



Faculty of Engineering and Applied Science

SOFE 4790U Computer Networks

Homework #1

| Name | Student # |
|----------------|-----------|
| Owen Musselman | 100657709 |

Note: Changes made after group discussion are highlighted in green.

Objectives 1.1, 1.2, 1.3:

Describe what distributed systems are.

Distributed systems are an environment where the components are located in different areas over a network. Where these machines will split the work and complete the work more efficiently than on centralized systems.

Describe characteristics, requirements, of distributed systems.

Distributed systems are systems that need to be able to run multiple processes in parallel. Additionally, distributed systems need to be scalable and adapt to fluctuating system traffic, be it in an increase in system resources or a decrease in resources. Distributed systems need to be fault tolerant, where the system should be available and reliable. Distributed systems need to be open, where the new components should be able to interact with old components, in essence these systems should be open in terms of software and hardware. Within these systems there is going to be heterogeneity with hardware, and networks. If distributed systems are not reliable and scalable the odds are high that the system will fall apart under higher workloads. The system should employ transparency to mask complexities of the system from users as it is not information they need (security could be compromised if there are no transparencies).

How does this affect a company's hardware and software?

A company's software could affect how the different systems communicate, along with how the services talk with one another. The software affects the level of connectivity between the different numerous components of the system at runtime setup. Whereas the hardware of a company could be affected if the company uses cloud based services to their advantage, or if they have the monetary and engineering means, they could build their own distributed system.

Identify key building blocks/ components and common characteristics.

Building blocks for a distributed system would be its reliability and availability. Without these two components a distributed system would not meet its purpose. Additionally, it should have a high efficiency as to deliver its services on time with little wasted resources. The system should also be easily managed to make troubleshooting, and maintenance easier.

Identify NFR and basic design issues.

The non-functional requirements of a distributed system are as follows:

- Scalability: The system should accommodate fluctuating system traffic.
- Reliability and Availability: The system should be reliable and available at all times.
- Security: The system should be secure from unauthorized access.
- Maintainability: System needs to be easily maintained.
- Usability: The system should be easy to use.
- Performance: The system should have high performance.

Other NFRs are:

- Supportability
- Lifetime Cost
- Flexibility
- Reusability

Basic design issues with distributed systems is that security is difficult as the different components need to communicate with each other securely with proper permissions needing to be assigned to assure the system is secure. The reliability of the system needs to be maintained, but that is difficult as the system has multiple points of failure making it difficult to track down what went wrong and where it went wrong. The availability of the system can be difficult as routine backup images of the system need to be made to save the state of the system in case of a failure so that the system can recover and function again in a timely manner.

Examples of distributed systems.

Some examples of distributed systems are:

- The internet as a whole.
- Telecommunication networks.
- GPS technologies.
- eCommerce technologies (PayPal, Amazon, etc)

Reflect on your previous course on OS. Can you provide a few paragraphs of a summary of what you studied in the course? What are the main functions of an distributed OS? How do you implement them?

In the previous OS course, materials studied included paging methods, which allows the operating system to get processes that are located on the system's secondary storage (hard drive) making it faster to access data. Paging is used to map the virtual address to the physical address of the memory (Memory management unit). It is noteworthy that the page size is the same as the frame size, this is done to ensure that there is no external fragmentation, where the dynamically allocated memory leaves a section of memory unusable. Internal fragmentation is when a process is allocated to memory but that process is smaller than the memory that was initially requested by the process.

Another topic studied was process scheduling which would decide which processes to end and which processes to execute next ensuring that the system functions accordingly. There are various queues that are used: Job ready queue stores all the processes in the system. Ready queue where the processes are ready and waiting to be executed. Device queue is where a process is blocked/halted while waiting for a device. Other schedulers are the short term, long term, and medium term. They are the CPU scheduler, Job scheduler, and the Swapper scheduler respectively. Context switches were also important as they are able to store and restore the CPU in the process control block (PCB), without context switching, multitasking would be a lot more difficult on systems. The ubuntu OS was used along side the C language in order to demonstrate how processes were created, and destroyed. Communication between processes was also covered, along with synchronization of processes using mutex

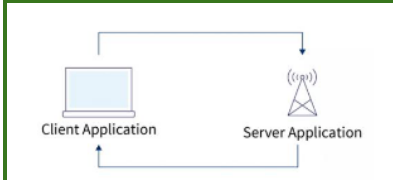
locks, and semaphores. Some of the CPU scheduling techniques covered were FIFO, LIFO, Round Robin, Shortest Job First.

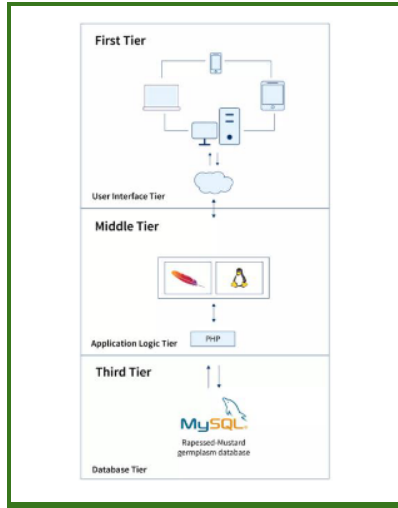
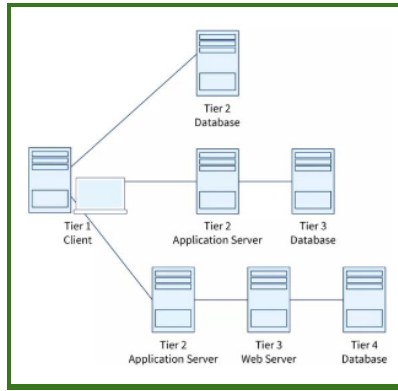
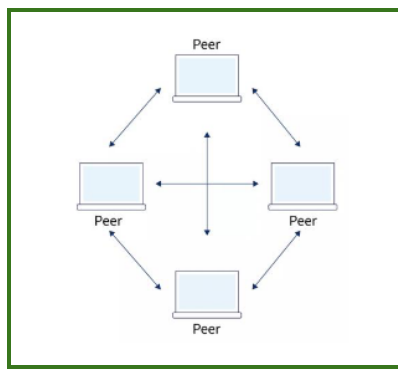
Table below added as the above paragraphs only talked about a couple of topics, the table shows a better representation of what was studied in the OS course.

| Topic | Concept |
|-----------------|---|
| Processes | Creation, Interprocess communication, deletion |
| Threads | Creation, deletion, shared memory |
| Synchronization | MUTEX locks, semaphores, deadlocks |
| CPU Scheduling | FIFO, RR, Shortest remaining time first, Shortest job first |
| File Management | Paging, Page table structure, Virtual memory |

Distributed OS is where applications are run on various machines and are linked together by communication. The communication is done by means of WAN or LAN connections which allows the system to share resources like memory, storage, and computation resources. This will allow more underpowered machines to utilize these resources when needed.

Implementation: this was added as I had did not know whether to talk about the actual hardware, or how to deploy the systems. After the discussion this was added.

| Type | Description | Implementation |
|---------------------|--|--|
| Client-Server Model | Mainly used for resource sharing where the server handles data and resources requested by the client. An example is Netflix. |  <pre> graph LR CA[Client Application] <--> SA[Server Application] </pre> |

| | | |
|--------------------------------|--|---|
| <p>Three - Tier</p> | <p>The three-tier architecture has three layers: presentation, application (business layer), and data. The presentation tier is where the user will access the application to make a request. The application layer is used for business logic. The data tier is the database tier where the application data is stored. An example of a three-tier architecture is a typical business application</p> |  |
| <p>Multi - Tier</p> | <p>Used for when application needs to forward data or requests to multiple networks</p> |  |
| <p>Peer - to - peer</p> | <p>Each system is a node while acting as a client or server. An example of P2P is downloading torrents</p> |  |

Compare the functionalities provided by a centralized file system to that of a distributed file system?

A centralized file system for example has all of a businesses' data stored in a single location. The centralized file system is simple in the sense there are less points of failure. This simplicity comes at a cost where the scale of the system is difficult to change at will. While a

distributed system (DS) allows for data within the system to be shared amongst servers. DS also lets many clients send and retrieve data. As stated before, a DS allows for data redundancy in case of system failures. Another functionality of DS is heterogeneity where the system contains different platforms that can be utilized for different purposes. Additionally, data integrity must be maintained by the system which is done by use of atomic transactions. The scalability of a distributed system is easily changed depending on the current needs of the system. If a distributed system is not connected to a network then the system has now way to access its resources, so it becomes useless, whereas a centralized system can lose its connection to a network and still function, for example a user's PC might not be connected to a network but it is still functional.

What are the components of the distributed system listed in the article? Can you predict other components required that are not mentioned in the article?

HDFS components:

- HDFS Client: How the users will access the HDFS, the client has a library that exports the HDFS interface. Once done the user can read, write, delete, and create directories.
- Block: A memory section where data is held.
- Inode: store file property information like permissions, namespace and disk space quotas.
- Balancer: Is a tool that is used for balancing disk space on a HDFS cluster by moving DataNode(s) replica(s) from places with large utilization to less utilized places.
- DataNode: keep track of data from applications. Is a dedicated server that stores application data.
- NameNode: Contains a map to the DataNodes, along with namespace maps.
- Backup Node: Like a checkpoint node it creates periodic checkpoints but it additionally maintains an up to date, in memory image of the filesystem namespace and will always be synchronized with the current NameNode state.
- Image: made up of inodes, along with lists of blocks which when put together create the metadata of the name system.
- Block scanner: Is executed periodically to scan a block and its replica(s) which then checks to see if the checksums match the current block data gathered.
- Checkpoint Node: Combines previous checkpoints and journals to create a new checkpoint and empty journal. Done to assist in protecting the filesystem metadata. This checkpoint node runs on a different machine than the NameNode as it has the same memory needs.
- Inter-Cluster Data Copy: Tool called DistCp which interacts with MapReduce which maps the task copies of the source data to the destination.
- Replication Management: Used to verify all blocks have the number of intended replicas.
- TCP-Based Protocols: Servers are communicating through TCP.

Predicted components could be HDFS Administrator. This user would be used by the system admins so that the system could be maintained easier. This would have admin specific commands to assist the admins for system navigation and maintaining. Another predicted component could be YARN in Mapreduce, it is a major component of Hadoop, Yahoo uses this API, this helps applications that do not have issues with overloading nodes within the system.

An additional predicted component would be Rack Awareness as it can help deal with reliability, and bandwidth for replica storage.

How middleware transparencies are provisioned within the described image sharing system?

Middleware transparencies are used to make this system (it is distributed) seem like a singular system is operating when the image sharing system is actually running on a number of machines to ensure stability and efficiency of the system. The middleware in this case makes development an easier task due to the middleware masking the heterogeneity of the hardware of the system (the lower level system details are also hidden too). Middleware is used to make the system as a whole easier to manage.

A HDFS is a distributed system, Why do we build distributed systems? What are the advantages? What are the disadvantages?

We build distributed systems to make up for where centralized systems fall behind. Also DS are built as there is only so much monolithic systems are capable of, and the cost to have monolithic systems is not worth it when DS is a better option. For example centralized systems have lower availability, and reliability while compared to distributed systems. In essence, distributed systems are more flexible than centralized systems. Additionally, distributed systems are built as they have a greater ability to scale with fluctuating traffic. For example, when traffic is high more resources can be assigned to the system to accommodate the increased traffic.

Advantages:

- Greater efficiency: Traffic is sent to various locations rather than one centralized system.
- Scalability: When the system increases in traffic/popularity bottlenecks may be encountered and with distributed systems it is easy to add additional resources to accommodate the increased traffic. This scalability is known as horizontal scalability.
- Cheaper (over time): implementing distributed systems has an expensive initial investment cost, but the cost of investment over time more than makes up for the initial cost while compared to centralized systems.
- Reliability: A distributed system has the ability to continue to function even if some nodes fail. The overall efficiency may decrease when these failures occur, but the system will still function.
- Open: The system can be accessed remotely, but also locally.
- Performance: System can send and run queries faster as there is more than likely a database location near the user making the request.

Disadvantages:

- Complexity: Maintaining and troubleshooting a distributed system is difficult, as there are many points of failure.

- Communication Errors: In the case where messages are passed from system to system, there may be a specific sequence that the messages must arrive in, so checks must be put in place to assure that messages arrive in order. **In other words, data may be lost.**
- Security: Data is stored in different locations in a distributed system, and this data may be sensitive and this information needs to be secure no matter the location it is in, as well as making sure the correct access is granted upon requests for access (which can be challenging with distributed systems).
- Increased Software Development Costs: Hard to implement distributed systems and due to that, it is more expensive.

What applications distributed systems are most suitable for?

Applications that are most suitable for distribution are systems such as: eCommerce, telecommunications, streaming services, data hosting services like dropbox. All of these systems need to have high availability, reliability, efficiency as they are crucial services that are used daily by masses and downtimes of minutes could cost tens of thousands, and reduced service satisfaction.

Why are distributed systems hard?

Distributed systems are difficult for many different reasons. One being hardware failure, where networking cables are not safe from failure and will break over time. Software failures also exist and much like hardware, software is not infallible. Another difficulty with distributed systems is that when solving these issues in the system the complexity of the system is increased, making the system potentially harder to maintain and troubleshoot. Additionally distributed systems are hard as networks are not 100% secure, bandwidth is limited, and message/data passing does not happen instantaneously. Difficulty with distributed systems is also shown is data consistency/synchronization. This is when different versions of the same data are stored in different locations, with the question being which one will be sent (could be aided by updating each file when a change is made). **Latency of distributed systems is not zero as messages are passed over a network which will slow things down.**