Distributed Systems

A distributed system is a collection of independent computers that appears to its users as a single coherent system [Tannenabum and Wetherall, 2011]. Although a distributed system shares many attributes with computer networks, there are some key differences that are worth mentioning. A computer network is group of individual computers connected, often by a single technology. The distributed system is to a network what an operating system is to a single system. It manages resources and can share them across the network. A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system [Tanenbaum and van Steen, 2017].

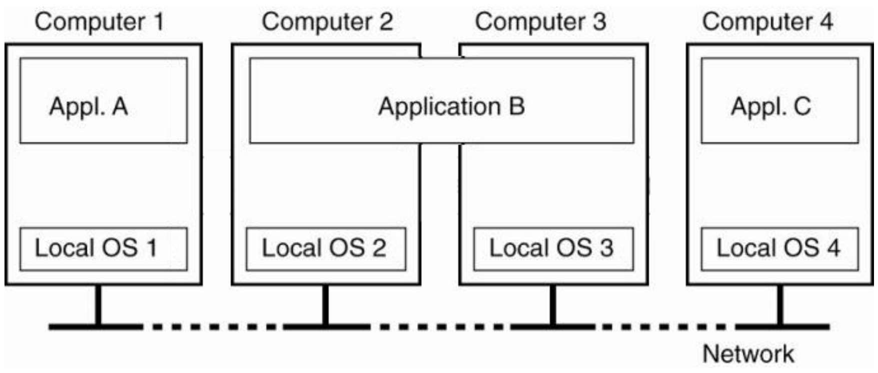

Figure 1 - A Computer Network

Users often consider a single system to entail a single computing device in a fixed location or mobile. Furthermore, computing tasks can be distributed within that single system through concurrency or parallelization. A distributed system is able to appear as a single system while comprising of multiple devices that are often geographically dispersed. Furthermore, computing tasks can be distributed across the distributed system. Distributed Systems are able to appear as a single system to its users by means of a layer of software placed over the operating systems of the computers which are part of that distributed system. This software is called middleware.

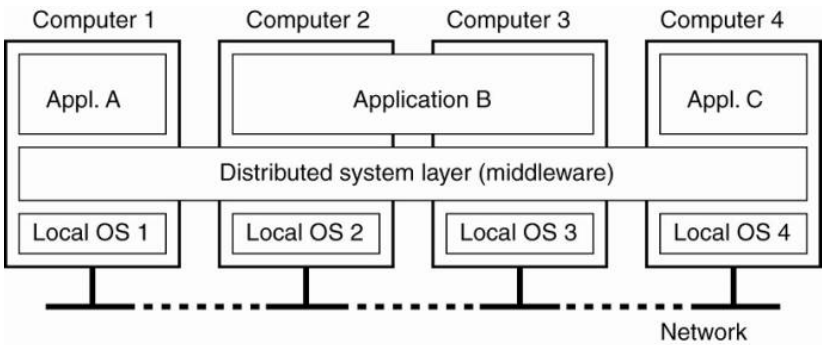

Figure 2 - A Distributed System

In the late 1960's, ARPANET initially only featured a few applications, and by the early 1970's, had an application being used for email. This email application is said to have been one of the earliest examples of a distributed application. Just over a decade later, newsgroups and Bulletin Board Systems (BBSes) would become widely used distributed discussion systems, providing a way for users to share news, ideas, and opinions. In the 1990's, the internet became popularized as a distributed system. If the internet was the network, the world wide web's web servers could be considered the middleware, allowing users to view information located on geographically dispersed severs through a single browser.

Distributed systems are often built for several reasons. One of these reasons is that stored data, processing, or users are in different physical locations. This is often the case with distributed databases that are available on multiple servers. Distributed databases are often found within organizations. Sensor Networks are another example. Sensor networks are made up of small geographically dispersed nodes that have one or more sensing devices. Such networks are used in environmental or system monitoring.

Another reason for building a distributed system is to have a system where computing devices work to achieve a common goal. Networked systems can be configured to provide more processing capability than a single system. An example of such a configuration is cluster computing. With cluster computing, multiple systems with identical hardware use the same network are used for parallel high-performance computing. Grid computing is another example of networked systems being used to provide additional processing capability. With grid computing, computers from different administrative domains work on parts of a larger task.

Distributed systems are also designed for system scalability and fault tolerance. Scalability is important for distributed systems because if work to be completed by the system increases, additional nodes can be added to the system to further disperse the workload. Fault tolerance is of equal importance for distributed systems because if one node fails, the system will still be available.

Types of distributed Systems

Distributed systems come in many forms. As the cost of computing decreased, and the performance of computing systems increased, the cluster computing configuration became a viable option for users who wanted high-performance computing capability. With cluster computing, multiple, often identical systems are connected as a network. These systems make use of parallel programming, where a single application is run in parallel on multiple machines.

Grid Computing describes computing devices from various administrative domains being networked. Each node is assigned a different type of task. A proposed architecture for grid computing systems consists of an application layer, a collective layer, a resource layer, a connectivity layer, and finally a fabric layer. The application layer is made up of the actual applications that make use of the grid computing configuration. The collective layer has services for scheduling tasks onto multiple resources. Both the connectivity and resource functions are at the same level above the fabric layer - the connectivity layer provides protocols for data transfer between resources. The Resource layer offers functions for getting another resource's configuration. Lastly, the Fabric layer is the lowest layer, which provides an interface and functions for resource management [Tanenbaum and van Steen, 2017].

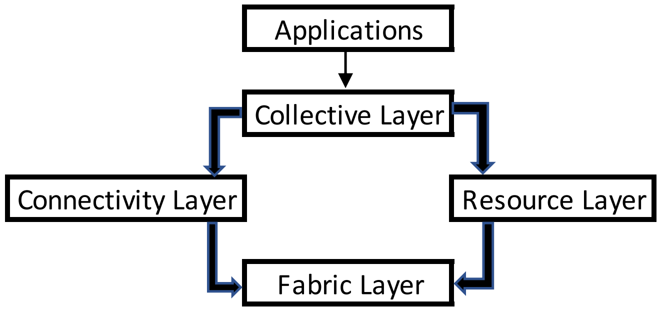

Figure 3 - A layered architecture for grid computing

Network applications such as distributed databases usually comprise of databases that are not all on the same system. Such databases can be dispersed over a network, the internet, or corporate intranets. These applications can be accessed by remote clients

Wireless Sensor networks contain small nodes which each have one or more sensing devices. These sensing devices are often low-power, battery operated devices that are wirelessly connected either to one another or centralized database. When connected to a database, the sensors can send their data to the centralized database, which aggregates the data for processing. Another option is for the operator of the database to query the appropriate nodes, which send their data to be processed.

Wireless sensor networks can be used in environmental monitoring, and control systems. Because the devices are numerous, small, and battery-powered, these networks are often designed for high efficiency. Since 802.11 tends to require high energy use, there have been various wireless connectivity protocols developed specifically for sensor networks, namely ZigBee and Z-wave. Routing protocols and operating systems have also been created for these low power networks. [Wikipedia, 2019].

System and Software Architecture

The goal of a distributed system can range from making data in geographically dispersed systems available to an individual user, to efficiently solving compute intensive problems. Because of the complexity and variety of goals for distributed systems, careful consideration is given to the architecture used for the software and systems.

One of the components that distinguishes a distributed system from a computer network is the layer of software that exists above the operating system of a networked computer. This layer of software is called middleware. This software uses several design patterns to allow communication between devices that may have different configurations, operating systems, or hardware. These design patterns are wrappers or interceptors.

A wrapper provides an interface to a client application. If applications on a distributed system have incompatible interfaces, a wrapper is created to enable compatibility between applications. Wrappers are created to allow the addition of nodes and applications to existing distributed systems. One of the major challenges with wrappers is that if one is created to enable communication between each application, having N applications will require N x (N-1) wrappers. This can become unwieldly; however, a broker can be created that exists between all applications and enables communication between any and all pairs of applications.

An interceptor is a piece of software that allows distributed systems to be extensible. A local interface is provided to an object that is identical to interfaces provided by a remote object. A call to the local interface is turned into a generic object invocation via middleware. This object invocation is sent through the transport level network interface as a message. At this point, a message level interceptor assists in sending the transferred invocation to the target object.

The client-server architecture is used to separate processes that implement a service from processes that request services. There is often not a clear distinction between client and server as processes may forward requests that are sent to them [Tanenbaum and van Steen, 2017]. The client and server use request-reply interactions. Connectionless protocols can be efficient for connecting clients and servers, allowing lost messages to be resent. Connection-oriented protocols are useful and reliable over Wide Area Networks.

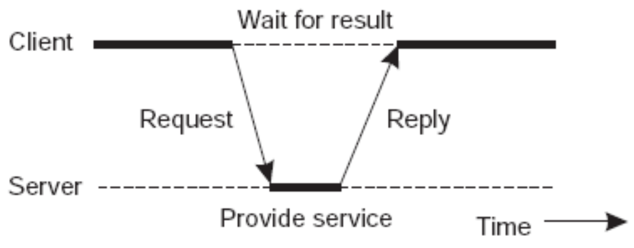

Figure 4 - Client-Server architecture

When physically implementing the client-server architecture, the two-tiered architecture can be used to designate some machines as clients and others as servers. The client machine will contain the user interface. The server machine will contain the application and the database. There are variations of this model that place some parts of the application and or database at the client level. Conversely, there are other variations that place parts of the user interface at the server level. Irrespective of what variation is used, the two-tiered architecture is physically implemented by designating these two types of machines.

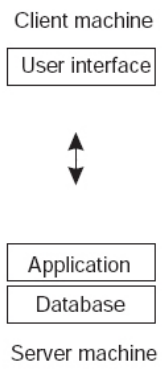

Figure 5 - Client-server in a two-tiered architecture

The three-tiered architecture is used when the system must be physically implemented using a client, a server, and a database. With the three-tiered architecture, a client requests an operation from an application server. The application server must then request data from the database server. The database server then returns the requested data to the application server, which provides a reply to the client. In the three-tiered architecture, the server also becomes a client, as it has to make requests to the database server.

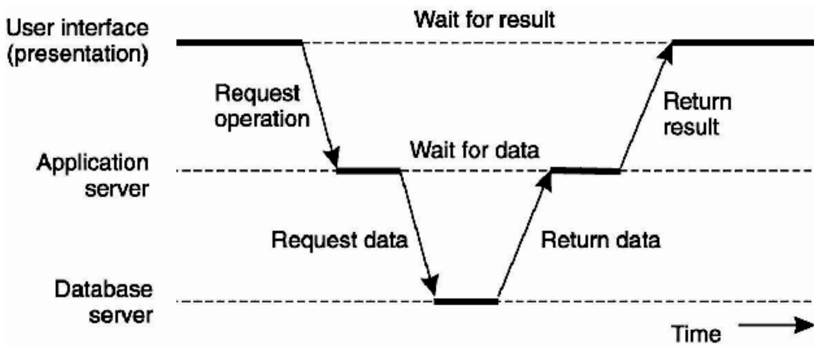

Figure 6 - Client - server three-tiered architecture

Another system architecture often used for distributed systems is the peer-to-peer architecture. This architecture distributes the load of the data by physically splitting the clients and servers. This architecture also allows all processes to be both client and server, meaning each process can request, or receive requests. Data that is stored within a peer-to-peer network has a key associated with it. These keys are then mapped to nodes which may contain the data. A user can find data within a distributed system by using the key to find the appropriate nodes. Peer-to-peer networks take on various forms ranging from structured, unstructured, to hierarchically organized.

A structured peer to peer network is one in which nodes have a specific network structure. A chord is an example of a structured peer to peer system. In a chord structure, each node maintains a list of keys for nodes that it is connected to. If a key is being searched for within the network, the node that receives the request will either forward the request to the either the farthest node that it has keys fir, or a to node that precedes the requested node key.

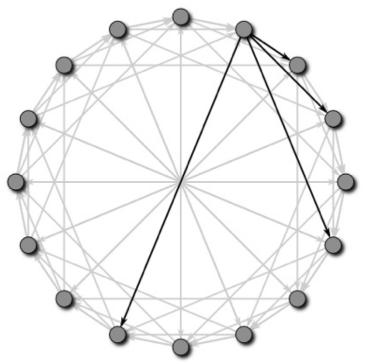

Figure 7 - Structured peer-to-peer network; Chord

An unstructured peer to peer network is one which doesn't have a defined network structure. This undefined network structure allows for new nodes to be added to the network. Each node only maintains a list of its neighbors. The unstructured peer-to-peer network unfortunately cannot be searched as easily as the structured network. Requests for a particular node can be completed by means of random walks or flooding. Random walks require that a node forward a request to one of its randomly selected neighbors. Flooding on the other hand, entails each node passing the request to each of its neighbors, essentially sending the request through the network until the searched node is found.

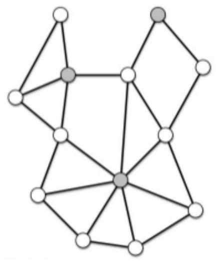

Figure 8 - Unstructured peer-to-peer network; Random graph

Flooding and Random walk searching can become inefficient as an unstructured peer-to-peer network get larger. Hierarchically organized peer-to-peer networks provide a way to manage a large number of keys that will need to be searched in an unstructured system. In this type of peer-to-peer system, a node from a section of the network is selected to maintain an index of data or availability. This node is referred to as a broker or super-peer. The super-peers, which are connected to their respective subsets of network nodes, are connected to one another to form what is called a super-peer network. When a new node is added to the network, it is connected directly to a super-peer. This results in more efficient storage or searching of data across a network.

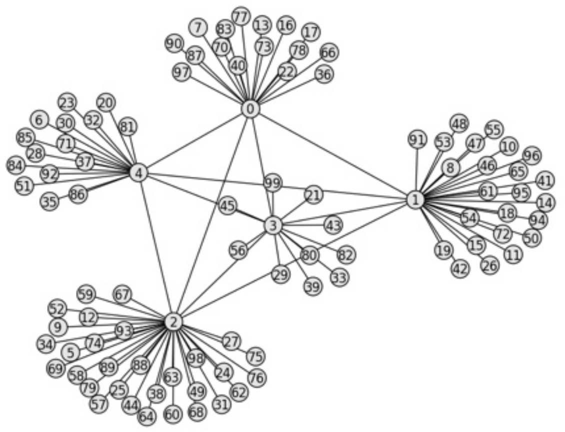

Figure 9 - Hierarchically organized peer-to-peer network

Interprocess Communication

Because a distributed system may have hundreds of thousands of processes occurring over a network, how those processes communicate is critical for the efficiency of the distributed system. The OSI reference model describes network communication using seven layers - the application layer, the presentation layer, the session layer, the transport layer, the network layer, the data link layer, and finally the physical layer.

For the internet, the application layer, which was initially intended for email, FTP, and remote job entry, now contains other applications. The presentation layer was intended to provide senders and receivers a way to identify the type of data being transmitted. The session layer provides a means to determine which end of a connection is sending data, and a means to continue transmissions from a checkpoint. Although this isn't' used for the internet, this layer is an important part of distributed systems. The transport layer allows data transport from a process on a source machine to a process on a destination machine. The Network layer gets packets from the source to destination using hops. The Data link layer allows the delivery of error-free data to and from the Network layer by using frames and acknowledgements. Lastly, the physical layer determines how to standardize signals to send bits across a network.

We described middleware as the layer of software that exists above the operating system of a networked computer. There are many protocols that can be used by different applications but should not be considered transport protocols. Given the various applications that can use the protocols, these protocols could be described as middleware. Distributed Systems proposes a reference model for networked communication has an application layer, a middleware layer, an Operating System layer, and finally a Hardware layer.

This reference model modifies the existing OSI model and adapts it to the distributed system. The application layer remains the top layer. The session and presentation layer contain application-independent protocols represented as the middleware layer. The transport and network layers are grouped into the interprocess communication services of the operating system layer. The hardware layer is used for actual communication establishment.

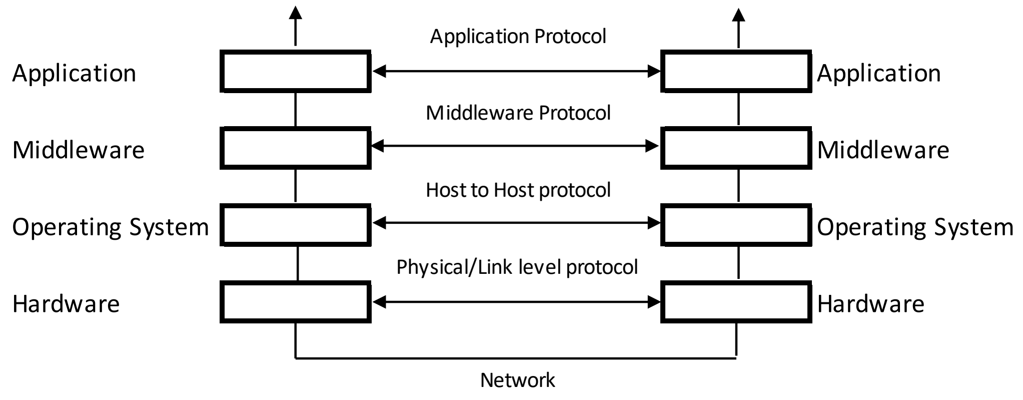

Figure 10 - A reference model for networked communication

Middleware Protocols

Patterns in Network Architecture defines an (N)-distributed-IPC facility as a distributed application consisting of at least one IPC application in each participating processing system. The (N)-DIF provides IPC services to applications via a set of (N)-API primitives that are used to exchange information with the application's peer. The corresponding application processes may be in other processing systems [Day, 2008]. A remote procedure call could be considered an (N)-distributed-IPC facility as its function is to call a procedure on another machine. Client stubs transform local procedure calls into network requests. Server stubs transform requests from the network into calls.

With Remote Procedure Calls, there is no way to be certain that a receiver will get a sent message. Messaging is synchronous, so a receiver may not be present when interprocess communication is attempted. One solution is to provide a means for messages to be delivered, even if the receiver is not present at the time of sending. Message Oriented Middleware, also referred to as message queueing systems, allows applications to send messages to one another via queues. A sent message can be stored in the queue if the receiver is not active. The stored message can be later retrieved if when the receiver becomes active, even if the sender is no longer active.

High performance multi-computers used for parallel computing would often have proprietary communications protocols which were incompatible with one another. This created a problem when moving platforms across different hardware. The MPI standard was created to allow communication compatibility between high performance multi-computers. It also provided developers with a means to create more efficient parallel computing applications that supported synchronous and asynchronous communication using a variety of primitives.

Operation	Description
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Figure 11 - Message Passing Interface primitives

Lastly, a class of protocols known as multicast communication protocols allow the sending of messages to all nodes in an overlay network. Multicasting requires that there is a unique path between every pair of nodes.

Process Synchronization and Coordination

Given that distributed systems often have processes occurring over various parts of the underlying network, each process' time should be synchronized with that of dependent processes. Furthermore, coordination is necessary to ensure that processes that are dependent on other processes are executed correctly. Universal Coordinated Time is the standard for current time. Servers with a connection to a Universal Coordinated Time signal are able to be contacted by other servers to retrieve an accurate time.

Clock Synchronization can occur through Network Time Protocol. If a server needs the time, it can contact a time server that receives a UCT signal and use timestamps to estimate delay or relative offset. This process is repeated so that eight pairs of offset and delay are calculated. The minimum value is the best estimation of delay and offset between the servers. The server adjusts its clock.

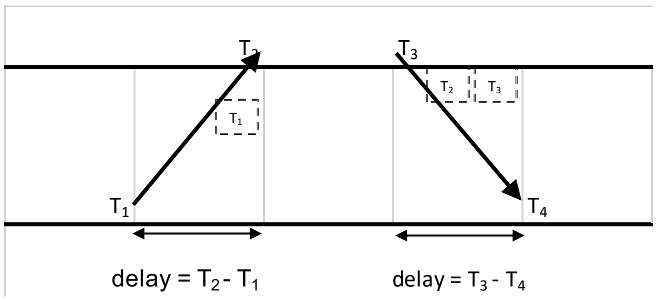

Figure 12 - Clock synchronization between server and UCT server

Offset, $\theta = \frac{(T2-T1)+(T3-T4)}{2}$ Delay, $\delta = \frac{(T4-T1)-(T3-T2)}{2}$

With the Berkeley Algorithm, a server polls other machines for their times and determines the average time. Based on the average, it provides an offset that they apply to their clocks. The machines adjust their clocks to this calculated average.

Reference broadcast Synchronization allows two nodes, p and q, to use linear regression of their delivery times to compute clock offset. Offset[p,q](t) = $\alpha t + \beta$, where α and β are computed from pairs of times from nodes p and q

For many processes, the accuracy of the system clock is not as important as the accuracy of the process' logical clocks. The logical clock provides the order in which events occur. Lamport's Logical Clocks is a relation specifying that all processes agree on the order in which events occur, then adjust their clocks if necessary. Similarly, vector clocks are more concerned with the order of events instead of the actual time that processes executed. Each process maintains a vector for the number of events that have occurred before it, enabling clocks that run at different rates to self-correct.

Mutual Exclusion allows one node or process to access a resource at a given time. Token-based algorithms allow for such concurrent sharing. A token-based algorithm allows processes to access a resource in a round-robin fashion. Another method for ensuring mutual exclusion is to designate a coordinator process which provides access to other processes that request its use. If the resource is currently being used, the requesting process is placed in a queue until the resource becomes available. Lastly, multiple coordinators can be provided for several replicas of a process. Each coordinator will vote on whether it not it can provide access for a given resource.

So how exactly are the coordinator processes chosen? Election Algorithms are used to determine which process will be the coordinator. An example of an election algorithm is the bully algorithm. If a process notices that the coordinator is unresponsive, it will send an election message to other higher-numbered processes. All responders to the election message must send an election request to all remaining higher numbered processes. The Bully Algorithm is named because higher numbered process eventually ends up as coordinator.

Super peer networks were defined earlier as a large, unstructured peer-to-peer networks where a node had to be designated as the broker for a subset of nodes in the network. This role of broker or index-keeper is synonymous with that of a coordinator. The process for selecting coordinators in peer-to-peer networks is done by distributing tokens to randomly chosen nodes. These randomly chosen nodes will be super-peers. These super-peers need to be cover the various parts of the distributed system. The tokens assigned to these nodes are assigned opposing values relative to one another. The tokens are passed through the network so that they end up with nodes where the opposing values of tokens no longer act upon one another. The final holder of the tokens becomes the super-peer coordinators of the network.

Distributed Event Matching assumes that processes should enable coordination by sending out notifications related to events. Furthermore, it allows interested processes to specify events that they wish to receive notifications for. The processes sending notifications are considered publishers. Those processes that have specified notifications that they wish to receive are subscribers.

Distributed computing presents new use cases and formats. Although we have primarily focused on parallel computing, grid computing, the world wide web, and wireless sensor networks, many of today's burgeoning technologies utilize principles used in distributed systems. In cloud computing, users employ Software as a Service to share applications running on a distributed infrastructure. Users can access an integrated environment in the form of Platforms as a Service, enabling them to deploy parallel applications. Infrastructure provides users the means to share hardware resources such as computing power and storage [Fantacci, Tarchi and Tassi, 2011]. Blockchain uses a distributed peer-to-peer-like architecture, and also employs transaction mechanisms very similar to those used in distributed computing [Herlihy, 2019].

Bibliography

"Distributed Computing", https://en.wikipedia.org/wiki/Distributed_computing, accessed April 2019.

Day, John. *Patterns in Network Architecture: A return to Fundamentals.*Pearson Education, 2008.

Romano Fantacci, Daniele Tarchi and Andrea Tassi (2011). Wireless Communication Protocols for Distributed Computing Environments, Advanced Trends in Wireless Communications, Dr. Mutamed Khatib (Ed.), ISBN: 978-953-307-183-1, InTech, Available from: http://www.intechopen.com/books/advanced-trends-in-wireless-communications/wireless-communication-protocols-for-distributed-computing-environments

Herlihy, Maurice*. "Blockchains from a Distributed Computing Perspective" Communications of the ACM,* Vol. 62, No. 2 (February 2019).

Van Steen, Maarten, and Andrew S. Tanenbaum. *Distributed Systems.* Maarten van Steen, 2017.

Tanenbaum, Andrew S. and David J. Wetherall. *Computer Networks*.Prentice Hall, 2011.