Module 1: Introduction to Distributed Database Systems

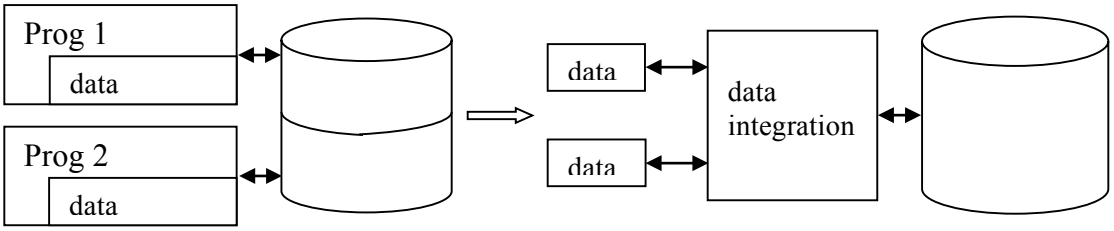
Introduction

Distributed Database Systems are composed of both *database systems* and *computer networks.* On the surface, the concept of a distributed database system seems like an oxymoron. Database systems emphasize centralizing the control of access to data, while computer networks emphasize distribution of control.

The reality is that databases are not about centralizing the management of data. They are really about integrating data and supplying a common access method to data.

The Early Days – Before Database Systems

Early on, application programs stored their data in system files. Sometimes, multiple programs accessed the same data, but access to the data was not well coordinated because the data was just stored in files. Each program used its own view, or data description, of the data to interact with the data files. Database Management Systems were invented to integrate and control access to data that may be common to multiple applications. See the figure below:



Distributed Processing

Distributed processing means many things to many people. It can include:

- Distributed functions: a single program is distributed across multiple processors.
  - Here we can think of a complex program distributed over multiple core processors.
- Distributed computing: autonomous functions are distributed across a network.
  - Here we can think of distributed software agents monitoring multiple pieces of equipment in a nuclear power plant. The agents communicate their health and status to a controller (or even to each other) to ensure the proper running of the plant.
- Networks: independent of function
- Multiprocessors: multiple CPU's in the same computer

In any computer system, there are always aspects of distribution including CPUs, I/O, printers, etc.

Distributed Computing

A *Distributed Computing System* is a set of autonomous processors that are interconnected by a computer network and that cooperate in performing an assigned task. Processors must be able to execute a computer program.

Many things can be distributed:

- Processing logic
- Function
- Data
- Control

For distributed database systems, all of these are distributed.

Why Distribute Systems?

It is difficult to think of complex systems that are not distributed. For example, distribution can be found in:

- Cloud systems
- Smartphone apps
- Organizational enterprises
- E-commerce systems
- Server farms
- Multimedia applications
- Manufacturing control systems

As we will see throughout the course, there are many benefits to distributed systems.

- Provide reliability through duplication
- Provide better response time because different machines are geographically closer to different users (locality of reference)
- Support complex data integration in ways that could not be supported by a single system
- Provide organizations with more independence and autonomy over systems for which they are responsible
- Provide more computational horsepower through divide-and-conquer methods
- Reduce the cost of software

So What Is a Distributed Database System?

A *Distributed Database System* is a collection of multiple, logically interrelated databases distributed over a computer network.

A *Distributed Database Management System (DDBMS)* is the set of software systems that manages distributed databases and makes the distribution transparent to the user.

Watch Module 1A video – What is a distributed database?

This course covers the theoretical foundation of implementing *distributed database systems*!

Advantages of Distributed Database Systems

Transparent Management of Distributed and Replicated Data

A distributed database management system (DDBMS) presents users with an interface that makes the distributed databases appear as a single, coherent database. However, under the covers, the DDBMS must transparently integrate the underlying data.

Transparency hides many of the lower-level implementation issues. Users and applications do not need to have an understanding of the complexities of integration – they just manage the DDBMS as if it were a single database system. This means the DDBMS must know where all the data is located, the structure of each schema, how the distributed data is related across the data systems, and how to efficiently pull the right data together from the right systems in an efficient manner to answer queries. In addition, the DDBMS handles data replication properly and automatically.

A significant part of the course will discuss how a DDBMS manages query processing consistently and automatically.

Reliability through Distributed Transactions

DDBMSs must handle multiple users' transactions to ensure correctness. The standard for correctness in database transactions is called the ACID properties.

- Atomicity – all updates to a DDBMS in an individual transaction must operate as an atomic operation. Either they all work or none work.
- Consistency – a transaction takes the DDBMS from one consistent state to another consistent state.
- Isolation – a transaction in a multiple user DDBMS operates as if it were the only transaction occurring. Effects of other user transactions are not noticed.
- Durability – a completed transaction will permanently change the database, even if the system crashes.

This topic will be discussed thoroughly later in the course. However, maintaining ACID properties would be nearly impossible without a DDBMS.

Improved Performance

DDBMS algorithms are tuned for distribution, from the perspectives of both schema design and query evaluation. Database schemas are designed and distributed to

optimize usage patterns. This means that understanding how a database will be used will drive the distributed design. If schema design were left up to each user, it would be nearly impossible for the DDBMS to perform optimally.

Distributed queries are executed with high efficiency to optimize the response time for users. The DDBMS generates efficient query plans based on internal statistics that describe data values, size, and distribution. Here, too, formulating efficient query algorithms and designing databases optimally are beyond the capability of most users. Even if users have the ability to formulate optimal queries, they do not have access to internal DDBMS statistics to make effective design and query choices.

Easier System Expansion

Adding new nodes to a distributed database system is straightforward. Each new database node fits into a pre-existing architecture where much of the database integration software is already in place. The new database node is managed consistently within the context of the other database nodes.

Without a DDBMS, all applications needing data at the new node would need to be modified and tuned to the specifics of that database, including access patterns, query semantics, and transaction management. Much more of this will be discussed as the semester progresses.

Transparency Issues

- Distribution of data
- Replication of data
- Data independence
- Network
- Fragmentation

Watch Module 1B video – Transparency Issues

Why the Computer Network is Important

Distribution is an important aspect of *distributed* database systems. When users formulate queries, data is going to be brought together from different sites via the network. Therefore, it is critical to understand the effects that the network has on queries. Thus, we must delve into the aspects of computer networks that will be important to us later in the course when we discuss the performance aspects of query planning and execution.

Network Bandwidth limits the amount of data that can be shipped between processors. It is an important limiting factor in the executing speed of distributed joins. In fact, for all distributed joins, there are multiple query execution plan options. Knowledge of the network bandwidth limitations will help a query planner estimate the best query execution plan.

Network Latency limits the start time of distributed operations. If there is a choice of network paths to use, the latency can affect which path is selected. It can play a part in the order of operations in a query plan. The effect of both latency and bandwidth,

especially the effects they have on query planning and execution, may ultimately guide the design of the distributed database schema to compensate for network limitations.

Problems this Course Addresses

- Distributed Database Design
- Distributed Query Processing
- Distributed Directory Management
- Distributed Concurrency Control
- Distributed Deadlock Management
- Reliability of Distributed DBMS
- Operating System Support
- Heterogeneous Databases
- Scaling Databases to Very Large Sizes

Watch Module 1C video – Problems this Course Addresses

The Relational Model

The relational model is perhaps the most popular model for structured data. It is also the data model that has the most rigorous mathematical underpinnings. There are many textbooks that describe these models in detail. The following paragraphs will provide a very rough overview.

Many algorithms exist in relational database management systems that can be augmented to support distributed database systems. This course will take full advantage of the richness of relational database theory and apply it to distributed databases.

Other Popular Data Models

The last half of this course will explore large-scale parallel architectures and algorithms that operate on them. These architectures support data sizes (such as in the petabyte range) that cannot be supported by traditional distributed relational database systems because of their size. These systems are generally not transactional and therefore do not support ACID properties, concurrency control, and reliability protocols. In addition, these systems execute algorithms, not queries. For example, the query systems that run on the Hadoop ecosystem, such as HiveQL, do not run queries per se. Rather, they execute MapReduce jobs that are algorithms that run in parallel. Therefore, query optimization is less of an issue. However, a benefit of these systems is that they support many different types of data structures including:

- XML and JSON data (hierarchical data structures)
- Flat-file, Excel spreadsheets
- Graphs (nodes and edges)
- Knowledge bases (OWL, triple stores, etc.)

Specifically, we will discuss the Hadoop ecosystem because of its popularity.

Definition of Some Concepts

A ***database*** models some real life concept and captures actual data that represents some state of real life.

A ***relational database*** is a database modeled by relations.

A ***relation*** R defined over n sets D_1, D_2, … D_n, where D_i represents some domain.

An ***n-tuple*** (tuple) is a set < d_1, d_2, …, d_n > where d_1 e D_1, d_2 e D_2, …

D_1	D_2	D_3	D_4	…
d_{11}	d_{12}	d_{13}	d_{14}	…
d_{21}	d_{22}	d_{23}	d_{24}	…
d_{31}	d_{32}	d_{33}	d_{34}	…
…	…	…	…	…

In the above relation, or table, the domains D_1, D_2, … are represented by the first row. The tuples, or rows, represent groups of values d_1, d_2, … in their respective domains. A table can have 0 or many rows.

Some example tables are:

- CUSTOMER (cust_no, cust_name, cust_address)
- ORDER (cust_no, part_no, quantity)
- PART (part_no, part_name, manufacturer, cost)
- MFG (manufacturer, owner)

These represent four tables. The customer table has three domains: **cust_no, cust_name**, and **cust_address**. The domain **cust_no** is underlined to indicate that it is a key.

A **key** is a domain (or a set of domains) that functionally determines the non-key domains in a row. In the CUSTOMER table,

CUSTOMER: cust_no -> (cust_name, cust_address)

Similarly, for the other tables:

PART: part_no -> (part_name, manufacturer, cost)

ORDER: (cust_no, part_no) -> (quantity)

MFG: (manufacturer) -> (owner)

Watch Module 1D video – Normalization Addresses Anomalies in Database Systems

Relational Algebra

In many modules in this course, we will express queries using relational algebra expressions. It will be the language we will use to describe distributed database design as well as query planning and execution.

Watch Module 1E video – Relational Algebra Notation

Distributed Database Design

Computer Architectures

Computer architecture is the conceptual design and fundamental operational structure of a computer system. It is the functional description of that system including:

- Functional requirements
- Design implementations for the various parts of a system
- System performance including speeds and interconnections

Architecture typically refers to the internal structure of the system that enables logical operations. Architecture design issues revolve around:

- Tradeoffs between cost and performance
- Reliability
- Feature set
- Expandability

A database management system architecture is usually based on a reference model, which is an idealized architecture model. It is a conceptual framework that divides the system into manageable pieces and defines their interrelationships. Database system architectures can be described from multiple perspectives:

1. A *Component Architecture* describes components of a system and their interrelationships. Each component provides functionality and their interaction provides overall system functionality. This is the best decomposition if you want to build a system. However, it is not the best representation to help understand the system.
2. A *Functional Architecture* defines different classes of users and the functions that they can perform. Often, user classes are decomposed hierarchically. The ANSI/SPARC architecture in the figure below is an example. However, this architecture perspective does not help you build the system nor does it help you understand the complexity of the system
3. A *Data Architecture* represents both the data and the views of the data. An architecture framework helps understand how they are actualized. Since data is the central resource of a database management system, this is the representation of choice for us. However, to fully define the system, one also needs the functional and component architectures.

(Watch Module 1F video – ANSI/SPARC Architecture)

Architecture Models for Distributed Databases

There are three orthogonal categories of description of distributed database architectures:

- Autonomy – describes the level of data independence among participating individual databases
- Distribution – describes the level of the distribution of data
- Heterogeneity – describes the differences in participating individual databases

(Watch Module 1G video – Distributed Database Architecture Categories)

(Watch Module 1H video – Multi-Database Management Systems)

Conclusion

In this module, we provided an introduction to distributed database systems that will serve as the basis for the rest of the course. We discussed issues that concern a distributed database. We discussed relational algebra notation that will be used throughout the course. In addition, we provided an understanding of the role of data architectures that provide frameworks for distributed database systems. Data architectures focus on user views; data views; and levels of autonomy, distribution, and heterogeneity. The tradeoff of these issues will drive most of this course in terms of both distributed database design and distributed query processing algorithms.