UNIT I DISTRIBUTED DATABASES 9

Distributed Systems – Introduction – Architecture – Distributed Database Concepts – Distributed Data Storage – Distributed Transactions – Commit Protocols – Concurrency Control – Distributed Query Processing

---

DISTRIBUTED DATABASES

What Are Distributed Systems?

A distributed system is a **computing environment in which various components are spread across multiple computers (or other computing devices) on a network**. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task.

How do distributed systems communicate with each other?

Distributed System Architecture Distributed systems must have a network that connects all components (machines, hardware, or software) together so they can transfer messages to communicate with each other. That network could be connected with an IP address or use cables or even on a circuit board.

One of the major disadvantages of distributed systems is the complexity of the underlying hardware and software arrangements. This arrangement is generally known as a topology or an overlay. This is what provides the platform for distributed nodes to communicate and coordinate with each other as needed.

1. What is a Distributed System
2. Distributed System Architectures
3. Architectural Styles
4. System Level Architecture
5. A Comparison Between Client Server and Peer to Peer Architectures 6. Middleware in Distributed Applications
7. Centralized vs. Decentralized Architectures
8. Summary on Structured and Unstructured P2P Systems

1) What is a Distributed System?

A distributed system is a software system that interconnects a collection of heterogeneous independent computers, where coordination and communication between computers only happen through message passing, with the intention of working towards a common goal.

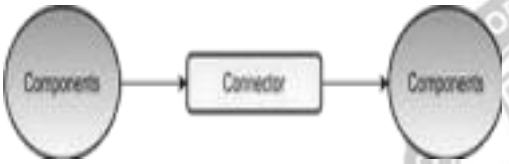
The idea behind distributed systems is to provide a viewpoint of being a single coherent system, to the outside world. So, the set of independent computers or nodes are interconnected through a Local Area Network (LAN) or a Wide Area Network (WAN).

2) Distributed System Architectures

In this blog, I would like to talk about the available Distributed System architectures that we see today and how they are being utilized in our day to day applications. Distributed system architectures are bundled up with components and connectors. Components can be individual nodes or important components in the architecture whereas connectors are the ones that connect each of these components.

 Component: A modular unit with well-defined interfaces; replaceable; reusable

 Connector: A communication link between modules which mediates coordination or cooperation among components



So the idea behind distributed architectures is to have these components presented on different platforms, where components can communicate with each other over a communication network in order to achieve specific objectives.

3) Architectural Styles

There are four different architectural styles, plus the hybrid architecture, when it comes to distributed systems. The basic idea is to organize logically different components, and distribute those computers over the various machines.

 Layered Architecture

 Object Based Architecture

 Data-centered Architecture

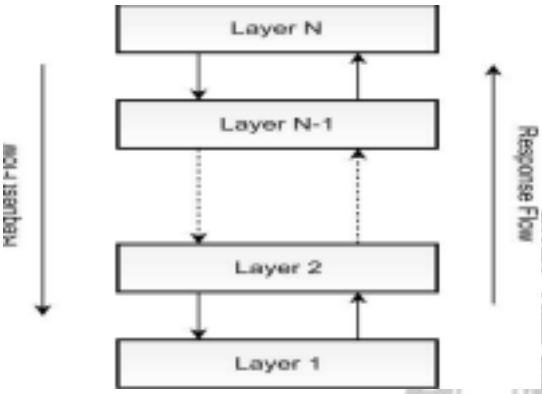
 Event Based Architecture

 Hybrid Architecture

Layered Architecture

The layered architecture separates layers of components from each other, giving it a much more modular approach. A well known example for this is the OSI model that incorporates a layered architecture when interacting with each of the components. Each interaction is sequential where a layer will contact the adjacent layer and this process continues, until the

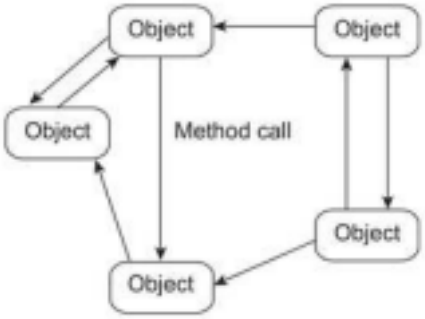
request is been catered to. But in certain cases, the implementation can be made so that some layers will be skipped, which is called cross-layer coordination. Through cross-layer coordination, one can obtain better results due to performance increase.
The layers on the bottom provide a service to the layers on the top. The request flows from top to bottom, whereas the response is sent from bottom to top. The advantage of using this approach is that, the calls always follow a predefined path, and that each layer can be easily replaced or modified without affecting the entire architecture.



Object Based Architecture

This architecture style is based on loosely coupled arrangement of objects. This has no specific architecture like layers. Like in layers, this does not have a sequential set of steps that needs to be carried out for a given call. Each of the components is referred to as objects, where each object can interact with other objects through a given connector or interface. These are much 1

more direct where all the different components can interact directly with other components through a direct method call.

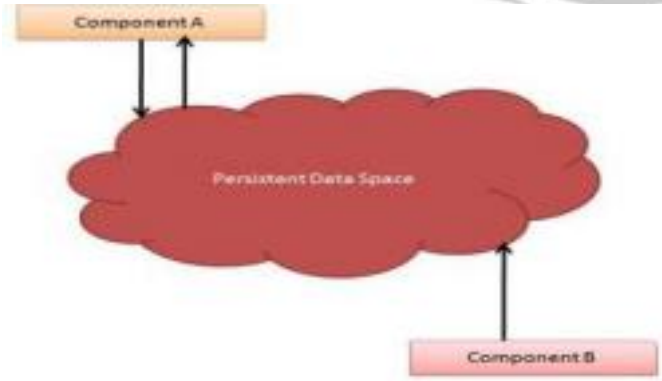


As shown in the above image, communication between object happen as method invocations. These are generally called **Remote Procedure Calls (RPC).** Some popular examples are Java RMI, Web Services and REST API Calls. This has the following

properties. This architecture style is less structured.
 component = object
 connector = RPC or RMI
When decoupling these processes in space, people wanted the components to be anonymous and replaceable. And the synchronization process needed to be asynchronous, which has led to **Data Centered Architectures** and **Event Based Architectures**.

Data Centered Architecture

 As the title suggests, this architecture is based on a data center, where the primary communication happens via a central data repository.
 This common repository can be either active or passive.
 This is more like a producer consumer problem.
 The producers produce items to a common data store, and the consumers can request data from it.
 This common repository could even be a simple database. But the idea is that, the communication between objects happening through this shared common storage. This supports different components (or objects) by providing a persistent storage space for those components (such as a MySQL database).
 All the information related to the nodes in the system are stored in this persistent storage.
In event-based architectures, data is only sent and received by those components who have already subscribed.
Some popular examples are distributed file systems, producer consumer, and web based data services.



Event Based Architecture

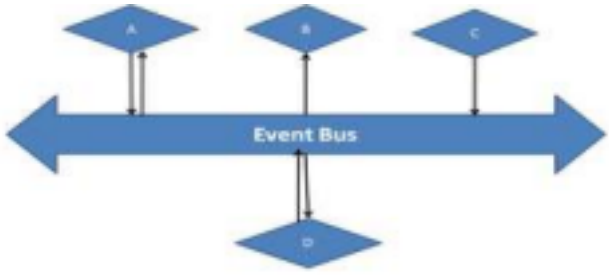
 The entire communication in this kind of a system happens through events. When an event is generated, it will be sent to the bus system. With this, everyone else will be

notified telling that such an event has occurred. So, if anyone is interested, that node can

pull the event from the bus and use it. Sometimes these events could be data, or even URLs to resources. So the receiver can access whatever the information is given in the event and process accordingly.

 These events occasionally carry data. An advantage in this architectural style is that, components are loosely coupled. So it is easy to add, remove and modify components in the system.

 One major advantage is that, these heterogeneous components can contact the bus, through any communication protocol. But an ESB or a specific bus, has the capability to handle any type of incoming request and process accordingly.



This architectural style is based on the publisher-subscriber architecture. Between each node there is no direct communication or coordination. Instead, objects which are subscribed to the service communicate through the event bus.

The event based architecture supports, several communication styles.

 Publisher-subscriber

 Broadcast

 Point-to-Point

The major advantage of this architecture is that the **Components are decoupled in space - loosely coupled.**

4) System Level Architecture

The two major system level architectures that we use today are **Client-server** and **Peer-to peer** (P2P). We use these two kinds of services in our day to day lives, but the difference between these two is often misinterpreted.

Client Server Architecture

The client server architecture has two major components.

 The client and

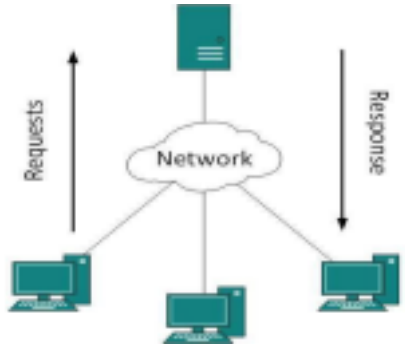
 The server.

 The Server is where all the processing, computing and data handling is happening, whereas the Client is where the user can access the services and resources given by the Server (Remote Server).

 The clients can make requests from the Server, and the Server will respond accordingly.

 Generally, there is only one server that handles the remote side. But to be on the safe

side, we do use multiple servers will load balancing techniques.



As one common design feature, the Client Server architecture has a centralized security database. This database contains security details like credentials and access details. Users can't log in to a server, without the security credentials. So, it makes this architecture a bit more stable and secure than Peer to Peer. The stability comes where the security database can allow resource usage in a much more meaningful way. But on the other hand, the system might get low, as the server only can handle a limited amount of workload at a given time.

Advantages:

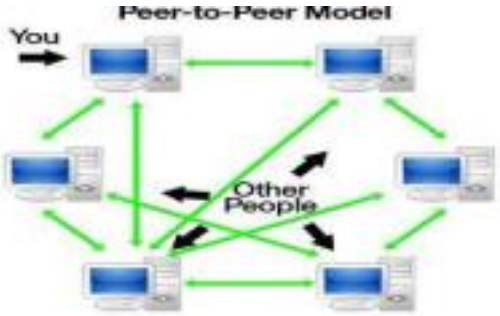
- Easier to Build and Maintain
- Better Security
- Stable

Disadvantages:

- Single point of failure
- Less scalable

Peer to Peer (P2P)

The general idea behind peer to peer is where there is no central control in a distributed system. The basic idea is that, each node can either be a client or a server at a given time. If the node is requesting something, it can be known as a client, and if some node is providing something, it can be known as a server. In general, each node is referred to as a Peer.



In this network, any new node has to first join the network. After joining in, they can either request a service or provide a service. The initiation phase of a node (Joining of a node), can vary according to implementation of a network. There are two ways in how a new node can get to know, what other nodes are providing.

 Centralized Lookup Server - The new node has to register with the centralized look up server and mention the services it will be providing, on the network. So, whenever you want to have a service, you simply have to contact the centralized look up server and it will direct you to the relevant service provider.

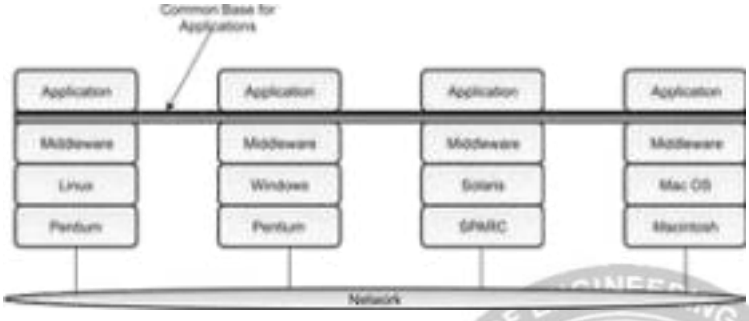
 Decentralized System - A node desiring for specific services must, broadcast and ask every other node in the network, so that whoever is providing the service will respond. **5) A Comparison between Client Server and Peer to Peer Architectures**

BASIS FOR COMAPAISON	CLIENT-SERVER	PEER-TO-PEER
Basic	There is a specific server and specific clients connected to the server.	Clients and server are not distinguished; each node act as client and server.
Service	The client request for service and server respond with the service.	Each node can request for services and can also provide the services.
Focus	Sharing the information.	Connectivity.
Data	The data is stored in a centralized server.	Each peer has its own data.
Server	When several clients request for the services simultaneously, a server can get bottlenecked.	As the services are provided by several servers distributed in the peer-to-peer system, a server in not bottlenecked.
Expense	The client-server are expensive to implement.	Peer-to-peer are less expensive to implement.
Stability	Client-Server is more stable and scalable.	Peer-toPeer suffers if the number of peers increases in the system.

6) Middleware in Distributed Applications

If we look at Distributed systems today, they lack the uniformity and consistency. Various heterogeneous devices have taken over the world where distributed system cater to all these devices in a common way. One way distributed systems can achieve uniformity is through

a common layer to support the underlying hardware and operating systems. This common layer is known as a middleware, where it provides services beyond what is already provided by Operating systems, to enable various features and components of a distributed system to enhance its functionality better. This layer provides a certain data structures and operations that allow processes and users on far-flung machines to inter-operate and work together in a consistent way. The image given below, depicts the usage of a middleware to inter-connect various kinds of nodes together.

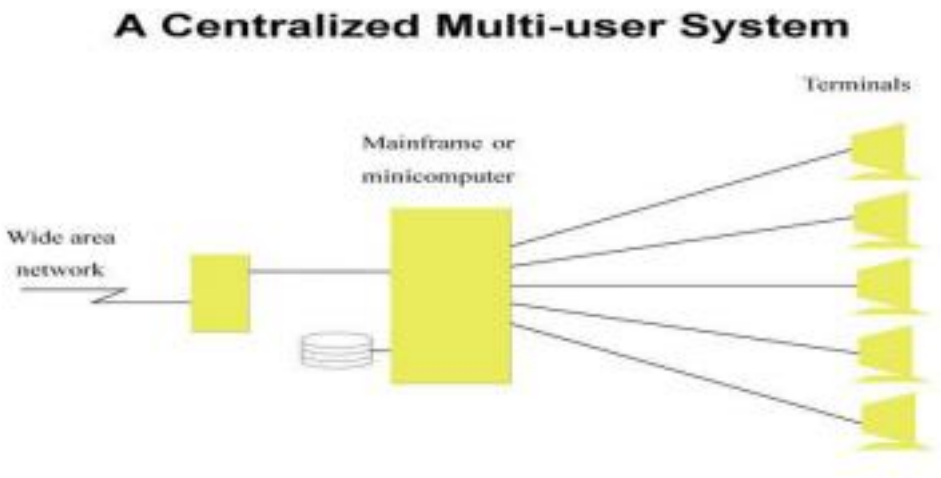


7) Centralized vs Decentralized Architectures

The two main structures that we see within distributed system overlays are Centralized and Decentralized architectures. The centralized architecture can be explained by a simple client server architecture where the server acts as a central unit. This can also be considered as centralized look up table with the following characteristics.

- Low overhead
- Single point of failure
- Easy to Track
- Additional Overhead.

1



When it comes to distributed systems, we are more interested in studying more on the overlay and unstructured network topologies that we can see today. In general, the peer to peer systems that we see today can be separated into three unique sections.

 Structured P2P: nodes are organized following a specific distributed data structure **Unstructured P2P**: nodes have randomly selected neighbors

 Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

Structured P2P Architecture

The meaning of the word structured is that the system already has a predefined structure that other nodes will follow. Every structured network inherently suffers from poor scalability, due to the need for structure maintenance. In general, the nodes in a structured overlay network are formed in a logical ring, with nodes being connected to the this ring. In this ring, certain nodes are responsible for certain services.

A common approach that can be used to tackle the coordination between nodes, is to use distributed hash tables (DHTs). A traditional hash function converts a unique key into a hash value that will represent an object in the network. The hash function value is used to insert an object in the hash table and to retrieve it.

In a DHT, each key is assigned to a unique hash, where the random hash value needs to be of a very large address space, in order to ensure uniqueness. A mapping function is being used to assign objects to nodes based on the hash function value. A look up based on the hash function

value, returns the network address of the node that stores the requested object. **Hash Function**: Takes a key and produces a unique hash value

 Mapping Function: Map the hash value to a specific node in the system **Lookup table**: Return the network address of the node represented by the unique hash value.

Unstructured P2P Systems

There is no specific structure in these systems, hence the name "unstructured networks". Due to this reason, the scalability of the unstructured p2p systems is very high. These systems rely on randomized algorithms for constructing an overlay network. As in structured p2p systems, there is no specific path for a certain node. It's generally random, where every unstructured system tried to maintain a random path. Due to this reason, the search of a certain file or node is never guaranteed in unstructured systems.

The basic principle is that each node is required to randomly select another node, and contact it.

 Let each peer maintain a partial view of the network, consisting of n other nodes Each node P periodically selects a node Q from its partial view

 P and Q exchange information and exchange members from their respective partial views

Hybrid P2P Systems

Hybrid systems are often based on both client server architectures and p2p networks. A famous example is Bittorrent, which we use everyday. The torrent search engines provide a client server architecture, where the trackers provide a structured p2p overlay. The rest of nodes, which are also known as leechers and seeders, become the unstructured overlay of the network, allowing it to scale itself as needed and further.