CMPSCI 677 Operating Systems **Spring 2017**

Lecture 2: January 24

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Phuthipong Bovornkeeratiroj***

2.1 Lecture 2

Distributed systems fall into one of the architectures teaching in this lecture. It is important to understand so that we can decide the architecture before implementing a new system.

2.2 Lecture Material

2.2.1 Types of Architectures

2.2.1.1 Layered Architectures

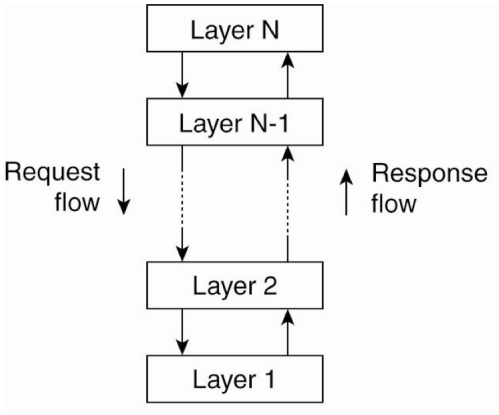

Figure 2.0: Layered Design Architecture

Layered Architecture looks like stack as seen in Figure 2.0. The system is partitioned into a sequence of layers and each layer can communicate to only an above or below layer. For example, Layer i can communicate with Layer i+1 and i-1 but not the others (e.g. Layer i+2). This architecture is especially common in web applications where this architecture is divided across the client and server.

A common instance of these systems are Multitiered Architectures.

2.2.1.2 Object Based Architecture

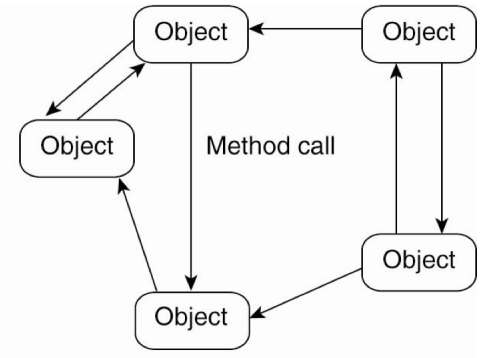

Figure 2.1: Object-based Style

The concept of this architecture is similar to object oriented programming (OOP). As shown in Figure 2.1, the system can have many objects and each object exposes its own interface which other objects can use it. All object can communicate with any other object without restriction unlike Layered Architecture.

2.2.1.3 Event-Based Architecture

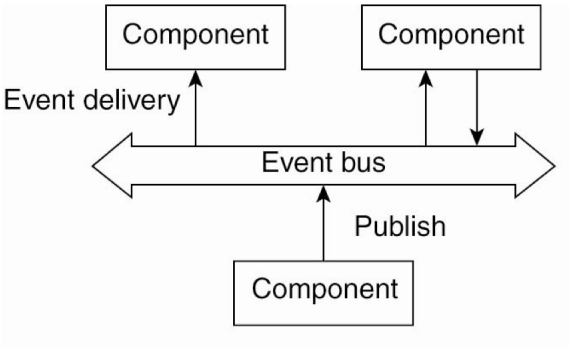

Figure 2.2: Event-based Architecture

An Event-based architecture has many components (similar to object) but each component cannot communicate with each other directly but they can communicate via event bus. It uses a publish-subscribe (producer-consumer) paradigm. In this architecture, component that send an event to the event bus is a publisher, and component that subscribes to certain types of events on the event bus is a subscribers. Each component will work asynchronously.

There are many kinds of event bus such as memory-based, or disk-based.

2.2.1.4 Shared Data Space

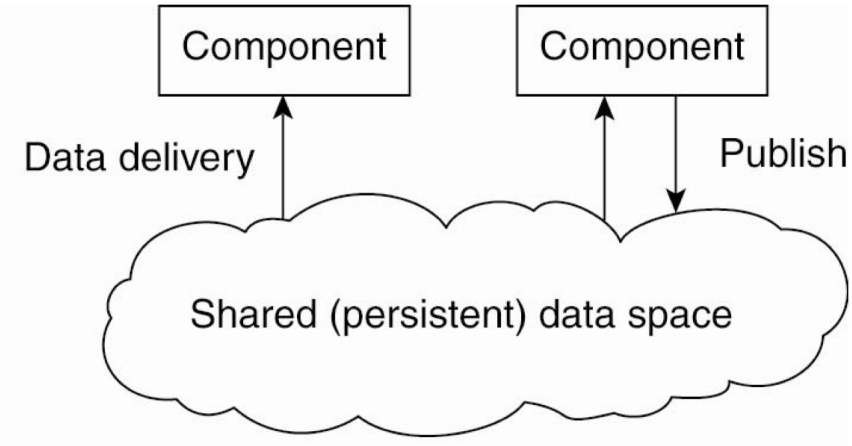

Figure 2.3: Shared Style

This architecture looks like Event-based architecture but, unlike Event-based architecture, it decouples in time and space. It can be viewed as a form of a "Bulletin-board". The publisher posts a note to the shared space, and then a subscriber will look for notes of interest. The data that is posted can be in the shared data space for a while so the subscriber may not get a real-time data. This architecture can be either push or pull model.

Decoupled in time means the data that is published is not address to anyone in particular. Decoupled in space means the data that a component can get may not be a real-time data.

2.2.2 Resource-based Architecture

Representational State Transfer (REST) is a popular architecture for web service (API). It has a standard naming scheme which all services offer same interface (4 HTTP operation which are GET/PUT/POST/DELETE).

The example in the lecture shows a link to Amazon Web Server S3 which is a simple storage and uses for web service. The web service may return in JSON format which is a key-value component. XML is another popular format in the past.

2.2.3 Client-Server Architecture

This is the most popular architecture. Client sends request to server then server sends response back to client. This is also a multitiered architectures. Application is divided into 3 layers which are User-interface level, Processing level, and Data level.

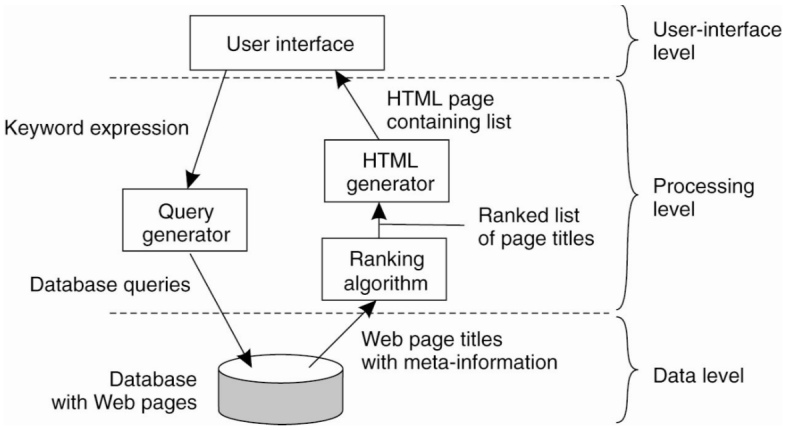

Figure 2.4: Search Engine Example

Search Engine is a good example of how application layering works.

Multitiered Architectures

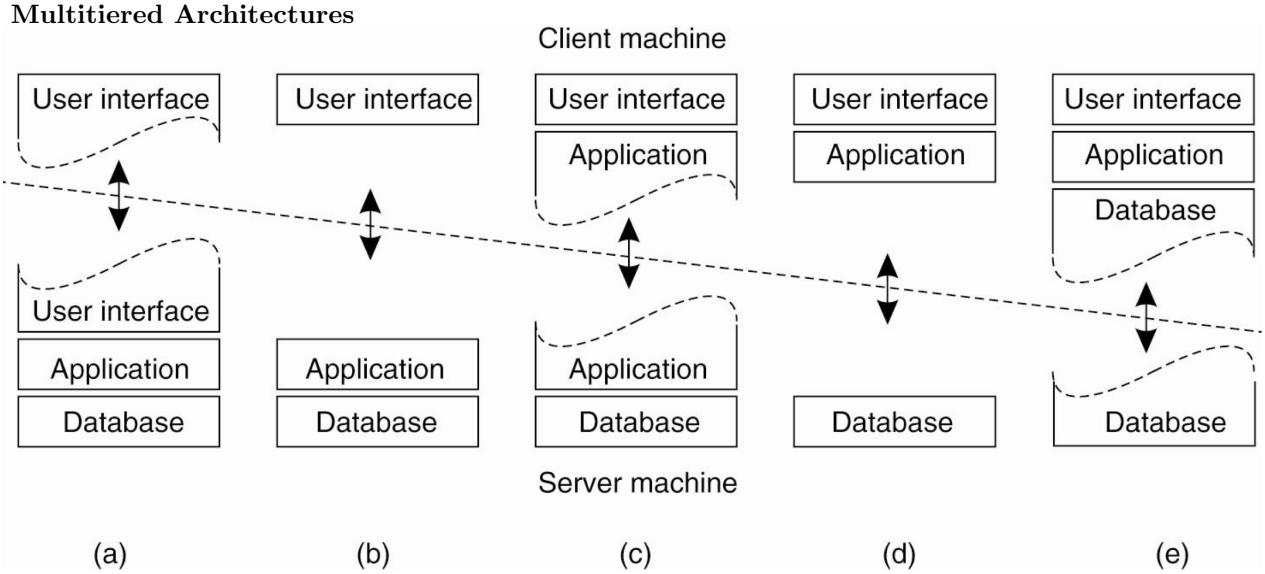

Figure 2.5: Client Server Choices

Here, (a) and (b) can represent a browser-based such as web application (e.g. Javascript) but in (a) server can perform some works on the interface such as error checking or validating the input. (c) could be an instance of a smartphone app, where the application's backend is usually split between the device and the server, or game, where the application have to be on both client and server side. Desktop applications usually follow (d) where only the database is on the server. Lastly, (e) is usually done through caching. For example, google's offline mail caches a small subset of the users email locally.

Three-tier Web Application is another good example, where the browser is on the client side, and the application and its database is on the server side:

$$\text{browser} \leftarrow | \rightarrow \text{http request} \longleftrightarrow \text{app server} \longleftrightarrow \text{database}$$

The client's browser sends a http request to HTTP server (e.g. apache). HTTP server then sends the request to the app server (e.g. Python backend) for processing in which it may create a query to the database server. The database returns data to the app server, which then app server sends the results the HTTP server which then forward it to the browser. The sequential nature of this architecture is a type of layer architecture seen earlier in the lecture.

These tiered architectures can use more or less than 3 layers depending on their setup. In addition, the divide between user and server is not set in stone. There exists a spectrum of choices shown below with the dotted line representing the client server divide:

2.2.4 Decentralized Architectures

Decentralized architectures are also known as Peer to Peer (P2P) systems. Unlike the client-server architecture, each node (peer) can be a client, server, or both with all nodes being mostly equal. They can also come as structured or unstructured systems. Peer can also come and go at any time unlike server that has to be there all the time.

2.2.4.1 Chord Architecture (Structured)

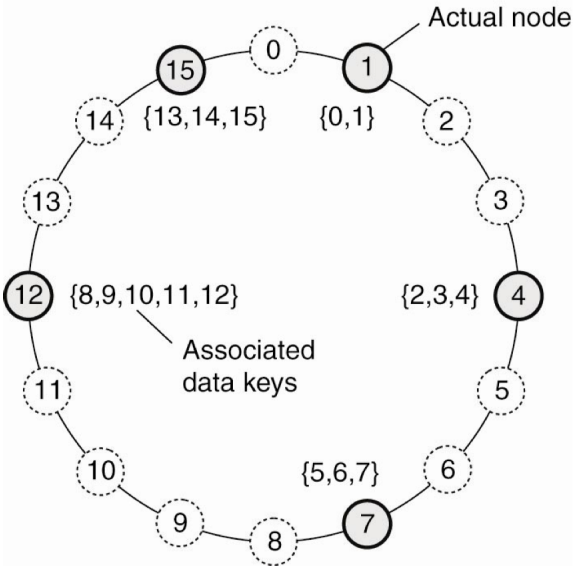

Figure 2.6: Chord Structure

This chord looks like a logical ring topology but the Internet is not a ring so it is an overlay topology. The high level method used in these types of architectures is a key to value hash. Here, a search string is hashed into an int (key), and if there is a node in the system that matches the key then go to that node and download the file.

The chord structure manages leaving by simply assigning the leaving node's keys to the next node above it. If n_7 were to leave, n_{12} would then be responsible for [12,11,10,9,8,7,6,5]. If a node joins, it chooses an unfilled position from 1 to n and takes a section of the files from the higher node. In our current case, if n_8 joins after n_7 left, n_8 would be responsible for [8,7,6,5]. As one can see, joins and leaves are symmetric. Replication or redundancy is used so that when node leaves, the system still works.

Lecturer mentioned about the paper in the class which has more detail about Chord.

https : *//pdos.csail.mit.edu/papers/chord* : *sigcomm*01/*chord*$_s$*igcomm.pdf*

2.2.4.2 Content Addressable Network - CAN (Structured)

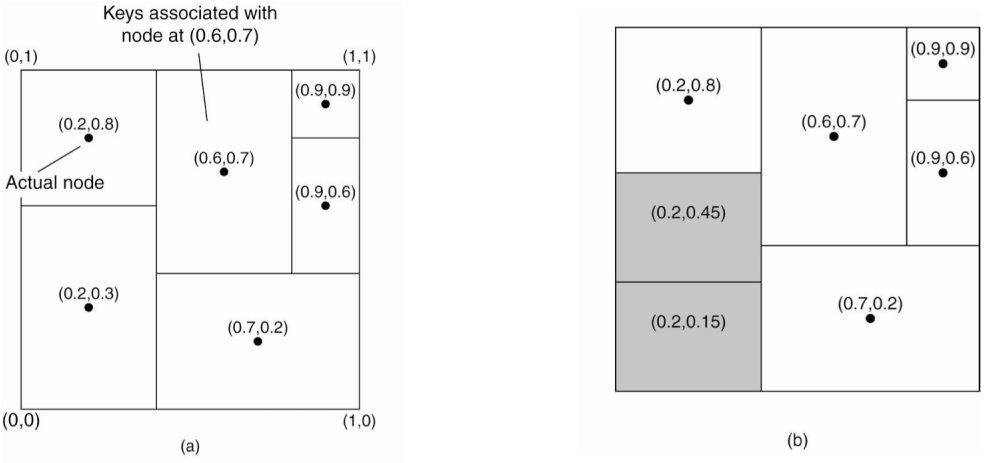

Figure 2.7: CAN Structure, with (b) showing a join procedure

As opposed to chords, CANs use a d-dimensional coordinate system. To make illustrations easier, we will set $d = 2$ for the rest of this section. Each piece of content in a CAN has 2 identities: $< id.x, id.y >$ or <filename, filetype>. For examples, filename, "Foo" may have a different filetype such as .jpg and .txt. This means that each node is responsible for a rectangular partition of the coordinate space, as seen in Figure 2.6. The user can have a more fine-grained query in this structure.

Joins and leaves are thus more difficult, as the merging of 2 rectangles is not always a rectangle. If a node leaves, the system must partition that rectangle to merge it with other already present rectangles. However, joining involves simply choosing a location (x and y) at random for the node and splitting that rectangle with the new node.

2.2.4.3 Unstructured P2P Systems

Rather than adhering to some topological protocol such as ring or tree, unstructured topology is defined by randomized algorithms. Each node picks a random set of nodes, the number of node is based on choice of degree such as k = 2 means the new node will randomly link to 2 existed nodes, and establishes logical connections. When a node leaves, the connections are severed and any remaining nodes can establish new links to offset the lost connections.

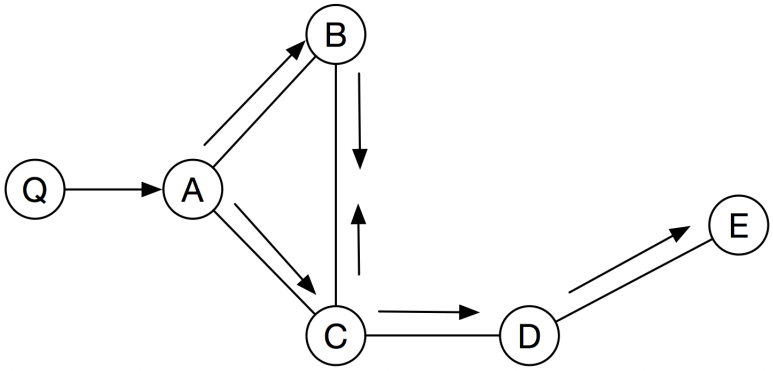

Figure 2.8: Search in Unstructured P2P System

Search Search is done by propagating through the graph as seen in Figure 2.7. Here, a query (Q) is passed to node A, which is then propagated through the network as each node queries its neighbors. Eventually, the signal is back propagated to the sender. This can easily flood the system, so one can create a hop count limit to reduce unnecessary traffic. Each time the query is passed to a neighbor, the hop count is decremented. Upon reaching 0, the node will simply return not found.

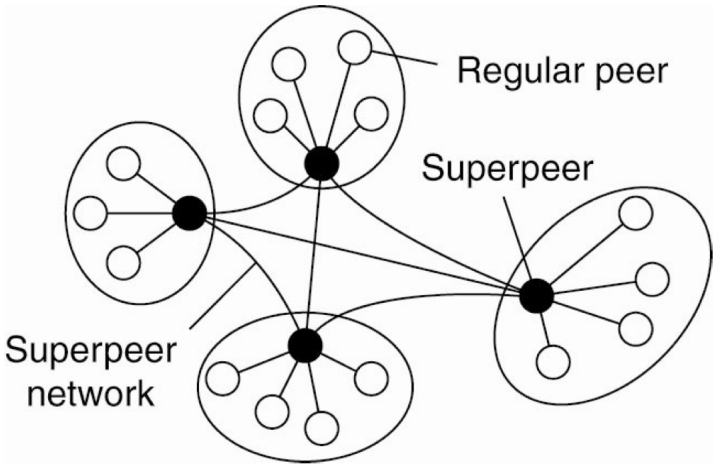

Figure 2.9: Graph with SuperPeer Structure

SuperPeers A small modification to the completely unstructured P2P system allows for much more efficient communication. The P2P graph is partitioned into clusters, where one peer within each cluster can communicate with other peers outside of the cluster. These super-peers are dynamically elected within each group, and should have additional resources to facilitate the increased communication demand.

The restricted communication reduces unneeded calls to neighbors and prevents the huge amount of broadcast traffic found in the completely unstructured P2P system. The downside is that there is a lot of traffic going through super-peer, even though super-peer does nothing at all.

In the past, Skype was a good example of how SuperPeers work. There is more information about how Skype used to work in the book on page 88-90.