Distributed Database Systems - Introduction

Diego ROJAS

October 9, 2022

Contents

1 Distributed data processing

A *distributed computing system* consists in a number of autonomous programs interconnected by a computer network.

Processing logic, *control*, *function* and *data* can all be distributed within a system.

2 What is a Distributed Database System?

A *Distributed Database System* (DDBS) can be defined as a collection of multiple, logically interrelated databases distributed over a computer network.

A *Distributed Database Management System* (DDBMS) is defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users

3 Data Delivery Alternatives

In a DDBS, data delivery consists in three things.

- The *delivery mode* (*push only*, *pull only*, *hybrid*)
- The *frequency* (*periodic*, *conditional*, *ad-hoc/irregular*)

- The *communication method* (*unicast*, *one-to-many*)

4 Promises of DDBSs

4.1 Transparent Management of Distributed and Replicated Data

DDBSs are *transparent* meaning the implementation details are hidden from the end-user.

DDBS may enforce:

- *Data independence* whereby the *logical representation* (schema definition) is decoupled from the *physical representation* (on-disk storage).
- *Location transparency* whereby the storage location is hidden.
- *Network transparency* whereby the network implementation is hidden.
- *Naming transparency* whereby the names are unique.
- *Replication transparency* whereby replication (or lack thereof) is hidden.
- *Fragmentation transparency* whereby possibly fragmented data behaves as if whole.

Full transparency may hurt performance, data management, modularity and more. Query languages, operating systems and DDBMSs may provide different levels of transparency to manage the tradeoff between ease-of-use and performance.

4.2 Reliability through distributed transactions

DDBSs prevent SPoF. The failure of a single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system.

DDBSs implement transactions which guarantees *data consistency*.

4.3 Improved Performance

Data localization whereby data is stored next to its point of use, increases the performance of a DDBS.

Contention for I/O and CPU is limited compared to centralised databases.

4.4 Easier System Expansion

Increase in database size can be addressed by allocating additional nodes to a distributed database cluster.

While mainframes are still very useful, a collection of small machines is sometimes more economically viable.

5 Complications Introduced by Distribution

- Data replication implies keeping the data consistent when it's updated.
- Failure of a component must not break data consistency.
- Synchronisation of operations is considerably harder than for a centralised system.

6 Design Issues

6.1 Distributed Database Design

Within a DDBS, data can be:

- *Replicated* where the entire database is stored in each node. Databases can also be *partially replicated*.
- *Partitioned* (or *non-replicated*) where each node contains a disjoint subset of the data.

Fragmentation refers to the separation of the database into partitions called *fragments*. Fragmentation can be horizontal (*selection*) or vertical (*projection*).

Distribution is the optimum distribution of fragments.

Fragmentation and distribution aim at lowering cost of storage, communication overhead and processing of transactions. This problem is NP-hard therefore it is usually solved using heuristics.

6.2 Distributed Directory Management

Must define if and how directories are fragmented across a DDBS.

6.3 Distributed Query Processing

Must define when and how to distribute queries to maximize performance.

6.4 Distributed Concurrency Control

One of the most researched issues introduced by DDBSs.

DDBSs must ensure that all the values of multiple copies of every data item converge to the same value. This is called *mutual consistency*.

6.5 Distributed Deadlock Management

Must define how locking is used to avoid deadlocks.

6.6 Reliability of Distributed DBMS

Must ensure failure of a location or communication failure between two locations is accounted for and doesn't bring the system down.

6.7 Replication

If the database is replicated, the consistency of replicas becomes an issue. Replication can either be *lazy* whereby a transaction completes before the result of the transaction is propagated to all the sites or *eager* whereby all updates must be propagated before the transaction returns.

6.8 Relationship among Problems

The aforementioned problems are closely related. For example, concurrency control is linked to deadlock management. The solution to one problem can affect the solution to another.

7 Distributed DBMS Architecture

There are three majors DDBS architectures.

- Client/server systems
- Peer-to-peer distributed DBMS
- Multidatabase systems

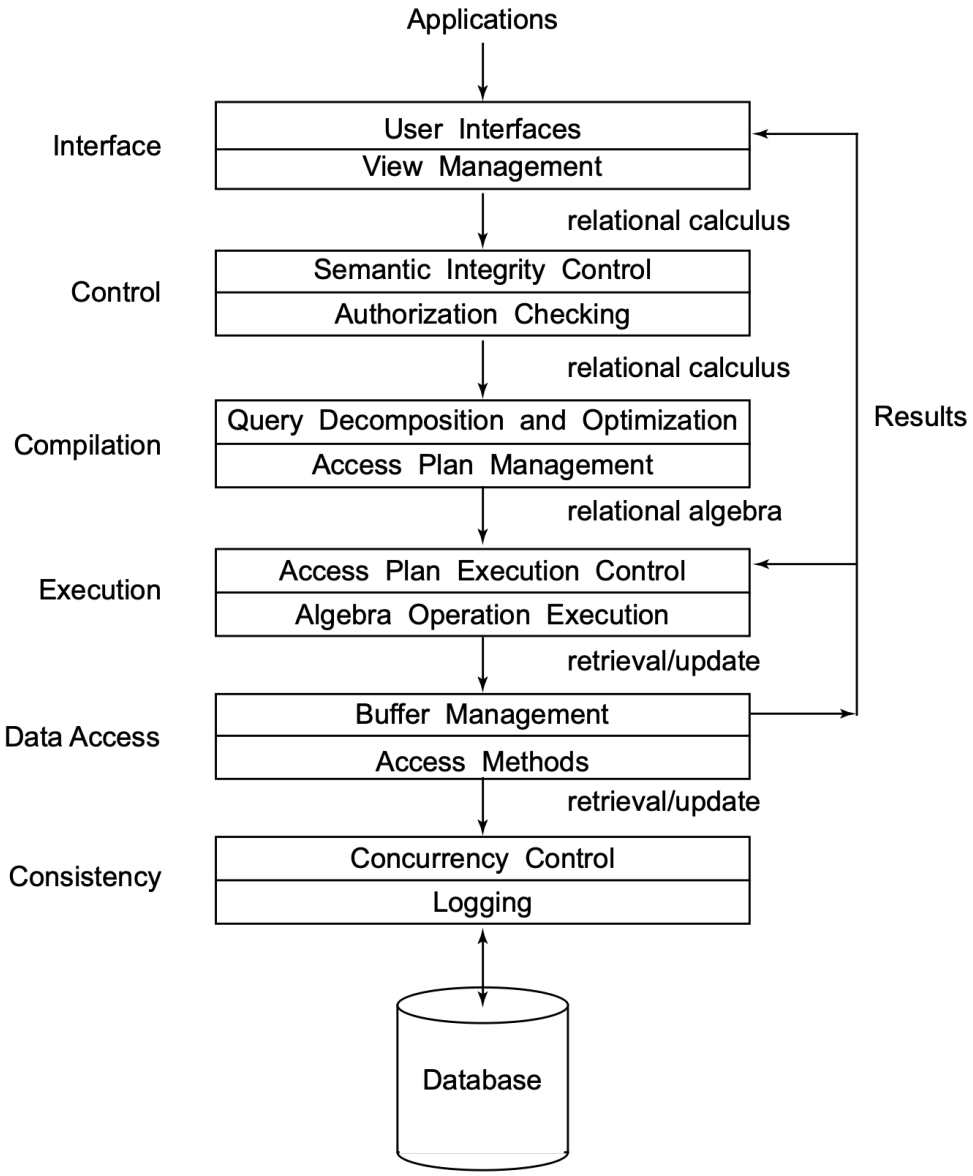
7.1 ANSI/SPARC Architecture

The ANSI/SPARC architecture defines three views of a database:

- *External view* which is that of the end user.
- *Conceptual view* which is that of the database.
- *Internal view* which are the access paths and physical representation.

This provides the basis for data independence as one view can change without affecting the others. The separation of the external schemas from the conceptual schema enables *logical data independence*, while the separation of the conceptual schema from the internal schema allows *physical data independence*.

7.2 A Generic Centralized DBMS Architecture



7.3 Architectural Models for Distributed DBMSs

7.4 Autonomy

Autonomy refers to the distribution (or decentralization) of control. The autonomy of a DBMS can span from *tight integration* to *semiautonomous* to *total isolation*.

7.5 Distribution

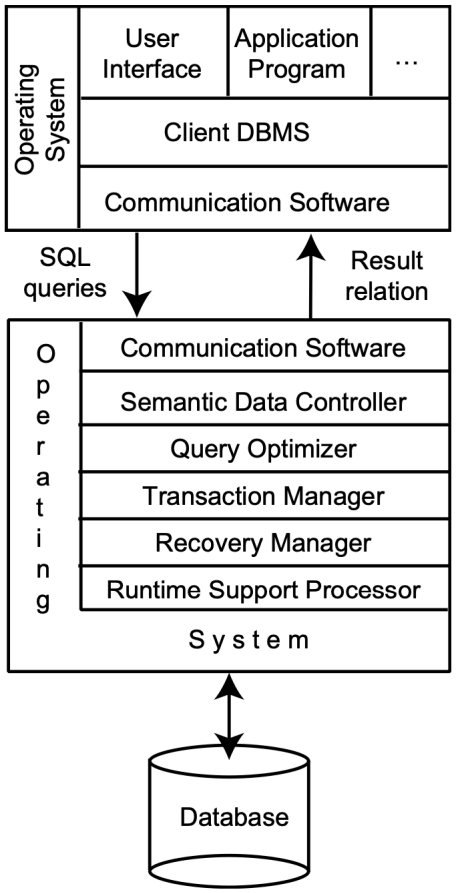
Distribution is split into two classes: *client-server* distribution and *peer-to-peer* distribution (also called *fully distributed*).

7.6 Heterogeneity

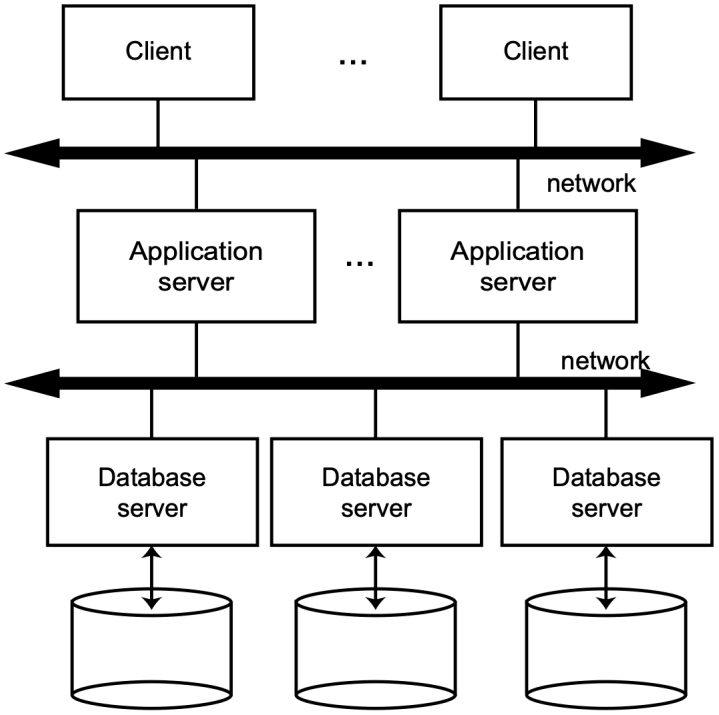
Can be found in hardware, networking protocols and data management.

7.7 Architectural Alternatives

7.8 Client/Server Systems



In the *multiple servers/multiple clients* approach, clients can either intentionally connect to the relevant server (*heavy clients*) or connect to a "home server" which then communicates with other servers as required (*light clients*).

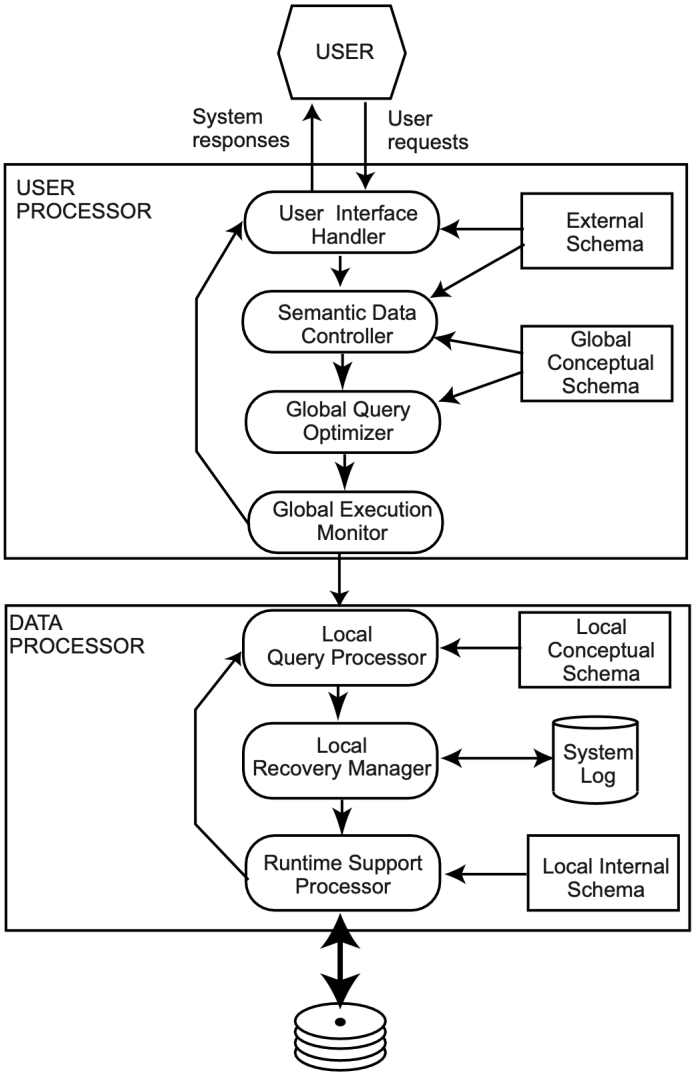


This is what a distributed client/server architecture would look like in the traditional three-tiered web setting.

7.9 Peer-to-Peer Systems

In peer-to-peer systems, there is generally no differentiation between the functionality of each site in the system.

Each site is organised in two parts within the same machine: the *user processor* and *data processor*. Here is an overview of the components that make up each part.


USER
System responses
User requests
USER PROCESSOR
User Interface Handler
External Schema
Semantic Data Controller
Global Conceptual Schema
Global Query Optimizer
Global Execution Monitor
DATA PROCESSOR
Local Query Processor
Local Conceptual Schema
Local Recovery Manager
System Log
Runtime Support Processor
Local Internal Schema

7.10 Multidatabase System Architecture