Distributed Schema for Chat Application

01.05.2022

Team Members

Rajat Kataria (2019101020)

Umang Srivastava (2019101090)

Overview

The purpose of this project is to design a distributed chat application and to provide users with an instant messaging tool that has the ability to handle multiples of users simultaneously when needed and can be easily done. The aim for this project was to make it highly scalable and fault tolerant.

Goals

1. The system should be highly scalable : Multiple clients and servers should be able to work together in the distributed system.
2. It should be fault-tolerant : The system should be up and running, even if one or multiple servers crash.

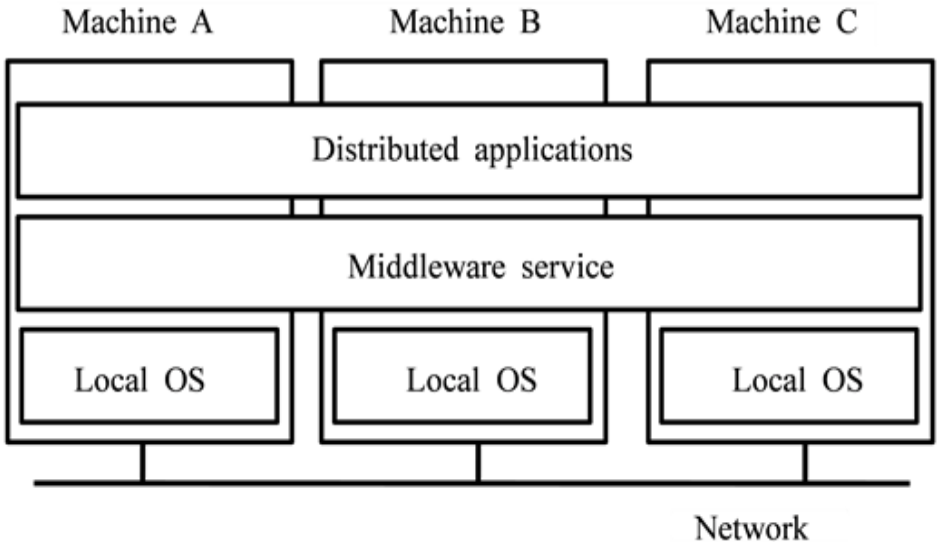
Distributed Systems

Distributed system are systems that don't share memory or clock, in distributed systems nodes connect and relay information by exchanging the information over a communication medium. The different computer in distributed system have their own memory and OS, local resources are owned by the node using the resources. While the resources that is been accessed over the network or communication medium is known to be remote resources.

In distributed system, pool of rules are executed to synchronize the actions of various or different processes on a communication network, thereby forming a distinct set of related tasks . The independent system or computers access resources remotely or locally in the distributed system communication environment, these resources are put together to form a single intelligible system. The user in the distributed environment is not aware of the multiple interconnected system that ensures the task is carried out accurately. In distributed system, no single system is required or carries the load of the entire system in processing a task.

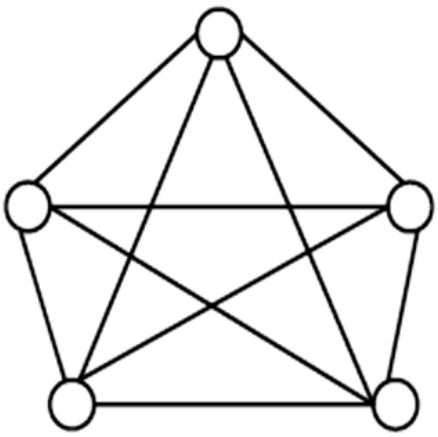
Architecture of Distributed Systems

The architecture of distributed system is built on existing OS and network software.. Distributed system encompasses the collection of self-sufficient computers that are linked via a computer network and distribution middleware. The distribution middleware in distributed system, enables the corresponding computers to manage and share the resources of the corresponding system, thus making the computer users to see the system as a single combined computing infrastructure . Middleware is the link that joins distributed applications across different geographical locations, different computing hardware, network technologies, operating systems, and programming languages. The middleware delivers standard services such as naming, concurrency control, event distribution, security, authorization etc.

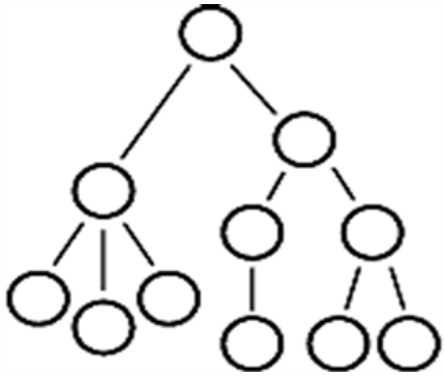


Different Types of Networks

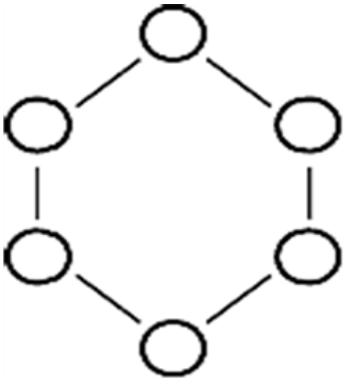
1. Fully connected network



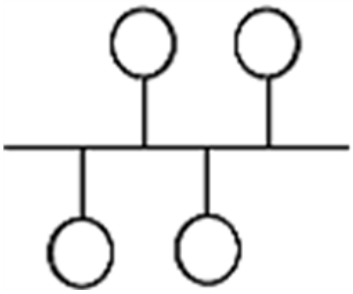
2. Tree structured network



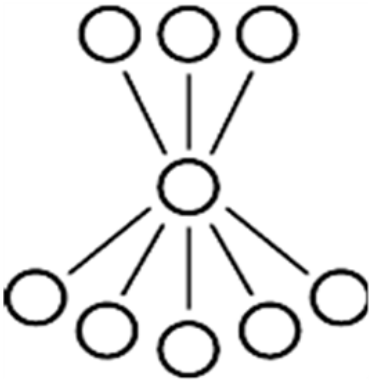
3. Ring structured network



4. Multi-access bus network



5. Star structured network



Brief Explanation about the algorithms used

I. Fault Tolerance

A faulty system creates a human/economic loss, air and rail traffic control, telecommunication loss, etc. The need for a reliable fault tolerance mechanism reduces these risks to a minimum. In distributed systems, faults or failures are limited or part. A part failure in distributed systems is not equally critical because the entire system would not be offline or brought down, for example a system having more than one processing cores (CPU), and if one of the cores fails the system would not stop functioning as though that's the one physical core in the system. Hence, the other cores would continue to function and process data normally. Nevertheless, in a non-distributed system when one of its components stops functioning, it would lead to a malfunction of the entire system or program and the corresponding processes would stop.

Fault tolerance is the dynamic method that's used to keep the interconnected systems together, sustain reliability, and availability in distributed systems. The hardware and software redundancy methods are the known techniques of fault tolerance in distributed systems. The hardware methods ensure the addition of some hardware components such as CPUs, communication links, memory, and I/O devices while in the software fault tolerance method, specific programs are included to deal with faults. Efficient fault tolerance mechanism helps in detecting faults and if possible recovers from it.There are various definitions to what fault tolerance is. In dealing with fault tolerance, replication is typically used for general fault tolerance method to protect against system failure

Fault tolerant Systems

Fault tolerant system is a vital issue in distributed computing; it keeps the system in a working condition in subject to failure. The most important point of it is to keep the system functioning even if any of its part goes off or faulty. For a system to be fault tolerant, it is related to dependable systems. Dependability covers some useful requirements in the fault tolerance system

these requirements include: Availability, Reliability, Safety, and Maintainability.

- Availability: This is when a system is in a ready state, and is ready to deliver its functions to its corresponding users. Highly available systems works at a given instant in time.
- Reliability: This is the ability for a computer system run continuously without a failure. Unlike availability, reliability is defined in a time interval instead of an instant in time. A highly reliably system, works constantly in a long period of time without interruption.
- Safety: This is when a system fails to carry out its corresponding processes correctly and its operations are incorrect, but no shattering event happens.
- Maintainability: A highly maintainability system can also show a great measurement of accessibility, especially if the corresponding failures can be noticed and fixed mechanically.As we have seen, fault tolerance system is a system which has the capacity of or to keep running correctly and proper execution of its programs and continues functioning in the event of a partial failure.

Although sometimes the performance of the system is affected due to the failure that occurred. Some of the fault is narrowed down to Hardware or Software Failure (Node Failure) or Unauthorised Access (Machine Error). Errors caused by fault tolerance events are separated into categories namely; performance, omission, timing, crash, and fail-stop..

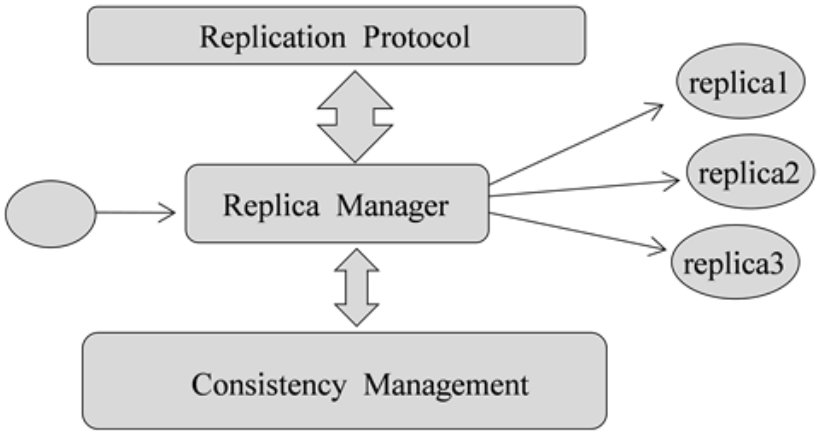
- Performance: this is when the hardware or software components cannot meet the demands of the user.
- Omission: is when components cannot implement the actions of a number of distinctive commands.Timing: this is when components cannot implement the actions of a command at the right time.
- Crash: certain components crash with no response and cannot be repaired.
- Fail-stop: is when the software identifies errors, it ends the process or action, this is the easiest to handle, sometimes its simplicity deprives it from handling real situations.

In addition to the error timing, three situations or form can be distinguished:

- Permanent error; these causes damage to software components and resulting to permanent error or damage to the program, preventing it from running or functioning. In this case a restart of the program is done, an example is when a program crashes.
- Temporary error; this only result to a brief damage to the software component, the damage gets resolved after some time and the corresponding software continues to work or function normally.
- Periodic errors; these are errors that occurs occasionally. For example when there's a software conflict between two software when run at the same time. In dealing with this type of error, one of the programs or software is exited to resolve the conflict.

Most computers if not all have some fault tolerance technique such as micro diagnosis,, parity checking, watchdog timer,, etc. an incompletely fault tolerant system have inbuilt resources to cause a reduction in its specified computing capability and reduce to a smaller or lower system by removing some programs that have been used previously or by reducing the rate at which specified processes are executed. The reduction is due to the decrease or slowdown in the operational hardware configuration or it may be some design faults in the hardware.

Replication Based Fault Tolerance Technique



The replication based fault tolerance technique is one of the most popular method. This technique actually replicate the data on different other system.

In the replication techniques, a request can be sent to one replica system in the midst of the other replica system. In this way if a particular or more than one node fails to function, it will not cause the whole system to stop functioning. Replication adds redundancy in a system.There are different phase in the replication protocol which are client contact, server coordination, execution, agreement, coordination and client response. Major issues in replication based techniques are consistency, degree of replica, replica on demand etc.Consistency: This is a vital issue in replication technique. Several copies of the same entity create problem of consistency because of update that can be done by any of the user. The consistency of data is ensured by some criteria such as linearizability , sequential consistency and casual consistency etc. sequential and linearizability consistency ensures strong consistency unlike casual consistency which defines a weak consistency criterion. For example a primary backup replication technique guarantee consistency by linearizability, likewise active replication technique.

Degree or Number of Replica: The replication techniques utilizes some protocols in replication of data or an object, such protocol are: Primary backup replication , voting and primary-per partition replication. In the degree of replication, to attain a high level of consistency, large number of replicas is needed. If the number of replica is low or less it would affect the scalability, performance and multiple fault tolerance capability. To solve the issue of less number of replica, in adaptive replicas creation algorithm was proposed.

II. Dynamic discovery of hosts

Here is some scenario:

Hosts: A, B, C, D, E, F, G

Your application knows only node (server) A. So your application asks A to join the net. And after A verifies your application to be allowed to and able to join the net tells your application all about B, C, D, E, F, G. Also the server A tells all the other nodes about your application joining the net. After this simple routine all hosts know about your application and your application knows about all the hosts.

Now lets say Bobs application is also joinging the network asking sever E to join. Bobs application learns all about the other hosts A, B, C, D, F, G. And E tells Bobs application about your application. But then there is this message (or event) distribution system your application listens to and E just publish a message that Bobs application joined. Since your application listens to this kind of messages your application learns that Bobs application has also joined.

After some time server D crashed. Since the hosts A, B, C, E, F, G constantly listens to messages of D they notice that D is not sending any messages any more. So they send messages telling everyone listening that D has left the network. After a short periode D recovered and rejoined the network... .

Thats basically all that is important about dynamic discovery of hosts. There are a bunch (at least one) of servers, you only know one or more (at least 3 are mostly recommanded if one has crashed and is recovering) and asks to join and to get some more information about other hosts. Then your application listens to status informations or is requesting one of the server to keep your application informed about the events of joining and leaving of servers from the logical network.

III. Voting and Leader election

The objective of leader election is to provide one item (a process, host, thread, object, or human) specific powers in a distributed system. These specific abilities could include the capacity to delegate tasks, the ability to edit data, or even the responsibility for managing all system requests.

The election of a new leader is a powerful instrument for increasing efficiency, minimizing coordination, simplifying architectures, and lowering activities. Leader election, on the other hand, may generate additional failure modes and scale obstacles. Furthermore, leader election may make evaluating the validity of a system more challenging.

The goal of the leader election is to select a processor who will coordinate the system's actions. A leader is usually chosen based on a criterion, such as

selecting the processor with the greatest identifier as the leader in any leader election procedure. The processors attain their terminated states when the leader is chosen. In a leader election algorithm, the terminated states are divided into elected and non-elected states. When a processor enters a non-elected state (or an elected state), it remains in that state at all times (or an elected state). The safety and liveness conditions for a leader election algorithm must be met:

- According to the liveness condition, every processor will eventually enter either an elected or non-elected state.
- Only one processor is allowed to enter the elected state as a safety condition for leader election. This processor takes over as the distributed network's leader.

Use Cases

When considering whether or not a leader election is appropriate, there are three scenarios to consider. When each node is roughly the same, and there isn't a clear contender for a permanently assigned leader, the first circumstance arises. This means that any node can be voted as the system's leader, and there is no single point of failure.

The second scenario occurs when the cluster is performing a very complex activity that requires close collaboration. Coordination can refer to various things, including deciding how the work will be divided, assigning work to certain nodes, and combining work outcomes from various nodes.

Consider the case of a scientific computation aimed at determining how a protein folds. This computation can take a long time due to many possible answers, but it will be significantly sped up if it is dispersed. A leader node will be required to allocate each node to a distinct component of the computation and then combine the results to obtain the whole folded protein configuration.

When a system executes many distributed writes to data and requires good consistency, the third instance where leader election brings value is when it executes many distributed writes to data. You can learn more about consistency in our Databases article. Still, it basically means that no matter which node processes a request, the user will always have the most recent data version. In this case, a leader ensures consistency by being the source of

truth about the system's most recent condition (and the leader election algorithm must preserve this properly).

Although not all applications require strong consistency, you can imagine how important it would be for a bank to ensure that a user's bank account total is accurate regardless of which server responds to their online banking request and that multiple transactions directed to the same bank account do not conflict with one another.

Advantages and Disadvantages

Leader election is a common pattern in distributed systems because it has some significant advantages:

- Humans find it easier to think about systems with a single leader. It consolidates all of the system's concurrency, lowers partial failure possibilities, and provides a single location to search for logs and metrics.
- A single leader can accomplish more. Rather than forming consensus on the changes to be made, it might often merely tell other systems about them.
- Because they can observe and control all of the changes made to the system's state, single leaders can easily provide consistency to their clients.
- By providing a single consistent cache of data that can be used simultaneously, a single leader can improve performance or lower costs.
- It may be simpler to write software for a single leader than writing software for a quorum. The single leader does not consider other systems that may be working in the same state simultaneously.

Leader election also has some considerable downsides:

- A single point of failure is a single leader. The entire system may become unavailable if the system fails to detect or correct a bad leader.
- When it comes to data size and request rate, a single leader equals a single point of the scale. A complete re-architecture is required when a leader-elected system needs to expand beyond a single leader.

- A single point of trust is a leader. When a leader does the wrong task and checks it, it can swiftly spread problems throughout the system. The blast radius of a bad leader is large.
- In leader-elected systems, partial deployments may be difficult to implement. Partial deployments are used in many Amazon software safety policies, including one-box, A/B testing, blue/green deployment, and incremental deployment with automatic rollback.

Leader Election Algorithms

With as few back-and-forth interactions as feasible, a leader election algorithm guides a cluster to agree on one node to operate as a leader jointly.

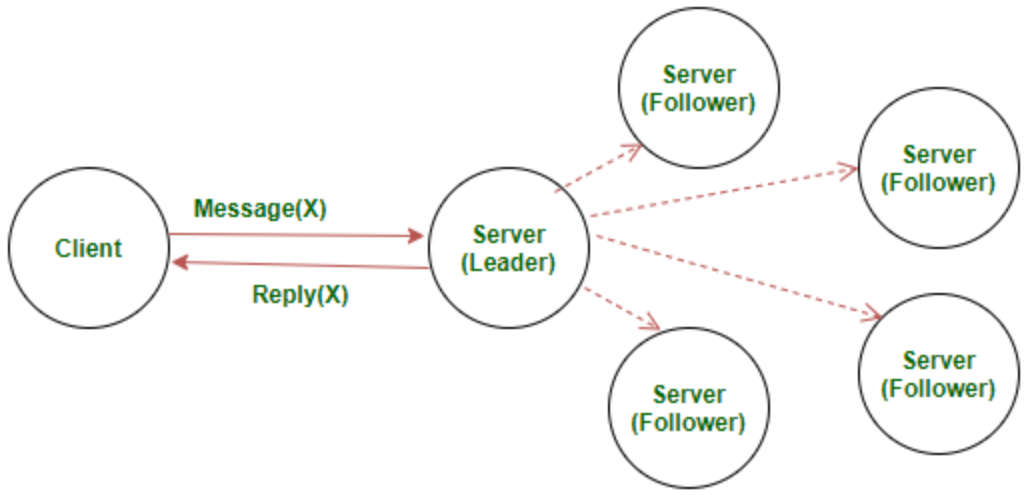
In general, the method assigns each node one of three states: Leader, Follower, or Candidate. In addition, the leader will be required to pass a “health check” or “heartbeat” regularly so that follower nodes may determine if the leader has been unavailable or failed, and a new one must be elected.

Whether you want a synchronous or asynchronous cluster determines the type of leader election mechanism you use. The asynchronous cluster has nodes synced to the same clock and sends messages in a predictable order. Messages are not reliably sent within a particular period of time or in any order in an asynchronous cluster.

Because any number of nodes in an asynchronous cluster can lag indefinitely, the leader election procedure can’t ensure both safety (no more than one leader will be elected) and liveness (every node will finish the election). In practice, implementations choose to ensure safety because it has more serious consequences for the service.

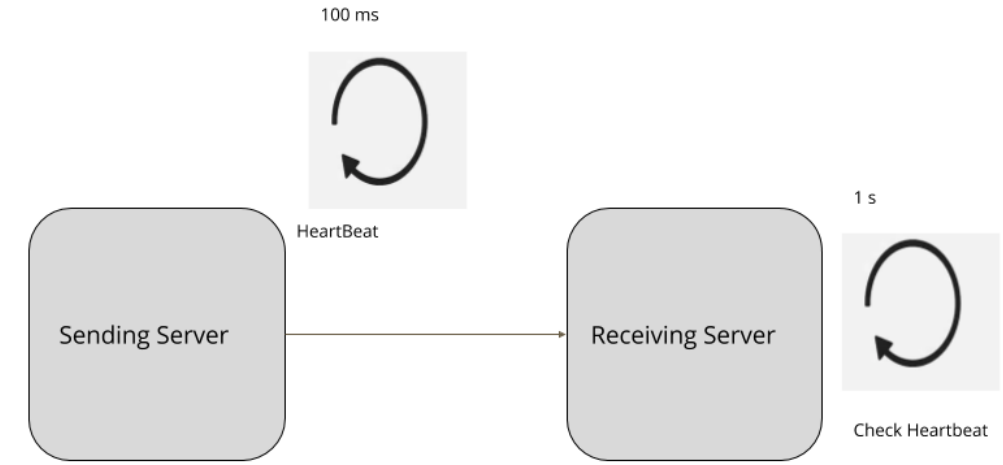
Synchronous algorithms are easy to reason about and potentially desirable because they can guarantee both safety and liveness. However, in practice, synchronizing a cluster necessitates imposing extra limits on how it runs, which aren’t necessarily possible or scalable.

Example : Bully algorithm, Paxos, Raft, etc.



IV. Heartbeat mechanism

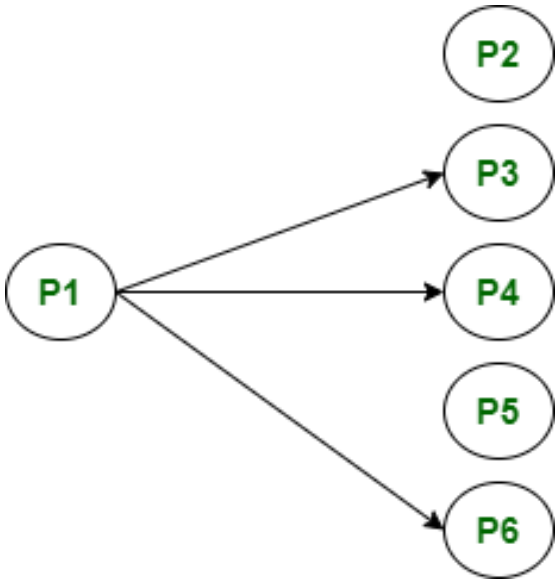
In distributed systems, a heartbeat is a periodic signal generated by hardware or software to indicate normal operation or to synchronize other parts of a computer system. Heartbeat mechanism is one of the common techniques in mission critical systems for providing high availability and fault tolerance of network services by detecting the network or systems failures of nodes or daemons which belongs to a network cluster—administered by a master server—for the purpose of automatic adaptation and rebalancing of the system by using the remaining redundant nodes on the cluster to take over the load of failed nodes for providing constant services. Usually a heartbeat is sent between machines at a regular interval in the order of seconds; a heartbeat message. If the endpoint does not receive a heartbeat for a time—usually a few heartbeat intervals—the machine that should have sent the heartbeat is assumed to have failed. Heartbeat messages are typically sent non-stop on a periodic or recurring basis from the originator's start-up until the originator's shutdown. When the destination identifies a lack of heartbeat messages during an anticipated arrival period, the destination may determine that the originator has failed, shutdown, or is generally no longer available.



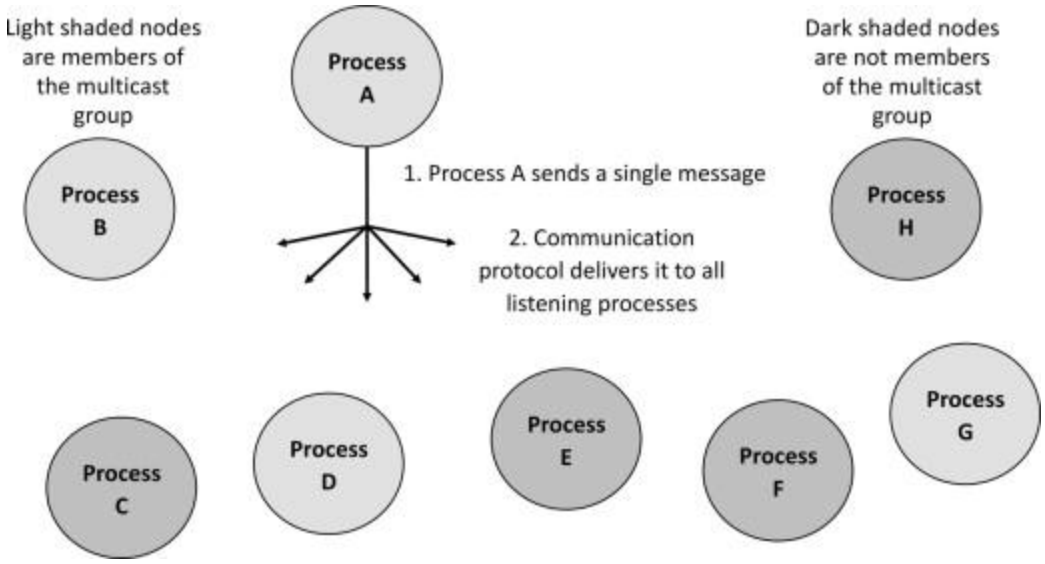
V. Multicast

Communication between two processes in a distributed system is required to exchange various data, such as code or a file, between the processes. When one source process tries to communicate with multiple processes at once, it is called Group Communication. A group is a collection of interconnected processes with abstraction. This abstraction is to hide the message passing so that the communication looks like a normal procedure call. Group communication also helps the processes from different hosts to work together and perform operations in a synchronized manner, therefore increases the overall performance of the system.

When the host process tries to communicate with a designated group of processes in a distributed system at the same time. This technique is mainly used to find a way to address problem of a high workload on host system and redundant information from process in system. Multitasking can significantly decrease time taken for message handling.



A reliable multicast is any computer networking protocol that provides a reliable sequence of packets to multiple recipients simultaneously, making it suitable for applications such as multi-receiver file transfer. Multicast is a network addressing method for the delivery of information to a group of destinations simultaneously using the most efficient strategy to deliver the messages over each link of the network only once, creating copies only when the links to the multiple destinations split (typically network switches and routers). However, like the User Datagram Protocol, multicast does not guarantee the delivery of a message stream. Messages may be dropped, delivered multiple times, or delivered out of order. A reliable multicast protocol adds the ability for receivers to detect lost and/or out-of-order messages and take corrective action (similar in principle to TCP), resulting in a gap-free, in-order message stream.



Files Details

I. Client.py

Contains the code to run the client for chatting application.

II. Server.py

Contains the code for the server part of the project.

III. Cluster/heartbeat.py

Contains code for the heartbeat mechanism to check if servers are alive or not.

IV. Cluster/hosts.py

Contains information about the hosts, sockets, ips, etc. of multicast, server and client.

V. Cluster/leader_election.py

Contains code for the leader election in case of a leader server fails.

VI. Cluster/ports.py

Contains information about ports on which server and multicast is running.

VII. Cluster/receive_multicast.py

Contains code for multicast receiver.

VIII. Cluster/send_multicast.py

Contains code for multicast sender.

Application Demo Video

https://drive.google.com/drive/folders/1FbSGiwZTE8skRIXrOl5aQfs4AEqsk_x6?usp=sharing