-specific code needs to reside on the client.

Example: this is the model used by Gordon's new administrative software. All applications use a common database server running on a single system. Some are PC-based applications that access the database directly via ODBC; others access the database through one of several application servers supporting different adminstrative applications.

2. It is also possible to use a DATA SERVER model, in which the server delivers physical database pages to the client, which then performs computation using them. This shifts some of the database layer load from the server to the client - but, of course, requires that the client software know more about the structure of the database, and incorporate some of the software typically considered part of the DBMS.

D. In none of the client-server variants we have considered do we do away with a central system containing the database - we simply reduce the amount of computation for which it is responsible. Further, all disk accesses needed are performed on the server's disk(s) that hold the database.

E. When the phrase "CLIENT-SERVER" is used without further qualification, it typically refers to to the Transaction Server model, with SQL providing the interface between the two layers, often through the use of ODBC or JDBC.

F. In addition to removing some of the computational load from the central system, the client-server model has a number of other advantages:

1. The possibility of much more sophisticated user interfaces.

2. The possibility of integrating database access with other, non-database applications on the client - e.g. doing analysis of data obtained from a database using a spreadsheet, or incorporating it into a document using a word processor.

3. The ability to develop needed applications quickly, without having to rely on the programming staff associated with the central database, and using a variety of development environments, possibly from a vendor other than the supplier of the database.

III. Use of Parallel Processing

A. The client-server model was motivated in part by a desire to shift some of the processing load from the central server to the local client systems, thus reducing the requirements placed on the central system. Ironically, it may have had the opposite effect - the client-server model expedites the development of more database applications, and may actually increase the burden on an organization's server system(s)!

B. In order to keep up with the demand for database accesses, server systems must often be able to handle a growing volume of database transactions - growing because or organizational growth and/or an increased number of applications using the database. One way to address this is with faster and faster hardware, or course - but at any given point in time there are technological limits as to how fast a single system can be. Two components of the server, in particular, can become performance bottlenecks:

1. The server's CPU.

2. The server's disk(s).

C. An alternative to acquiring ever faster hardware is to make use of PARALLELISM - with a server containing two or more CPU's that share the workload between them, possibly with multiple disks that can also be accessed in parallel. This is discussed at some length in chapter 21 of the book - one we will not have time to cover. (But you can certainly read it on your own if you wish!)

   1. Parallelism may be used in server systems under the client-server model.

   2. Parallelism may also be used in the centralized model - the central system becomes a cluster of CPU's made to look to the user as if they were a single system.

D. One important observation about parallelism is to recognize that there are a variety of reasons for using a parallel system. A given system's success must be measured against the goals that led to its installation.

   1. One possible goal is SPEEDUP - to make the processing of individual transactions (of the same size) faster. This would, of course, require the use of two or more CPU's to cooperate in the performing of a single transaction.

   2. Another possible goal is SCALEUP - to make it possible to handle a greater volume of work in the same amount of time. This, in turn has two sub-categories:

      a. BATCH SCALEUP involves increasing the SIZE of individual transactions, as would occur if the size of a database grew, so that operations such as select and join require scanning more tuples. This, again, entails having two or more CPU's cooperate in the performing of a single transaction.

      b. TRANSACTION SCALEUP involves either supporting larger transactions (as might happen if the size of the database grows and searches and joins become more cumbersome) or increasing the VOLUME of transactions, as would occur if the number of users accessing the database were to grow. This can be achieved by still having each transaction handled by a single CPU, but by using multiple CPU's to increase the number of transacactions that can be processed during a given period of time.

   3. For a variety of reasons (discussed in the text), efficiently dividing the work of a single transaction among two or more CPU's is relatively difficult. Thus, the easiest kind of performance improvement to attain is transaction scaleup - which, fortunately, is the kind of scaleup most often needed. However, there are also applications which require batch scaleup - e.g. decision-support systems that require analyzing large quantities of transactional data. Speedup is usually less of an issue.

E. The book discusses various architectural issues and alternatives in parallel database systems. In brief, such systems can be:

   1. Shared memory - multiple CPU's sharing a common memory and disk system, with each CPU having its own cache and or private local memory as well. (Absent the latter, memory contention would severly limit performance with even a few CPU's) Inter-CPU communication is very fast in this model, since it is done be read/writing data in shared memory.

   2. Shared disk - each CPU has its own memory, but all CPU's share a common disk system. Inter-CPU communication is done by message passing over a network - a slower operation than shared memory; but contention for access to shared memory is no longer a perormance-limiting issue. Since the disk system containing the database is shared, all CPU's have high-speed access to all parts of the database. This kind of system is sometimes called a CLUSTER.

      Example: Gordon's old administrative system eventually used a cluster comprising three CPU's (WISDOM, MERCY, and GRACE) accessing the common database. Individual users logged in to a particular member of the cluster (based on the department they worked for), but all had access to the database. The cluster as a whole was made to appear to the outside world as a single system known as HOPE. The basic model of computation was still a centralized one.

   3. Shared nothing - each CPU has its own memory and disk(s). The database may be partitioned and/or replicated between the various subsystems - so a given CPU may have fast access to parts of the database, while needing to access other parts "owned" by another CPU over the network that interconnects them.

   4. A hierarchical combination of the above.

F. Because of interaction/interference between the CPU's accessing the same database, there are practical limits as to how many CPU's can be used before the gain created by increased computational power is offset by the losses due to the systems "getting in each other's way".

G. We must also consider ways to parallelize access to the database itself on disk. (If the entire database resides on a single disk, then disk accesses are necessarily sequential, which severly limits parallelism between transactions.)

   1. Parallelizing of the disk accesses is inherent in the shared nothing model.

   2. It can also be achieved in the other two models by the use of RAID.

IV. Distributed Databases

A. If we take the idea of parallelism further, we move in the direction of a DISTRIBUTED SYSTEM.

   1. In the parallelism model we have discussed thus far, the overall system still resembles a centralized system in the sense that the database and the CPU's accessing it still reside at a single physical site.

   2. In a distributed system, the database is spread over a number of physical sites, each of which houses a portion of the database. (Often, this distribution mirrors the organizational structure of the database's owner.)

B. Distributed systems are characterized by a much looser coupling between systems, which facilitates increased gains through parallelism.

C. We discuss this as the next major topic.