CS352 Lecture - Database System Architectures

Last revised April 6, 2023

Objectives:

1. To discuss possible architectures for a DBMS - centralized, distributed
2. To introduce DBMS parallelism
3. To introduce speedup and scaleup (batch and transaction)
4. To introduce the CAP theorem
5. To introduce cloud computing and Ias, Pas, and Sas models

Materials

1. Projectable of two variants of client server architecture
2. Projectable of Figure 20.9

I. Introduction

A. Today's topic overviews several more advanced topics.

B. The textbook deals with these topics in chapters 20-24.

C. Obviously, the coverage here will be restricted to a very broad overview. More detailed coverage would often be found in a second DBMS course.

II. Centralized DBMS's

A. Most large databases require support for accessing 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 local or remote users, with all of the computation being done on the central computer. This is still the model used by some systems today.

1. Basic characteristics:

    a. All data resides on a single computer system, all sharing a common memory system and set of disks - perhaps using a RAID configuration.

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

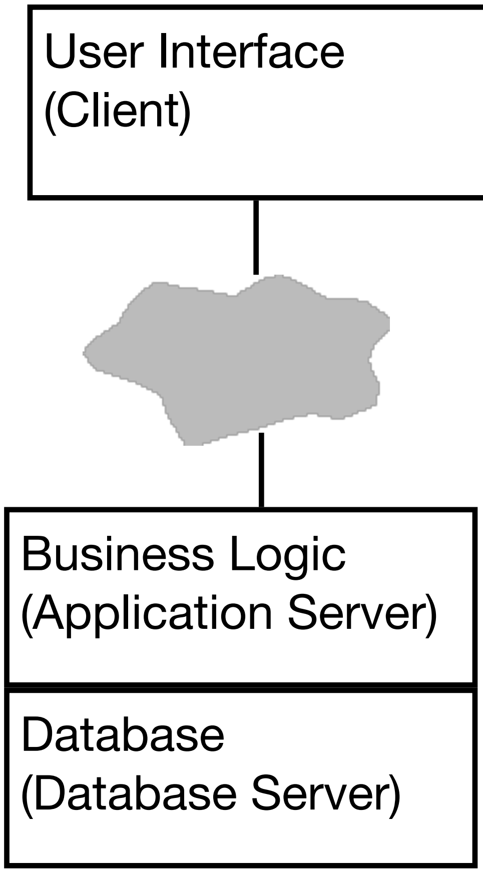
    c. The computer system may be a single user system or a multiuser system or even an embedded system such as SQLite, which is a C library that might be embedded into an application program Today, it is often a multicore computer or a virtual machine.

    d. Historically, multiuser systems afforded access to multiple users via text-only terminals. Today, it is common to offer access to multiple users via personal computers or workstations running terminal emulation software, or via a web interface.

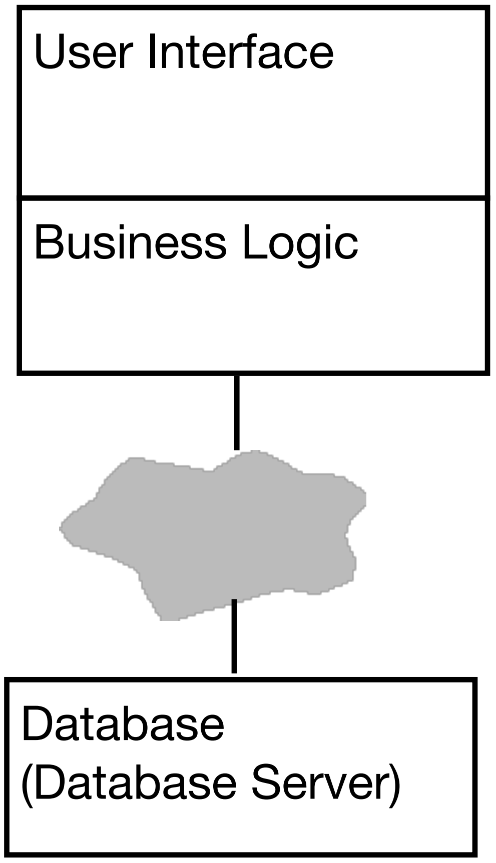
       Example: Gordon's first administrative computer system used this model when it was first installed in 1979. The administrative database resided on a single computer system. Campus offices accessed this database via DEC VTxxx terminals or later PC's running terminal emulation software, or even later via web-based applications accessing the database through a web server running on the same system. Though Gordon uses a different system today, it still provides access to multiple users in a similar way.

2. Often, a centralized database system is used as part of a client-server model that is structured something like one of the following alternatives.

   PROJECT


User Interface (Client)
Business Logic (Application Server)
Database (Database Server)

Thin Client Model



Thick Client Model

a. In either case, each layer has specific responsibilities:

   i. The user interface layer is responsible for interaction with the user, typically via a GUI (or some sort of command line). Frequently, this layer is implemented by a GUI running a set of application-specific web pages which may do some computation to validate input using javascript or something like it.

   ii. The business logic layer is responsible for the tasks that are specific to the application. In the thin client variant, this layer may use a generic web server to communicate with the client, but the bulk of the computation is done by code specific to the application - written either in a programming language or in scripts executed by the web server running on the server side of the network.

    iii. The database layer stores the data, but incorporates little or no application-specific knowledge beyond things like integrity constraints.

        (a) Triggers.

        (b) Stored procedures to be called by the business logic

b. In the thin client variant, the business logic layer and the database layer may reside on the same physical computer system, or two different systems connected by a network.

c. As noted in the text, when SQL is used as the medium of communication between the business layer and the database, it becomes very possible for the business 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 accessing different DBMS's.

d. Some examples:

    i. This is the model used by my.gordon and Gordon 360. The former uses software purchased from Jenzabar, and the latter uses code largely written by students in the Summer Practicum. But both access the college's Microsoft SQL Server database using SQL. In this case a thin client model is used - the program is accessed via a web browser accessing an application server and database server, The servers run in separate systems (actually virtual machines).

    ii. The programming project in this course uses a variant of the thick client model. The GUI uses code written by me and supplied in compiled form in `project.jar`. The business logic is performed by student-wrriten code in `Database.sqlj`. The database layer is IBM Db2.

In previous years, this client was a workstation in 244 and the database resided on joshua.cs.gordon.edu, communicating over the campus network.

This pattern is still the case if we consider the container running as a virtual machine under Docker to be a separate "computer" (as in fact it is since it has its own OS). Now the two "computers" communicate over an internal network running on the host (which is what "-p 50000:50000" meant in the command you used when installing the container - each end of the internal network connection used port 50000 to communicate with the other "computer".)

III.Parallel Databases

A. Requirements for serving large numbers of users have led to the development of PARALLEL systems, often using hundreds of CPU's in a data center, connected by a local network.

Though this model resembles the centralized model in terms of the way users are serviced (e.g. by a client-server model using the web), it differs in some important ways from the original centralized model.

1. The use of massive parallelism - scores, hundreds, or even thousands of multicore CPU's housed in a data center that communicate with each other via an internal network rather than via shared memory.

2. The possibility of using fine-grained rather than coarse-grained parallelism.

    a. In coarse-grained parallelism, each transaction is run on a single CPU, with each CPU potentially running a different transactions.

    b. In fine-grained parallelism, an individual transaction may be spread across several CPU's

3. The use of a parallel file system rather than a single file system shared among the CPU's.

    a. Some variants of RAID are a simple form of this, but more massive parallelism is generally needed.

    b. Frequently, data is sharded on some basis to spread out the load among the parts of the storage system.

        i. As a simple example, data on customers might be spread among multiple storage subsystems on the basis of their customer ID, so that data on customers whose id ends with 1 might be stored in one subsystem; data whose id ends with 2 might be stored in another subsystem ...

        ii. Certain NoSQL database models facilitate sharding, since all the data associated with a given entity is stored in a single aggregate rather than being spread over multiple files.

    c. To improve performance, some form of replication of data can be used so that the data storage system holds multiple copies of part or all of the data (at least data that is frequently needed.)

        i. This has an obvious advantage in terms of protection against loss of data due to hardware failure (as we saw with RAID).

        ii. It also ensures that reading frequently accessed data doesn't become a bottleneck.

        iii. At the same time, it also creates issues when writing data to ensure consistency among all the copies - something which we won't have time to discuss.

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

a. In general, we are interested in two measures of overall system performance.

   i. Response tine (primarily an issue with systems that have some sort of interactive aspect)- defined as the amount of time that elapses between the time a request is submitted and the time the user begins to see the answer to the request - e..g in a web system, the time between a click on a link and the new page begins loading.

   ii. Throughput - the number of transactions that can be completed in a unit of time.

b. In a parallel system, we aim to achieve one or more of several improvements.

   i. One possible goal is SPEEDUP - to make the processing of individual transactions (of the same size) faster. This would, of course, generally require fine-grained parallelism - the use of two or more CPU's and or portions of the data storage subsystem to cooperate in the performing of a single transaction.

   ii. 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 deals increasing the SIZE of individual transactions, as would occur if transactions become more complex and/or if the size of a database grew, so that operations such as select and join require scanning more tuples. This, too, entails fine-grained parallelism - the use of two or more CPU's and or portions of the data storage subsystem to cooperate in the performing of a single transaction.

      (b) TRANSACTION SCALEUP involves 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 and portion of the data subsystem, but by using multiple CPU's and perhaps replicated data to increase the number of transactions that can be processed during a given period of time.

(c) Of course, some combination of these two might apply in a given situation - e.g. both more and bigger transactions.

Example: suppose Gordon grew to 50,000 students. If this were to happen, the course registration system would have to deal with more registration operations at the same time, and each transaction would presumably have to deal with a much larger list of course offerings. Speed might not increase, but hopefully it would not decrease!

c. For a variety of reasons (discussed in portions of the text we cannot get to), efficiently dividing the work of a single transaction among two or more CPU's is very challenging. 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. (e.g. what happens on an ecommerce site such as Amazon with recommendations like "you might also like ..." or "others who looked at this item also looked at ...". Speedup is usually less of an issue.

5. Both CPU and data storage parallelism create algorithmic challenges for things like consistency maintenance and concurrency-control measures. As a result, parallel database systems are major pieces of software engineering!

B. However, the centralized approach - whether realized using a single system or a data center of systems, has a number of disadvantages:

ASK

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

2. It may be that a centralized storage and manipulation of data goes hand-in-hand with centralized CONTROL of data - users have limited autonomy.

3. Issues like these have motivated the development of distributed database systems,

IV.Distributed DBMS's

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

1. In the use of parallelism 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, but parallelism is used to handle increased size and/or volume of transactions.

2. In a distributed system, the database itself is spread over a number of physical sites, each of which houses all or a portion of the database - with all or portions replicated at multiple sites.

   a. This may be done at the data storage level, resulting in various sorts of distributed file system - where the data belonging to a single file may replicated and/or sharded across multiple physical sites.

      i. In addition to supporting scaleup, a distributed file system may also address he issue of vulnerability to failure of a single site if the distribution model is such that all of the information in the database is available at multiple sites.

      ii. Distributed file systems are discussed in portions of chapter 21 of the book which we will not cover in this course, but you are certainly encouraged to read it on your own, perhaps after this course is over.

b. This may be done at the DBMS level, where DBMS's at multiple sites work together

i. PROJECT Figure 20.9 from book

ii. We will introduce distributed databases now, and discuss some of the issues that arise in this context.

B. Distributed DBMS systems are of two general types.

1. In a HOMOGENEOUS system, the same brand of DBMS software (and often the same type of hardware platform) is used at each site, and the database schema is basically the same.

All the nodes in a homogeneous systems generally belong to the same large organization (such as a multistate bank), and are designed to accomplish organizational goals such as increased availability in the face of threats of site failure.

2. In a HETEROGENOUS system, different brands of DBMS software may be used at different sites, and the database schemas may be quite different.

a. Heterogenous systems may arise as a result of organizational mergers that require merging of databases, or as a result of desires for improved inter-organizational communication.

b. If the schemas of the databases differ, it is may be necessary to use strategies like wrappers to create the appearance of a common scheme.

C. Distributed DBMS's are characterized by a much looser coupling between systems.

1. This facilitates increased gains through parallelism, but also requires dealing with the resulting algorithmic.

2. This also introduces complexities due to communication overhead and the possibility of failure of a network link.

D. Some of the key advantages of a distributed system are

1. Sharing of data generated at the different sites, without requiring that all the data be moved to a single central site.

2. The possibility of LOCAL CONTROL and AUTONOMY.

   Within boundaries established by the need for sharing, each site may be able to control its own data, determine what is stored how etc.

3. The possibility of improved response times to queries. As over-against a centralized system, a distributed system that stores data at the site(s) that use it the most allows them to access the data more quickly than they would if they had to get the data from a central site via a communication link.

4. With the rise of reliance on the internet and ecommerce, a fourth motivation that has always been there have become especially prominent: availability and robustness in the face of failures at a site or portion of the network.

E. Distributed systems also face a number of disadvantages and challenges.

1. One major disadvantage of a distributed system is the cost and time required for communication between sites.

   a. This is not necessarily a disadvantage when remote access to the database is needed, if the alternatives are a centralized system where ALL queries require communication vs a distributed system where SOME queries can be processed locally at the requesting site.

   b. But operations requiring access to data at multiple sites will almost always involve more communication between sites than would be required if all the data involved were at one location.

c. The performance impact of communication depends a great deal on what kind of communication links are used.

   In particular, note that the data rate of a network is determined by the slowest link. If the Internet is used, this is often the "last mile" connection between a DBMS site and the ISP.

d. The communication cost includes both the time to set up a message and the time to actually transmit it. This may entail passing through several nodes (recall discussion in CPS221), with overhead for each.

e. Depending on the configuration, communication cost may dominate disk access cost, in which case a distributed systems might need to be optimized to minimize the number and volume of messages, rather than disk accesses or may need to consider various tradeoffs - reduced disk accesses at the cost of more communication or vice versa.

2. A second disadvantage is increased complexity. Choosing a query processing strategy, performing updates, dealing with crashes, and concurrency control are all much more complex in a distributed system.

3. A third disadvantage related to the second is that distributed systems are much harder to debug. In fact, the algorithms used must be bug-proof; discovering the cause of a problem that arises under certain circumstances of operation timing is not possible using conventional debugging techniques.

F. Two key issues involved in setting up a distributed system are replication and sharding.

1. Replication refers to storing the same data at more than one location, both to facilitate processing and to ensure that the data does not become unavailable if a single site fails due to something like a power failure or a disaster. (Should that occur, the rest of the system should be able to continue processing and providing access to the data, perhaps offering only lower performance.)

a. If most of the access to the data is in the form of queries, with updates being rare, then replication does not introduce any problems.

   For example, a product catalog may be replicated at multiple sites to protect against having a single point of failure. The catalog may be updated wholesale periodically, with a copy of the new catalog sent to all the sites that hold replicas. At the precise moment an update is being done, there may be a brief period of inconsistency, but this is not problematic.

b. On the other hand, if replicated data is updated by frequent transactions, then all kinds of issues need to be considered such as those related to support for atomic transactions and concurrency.

2. Sharding refers to spreading portions of the data across multiple sites - perhaps to more naturally mirror the uses of the data and to support distributed control of it.

3. It is often the case that the same data is both replicated and sharded.

G. The CAP Theorem

1. There are three desirable characteristics in distributed database system that replicates data.

   a. Consistency - each replica of a data item give the same value when accessed. This implies that when the value of some data item is updated at one site, it must be updated everywhere else - which cannot happen instantaneously, of course - but we must ensure that no one sees a "stale" value of a data item that is being updated. This implies that some mechanism must prevent access to a data item elsewhere while it is being changed somewhere.

   b. Availability - defined this way in this context: ""Every request received by a non-failing node in the system must result in a response" [Lynch and Gilbert cited by Sadalage and Fowler]). In particular an update to a replicated data item requires access to most or all of the replicas to ensure consistency,

c. Partition tolerance - in a distributed system where nodes communicate via a network, failure of a portion of the network may result in partitioning of the network into two or more subnetworks with no way for nodes in one subnetwork to communicate with nodes in the other. Since this is unavoidable, the system must be able to function correctly despite this happening.

2. A theorem known as the CAP theorem says that in a distributed system, you can have any two of these three desirable properties, but you can't have all three. Since partitions are a consequence of physical factors beyond software control (power failures, tornados, etc.), this boils down to the possibility that we can have either consistency or availability - but not both.

3. Obviously, in some cases doing without consistency is not an option. (Example: a banking system.)   But in other cases, it may be desirable to forgo absolute consistency to ensure availability even in the case of a network partition.

   a. In this case, ensuring ACID transactions may not be possible. Some relational DBMS's (e.g. MySQL) allow the user choose whether or not ACID transactions are needed,   (Forgoing ACID transactions will improve throughput and/or performance.)

   b. There is also a family of non-relational DBMS's (collectively called NoSQL databases) which typically do not support ACID transactions.

   c. One alternative to strict ACID consistency in such systems is sometimes called BASE - Basically Available, Soft State, Eventually Consistent.

V. Cloud Computing

A. Historically, an organization relying on a database system has owned its own hardware - whether this be centralized, a server, or distributed.

B. Increasingly, many users are moving to the use of cloud computing, in which work the user contracts with a vendor to provide services using computers in a server farm somewhere in the cloud, rather than purchasing and managing their own computer. The user pays the vendor a fee based on how much the user needs/uses.

C. Cloud services are of three general types:

1. Infrastructure as a service (IaaS) - the vendor furnishes and maintains a basic computer system, with the user responsible for installing and managing whatever software is needed (including the OS, web servers, DBMS and application software), to accomplish their task.

2. Platform as service (PaaS) - the vendor furnishes and maintains a complete platform (e.,g. hardware, OS, networking, DBMS or whatever) with the user installing and managing application software that runs on top of this platform.

   Example: Amazon Web Services (Note how you sometimes see AWS in URL's for pages linked from other sites. That is because the entity you are dealing with uses AWS cloud services to furnish the lower layers of their system.)

3. Software as a service (SaaS) - the vendor furnishes a platform and application software which the user uses to accomplish a specific kind of task.

   Examples: Gordon's email system; Canvas

D. Pros and cons

1. There are several key motivations for use of cloud computing.

a. Ir allows greater equipment flexibility. If an organization's needs grow, it can request its service provider to provision additional units of service If the needs shrink, it can cut back on what it buys. All this without having to commit to major capital investment in equipment and/or having to deal with selling off unneeded equipment.

b. Related to this, the cloud provider is more likely to keep on top of hardware upgrades and software updates.

c. An organization does not need to have staff with the technical capabilities needed to maintain aspects of their system that they contract out. Furthermore, it is generally the case that the technical staff needed to service many clients is smaller than what would be needed if each client had their own staff.

d. Cloud providers often have technical staff on site 24-7, so that a problem can be addressed more quickly than if one of the organization's own staff had to rush to the site in the middle of their night.

2. Of course, there are also concerns:

a. Security of sensitive information that is stored off-site. This can be a particular concern with information that is legally protected.

b. Vulnerability to service provider problems (e.g. financial problems) which may make then unable to deliver the services they contracted to.