CMPSCI 677 Distributed and Operating Systems **Spring 2023**

Lecture 3: February 13

Professor: Prashant Shenoy *Scribe: Shreya Dubey (2023), Rajul Jain (2022)*

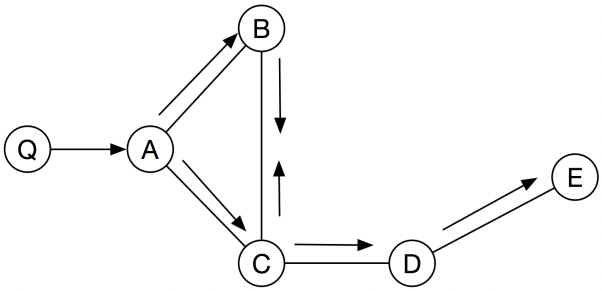
3.1 Decentralized Architectures (Module 3) Contd

3.1.1 Unstructured P2P Systems

Rather than adhering to some topological protocol such as a ring or a tree, unstructured topologies are defined by randomized algorithms, i.e., the network topology grows organically and arbitrarily. Each node picks a random ID and then picks a random set of nodes to be neighbors with. The number of nodes is based on the choice of degree. If $k = 2$, it means the new node will randomly link to 2 existed nodes and establish logical connections. When a node leaves, the connections are severed and any remaining nodes can establish new links to offset the lost connections.

Without structure, certain systems can become more complicated. For example, a hash table key lookup may require a brute force search. This floods the network, and the response also has to go back the way it came through the network. We observe that the choice of degree impacts network dynamics (overhead of broadcast, etc.). The unstructured notion of such P2P systems framed early systems, but newer systems have more structure in order to reduce overhead.

Figure 3.1: Search in Unstructured P2P System



From the figure above, we see that search in an unstructured P2P system is done by propagating through the graph as seen in the above figure. 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 backpropagated to the sender. This can easily flood the system as mentioned above, 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.

Question (Student): Is there a way to decide which node to inject the query to?

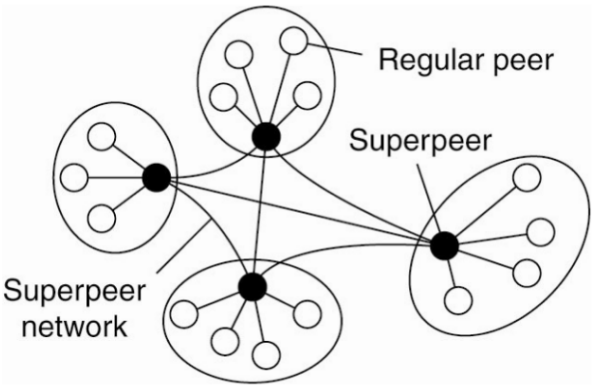
Answer (Instructor): The assumption is that the users are making these queries. So the user will run a copy of the peer-to-peer application on thir machine which is a node in the network. So, the user injects the query at your local peer and then it propagates to the rest of the network.

Question (Student): Can this query search create a cycle?

Answer (Instructor): Network topology can be arbitrary so the query might propagate and come back to the same node. But if nodes keep track of the queries already seen, the cycle can be avoided. Keeping track of outstanding queries solves the problem because nodes wait for the responses to come back in the reverse direction.

3.1.2 SuperPeers

Figure 3.2: Graph with SuperPeer Structure



A small modification to the completely unstructured P2P system allows for much more efficient communication and reduces overhead. The P2P graph is partitioned into clusters, where one peer, designated to be the superpeer, within each cluster can communicate with other peers outside of the cluster. These superpeers 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 number of messages should be lower. However, there may still be a lot of traffic still flooding the network albeit only going through superpeers.

An early versions of Skype was a good example of how superpeers work. It tracked where users were and if they were logged in from a specific cluster. It was a P2P system, but Skype has now moved to a client-server architecture instead.

Question (Student): What are some more examples of superpeers?

Answer (Instructor): BitTorrent and P2P backup systems. However, whenever an application is very important, they may not use P2P since P2P assumes that people are donating resources to make the system work.

Question (Student): Are node link connections static or dynamic?

Answer (Instructor): We can't assume that neighbors will stay up. The topology is constantly changing so we must assume dynamic connections and that links with new neighbors will be made.

Question (Student): What happens if the query doesn't reach the node that has the content because the number of hops a query can traverse is set to a small number?

Answer (Instructor): It's possible the content won't be found since the query distance is not sufficient. But generally, in a peer-to-peer network where nodes come and go, multiple copies of the content are present

in the network. Thus by random chance, with a reasonable threshold, one of the nodes having the content can be reached. Hence, its probabilistic with no guarantees.

Question (Student): Is there an advantage to flooding versus having a list of nodes and then asking these nodes in some order?

Answer (Instructor): Since it is a peer-to-peer network, it's a decentralized network. There is no central entity that keeps track of all the nodes.

Question (Student): What happens if the superpeer is down?

Answer (Instructor): A new super peer is chosen using leader election.

Question (Student): Can you have another layer on top of super peers where they elect another set of peers that are even more distinguished than the super peers?

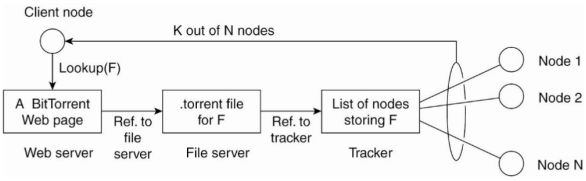
Answer (Instructor): This architecture is also possible. But this can result in a degree of centralization that should be avoided in a decentralized system.

Question (Student): If a new super peer gets elected, how do other nodes know about it?

Answer (Instructor): There should be a protocol by which the super peer advertises its election as a super peer and connects to other super peers.

3.1.3 Collaborative Distributed Systems

Figure 3.3: BitTorrent, an example of a collaborative distributed system



In a collaborative distributed system, files are split into chunks and spread across peers. A client can request these chunks and piece them together. This system allows parallel file download sources from multiple connections, which is faster than a sequential file download from a singular connection. A node can control how parallel it wishes to be, i.e., how many nodes or peers it connects to.

A system like BitTorrent can also take into account an altruism ratio, and slow down a node based on the ratio. If a node is just downloading without also uploading chunks in its possession, or more generally, provide services to other peers, then the system may reduce the download speed of the node. This incentivizes nodes to participate in and contribute to the network instead of freeloading so that they can get good performance.

Two key components are involved in a torrent system: the tracker and the torrent file. The tracker is an index that monitors which nodes have which chunks. The torrent file points to the tracker and can be posted on a web server. In short, the torrent file gets a client node to the tracker which shows which peers it needs, and then the client node can directly connect to those peers based on the configurable setting of how many peers it wants to connect to at one time.

Question (Student): Does the tracker get updated?

Answer (Instructor): As long as a user is connected to it, the tracker knows who has what content.

Question (Student): How do nodes agree on how a file is split?

Answer (Instructor): The file is split how you want. This is a configurable parameter in the system.

Question (Student): Are chunks of larger size stored on nodes with higher bandwidth?

Answer (Instructor): Chunks are of fixed size. But more download can happen from nodes that give better service.

Question (Student): In BitTorrent, can you download and play at the same time?

Answer (Instructor): BitTorrent can be combined with other applications like media player which is downloading and playing. It's better to download enough of the chunk before playing because if the download slows down, the playback will stall.

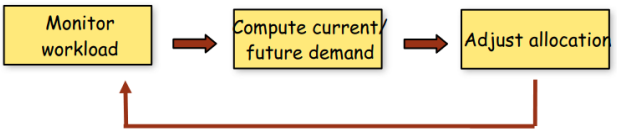
Question (Student): What does the tracker and Torrent do?

Answer (Instructor): Torrent stores information about the trackers. Trackers keep track of the nodes which have the content. Torrent is a file and we can't statically store information about all nodes that have the chunks because that information can change overtime.

3.1.4 Autonomic Distributed Systems

An autonomic distributed system can monitor itself and take action autonomously when needed. Such systems can perform actions based on the system performance metrics, system health metrics, etc. We will not dive too deep into this topic, but knowing the concept helps.

Figure 3.4: Automatic capacity provisioning



This is an illustration of how you might implement such a system. You can monitor the current workload, predict future demand, and if the system thinks the current resource is not enough, deploy more nodes. If the system thinks the resource is not enough, then the number of nodes could be reduced. This technique is also called elastic scaling. The workload prediction part can involve many techniques. For example, we could use feedback and control theory to design a self-managing controller. Machine learning techniques such as reinforcement learning can also be used.

3.2 Message-Oriented Communication

3.2.1 Communication between processes

Suppose two processes running on same machine want to communicate with eachother. There are 2 ways to achieve this.

Unstructured Communication:

- Processes use a shared memory/buffer or a shared data structure for communication.

- This type of communication is called "unstructured" because the buffer is just a piece of memory and has no structure associated with it. In other words, we can put any data that we want in the buffer and the processes have to interpret that data and do something with it.

Structured Communication:

- Processes send explicit messages to communicate, i.e., using Inter-Process Communication (IPC).
- This can be achieved by either low-level socket-based message passing or higher-level remote procedure calls.

Now suppose the two processes are running on different machines in a distributed system. If the processes communicate via unstructured communication, then we need to figure out a way to share buffers across machines. Clearly, making memory accessible across machines connected over a network is more difficult than using shared memory in the same machine. Hence, we prefer structured communication like sockets and RPCs.

Question (Student): Why do distributed systems need low-level communication support for both structured and unstructured communication?

Answer (Instructor): Because distributed systems are spread over a network.

3.2.2 Communication (Network) Protocols

Suppose two processes (App 1 and App 2), each using the OSI or TCP/IP model, communicate over a network. App 1 creates a message at the application layer which travels down the network stack layers (where each layer adds their headers to the original message) all the way to the physical layer. The message is then passed over to App 2's physical layer where it travels all the way up the network stack again to complete the communication.

Question (Student): Can the ordering of the layers change?

Answer (Instructor): The ordering will not change.

Question: (Student): Can the encryption happen at a higher level?

Answer: (Instructor): Only the message can be encrypted, not the other aspects. If enrcypting after Data link layer, nobody can be see what is there in it. However, hardware can be used to encrypt the message itself.

3.2.3 Middleware Protocols

In a distributed system, *middleware* is the layer residing between the OS and an application. The middleware layer sits between the application and transport layer in the network protocol stack.

3.2.4 TCP-based Socket Communication

TCP-based socket communication is a structured communication method which uses TCP/IP protocols to send messages. The socket creates network address (IP address and port number) for communication. The socket interface creates network end-points. For example, in a client-server distributed system, both the

client and the server need to create sockets which are bound to port number where they can listen to and send messages.

Question (Student) : What is the difference between a socket and a port?

Answer: (Instructor): A port number is the identifier of a socket and a socket is the abstraction.

Question (Student) : How many users can connect to a port?

Answer: (Instructor): Any number of users can connect to a port.

Question (Student) : Every time a client connects, is there a socket connection?

Answer: (Instructor): Yes, every client has its own TCP/IP connection to the server.

Question (Student) : What happens if the same client has multiple requests?

Answer: (Instructor): Depends on how the client is implemented. It can be a sequential client if it's in a while loop sending one request at a time or if the client has multiple threads, they can open multiple TCP connections to the server and send requests in parallel on each of these connections.

Question (Student) : How many processes can listen on the same port number?

Answer: (Instructor): There can be only on socket that is associated with a socket of a certain port number. For example, you can have multiple web servers but these should listen on different port numbers.

3.2.4.1 Understanding TCP Network Overheads

Consider a client-server model using TCP method for communication. As TCP method follows a 3-way handshake protocol for communication, this leads to a transfer of 9 messages for sending a single request and subsequently receiving a single response. This is a huge overhead.

Question (Student) : What happens when there are multiple client requests?

Answer: (Instructor): Typically, it depends on the protocol. In sequential processing, the client sends a request and then waits for an answer. There can be multiple requests with the same connection. You can send a request before the response to the first request is received only in some protocols. This requires tracking which question this response corresponds to.

Question (Student) : What happens if the client doesn't receive the response?

Answer: (Instructor): All the messages have a timeout. If the acknowledgment is not received by a certain time, the request will timeout and the client then has to send it again.

Question (Student) : Is a separate acknowledgement required for each message?

Answer: (Instructor): Packets are sent in sequence, so if acknowledgement for a message is sent, then it means all the messages before it were received, so you can send just one acknowledgement for multiple messages.

Question (Student) : Why does the server send the SYN meeage?

Answer: (Instructor): SYN message is used to establish one-way connection. Throught he first SYN, the client establishes a one-way connection to the server. Only client can send messages to the server. In order to send messages to the client, the server needs to send a SYN message to establish one-way connection to the client. This will then form a duplex connection.

Question (Student) : What happens if some messages in the middle get lost in a sequence of messages?

Answer: (Instructor): Acknowledgement of the first few messages will be sent. If the acknowledgement is not received after the previous ACK, it is assumed that the message after the previous message was lost so the client sends it again.

Question (Student) : When will the server refuse to connect with a client?

Answer: (Instructor): As long as the socket on which it is listening is open, it can continue to receive new connections. The only way to stop is to close the conection.

3.2.5 Group Communication

In a distributed system, when one machine needs to communicate with many machines, group communication protocols are used. This is analogous to sending an email to multiple recipients. For group communication, all the recipients subscribe to a *group address*. The network then takes care of delivering the messages to all the machines in the group address. This is called *multicasting*. If the messages are sent to all the machines in the network, then the process is called *broadcasting*.

3.3 Remote Procedure Calls

RPCs provide higher level abstractions by making distributed computing look like centralized computing. They automatically determine how to pass messages at the socket level without requiring the programmer to directly implement the socket code. In other words, instead of sending message to the server to invoke a method X, the programmer can call the method X directly from the client machine. RPCs are built using sockets underneath. The programmers do not write the this socket code. Instead, it is auto-generated by the RPC compiler. Stubs are another piece of RPC compiler auto-generated code which convert parameters passed between client and server during RPC calls.

There are a few semantics to keep in mind:

- Calling a method using RPC is same as invoking a local procedure call.
- One difference is that in an RPC, the process has to wait for the network communication to return before it can continue its execution, i.e., RPCs are *synchronous*.
- *You cannot pass pointers or references.* Pointers and references point to a memory location in the respective machine. If these memory locations are passed on a different machine, they will point to something completely different on that machine.
- *You cannot pass global variables.* As global variables lying on one machine cannot be accessed by another machine simply over a network. Hence global variables are not allowed in an RPC.
- *Pass arguments by value.*

Question: (Student) : If we absolutely need to pass pointers in an RPC, how do we achieve that?

Answer: (Instructor): Take the entire object (say, from the client machine) and pass it on to the server. Create a copy of this object on server and then create a local pointer to the object.

Question (Student) : Does the connection stay open from the client side?

Answer: (Instructor): Depends on the stub code. It can decide to setup a new TCP/IP connection for every RPC or send multiple RPCs on the same connection.

Question (Student) : When we write a message from the client, do we need to know the format of the message?

Answer: (Instructor): You don't write a message but invoke a function. Thus, only the function call needs to be constructed with the corect parameters which is then sent as a function call. The rest is taken care of by the stub.

3.3.1 Marshalling and Unmarshalling

Different machines use different representation of data formats. This creates discrepancy in understanding messages and data between different machines. Hence before sending messages to the other machine, the messages are converted into a standard representation such as eXternal Data Representation (XDR). Marshalling is the process of converting data on one machine into a standard representation suitable for storage/transmission. Unmarshalling is the process of converting from a standard representation to an internal data structure understandable by the respective machine.

Question (Student) : What is the advantage of using RPC over HTTP?

Answer: (Instructor): HTTP is an application-level protocol used for sending and getting replies. But RPCs allow any two applications to communicate with each other.

3.3.2 Binding

The client locates the server using bindings. The server that provides the RPC service registers with a naming/directory service and provides all of the details such as method names, version number, unique identifier, etc. When client needs to access a specific method/functionality, it will search in the naming/directory service if there's a service in the network which hosts this method or not and then accesses the server using the IP and port number listed in the directory.

3.4 RPC Implementation

3.4.1 Failure semantics

- Lost request/ response messages: The network protocols handle these errors by timeout mechanisms.
- Server and client failures: Need to be handled while creating distributed systems.

If client crashes after sending a request to the server, then the server response is referred to as **orphan**. To deal with the orphans, methods such as Extermination, Reincarnation, Gentle reincarnation, and expiration can be used.

3.4.2 Case study: SUNRPC

SUNRPC was developed for use with NFS. It is one of the widely used RPC systems. Initially, it was built on top of UDP, but later transferred to TCP. It uses SUN's XDR format.