International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 03 | Mar 2022 www.irjet.net p-ISSN: 2395-0072

Clock Synchronization in Distributed Systems

Amey Thakur[1] , Mega Satish[2]

[1-2]*Department of Computer Engineering, University of Mumbai, Mumbai, MH, India*

Abstract — *Clock discrepancies are troublesome in distributed systems and pose major difficulties. To avoid mistakes, the clocks of separate CPUs must be synced. This is to ensure that communication and resource sharing are as efficient as possible. As a result, the clocks must be constantly monitored and adjusted. Otherwise, the clocks drift apart. Clock skew causes a disparity in the time values of two clocks. Both of these issues must be solved in order to make effective use of distributed system characteristics. In this study, we briefly explored the features of distributed systems and its algorithms.*

Keywords — Clock Synchronization, Distributed Systems, Physical Clock, Logical Clock, Cristian Algorithm, Berkeley Algorithm, Network Time Protocol (NTP), Lamport's Clock, Vector Clock, Bully Algorithm, Ring Algorithm.

1. INTRODUCTION

1.1 Distributed systems and its types

Distributed System (DS) is a collection of computers connected via a high-speed communication network [1]. A distributed system is one in which interconnected hardware and software interact and coordinate their operations only through exchanging messages.

There are two types of Distributed Systems:

1. Homogeneous Distributed Systems:
- It is a distributed system such that all nodes have identical hardware, the same type of architecture, and operating system.

2. Heterogeneous Distributed Systems:
- It is a distributed system such that each node has its own operating system and machine architecture. Each node in a distributed system can share its resources, e.g., the producer-consumer processes and the client-server processes, sharing a printer or scanner. However, resources are finite and can be distributed in either collaborative or competitive forms. Resources like a printer and scanner cannot be used by multiple processes simultaneously, so they must wait for one process to complete and then give a chance to the next process. Another instance is producer-consumer as well as client-server operations that operate in extensive cooperation.

1.2 Need to resynchronize the clocks

So there is a need for proper allocation of available resources, to preserve the state of resources and coordination between processes. Clock synchronization [2] is critical for resolving these problems. Clock synchronization can be implemented by using the physical clock and logical clock.

1.3 Issues in Clock Synchronization

A basic technique of clock synchronization [2][3] is for each node to submit a time query message to the real-time server. The node gets a reply message with the value of time 't'. This method has the following issues:

- Every node's capacity to read the clock value of another node. This can raise errors due to delays in message communication between nodes. The time required to prepare, deliver, and get a blank message because of the lack of transmission problems and system load can be used to calculate delay.
- Time should never be reversed since it might lead to the recurrence of events or transactions, causing chaos in the system. Time going backwards is only a notion; it does not literally travel backwards.

1.4 Reasons for Delay in Synchronization

As discussed above, there are many reasons [3][4] for a communication delay that needs to be minimized to minimize delay and get a nearby accurate time.

1. Communication Link Failure: For example, when sending a request message, the communication link is working properly and the message reaches

the server. If at the time of receiving a message, the communication link may fail due to some break. And the client may not be able to get a reply message. After recovery, the reply reaches the client which contains a false time value.

2. Fault Tolerance: If any failure happens during exchanging messages, the clock time may be incorrectly interpreted. So the system should be fault-tolerant so that it can work in a faulty situation and minimize the clock drift value.
3. Propagation Time: Due to heavy traffic or congestion in the network, it may cause a large propagation time from server to client. It may cause the inaccurate reading of the clock value in the reply.
4. Non-Receipt of Acknowledgement: It may be possible that due to the above reasons client will not get a reply within a round trip time and therefore it sends multiple requests to the server for synchronization.
5. The Bandwidth of Communication Link: Due to the low bandwidth of communication links, congestion may occur in the network. As a result, requests for time will be unable to reach the server and reply messages will be unable to get to the client, affecting clock synchronization.

2. Clocks in Synchronization

In synchronization [2], there are two types of clocks.

1. Physical Clock:
- Time isn't a big issue in traditional centralized systems, where one or more CPUs share a common bus. The entire system shares the same understanding of time, right or wrong, it is consistent.
- In distributed systems, this is not the case. Every system, though, has its own timer that keeps the clock running. These clocks are based on the oscillation of a piezoelectric crystal or a similar integrated circuit. They are not flawless, but they are relatively precise, reliable, and accurate. This implies that the clocks will differ from the correct time. Every timer is different in terms of characteristics — characteristics that might change with time, temperature. Thus, each system's time will drift away from the true time at a different rate — and perhaps in a different direction (slow or fast).
- It is feasible to coordinate physical clocks across several systems, but it will never be accurate. The drifting away from the real-time from each clock is something that happens in a distributed system.

2. Logical Clock:
- Logical clocks mean creating a protocol on all computers in a distributed system so that the computers can keep a uniform ordering of happenings inside some virtual time range.
- In a distributed system, a logical clock is a technique for recording temporal and causative links. Because distributed systems may lack a physically synchronized global clock, a logical clock provides for the global ordering of occurrences from various processes in certain systems .

3. Clock Synchronization Algorithms

3.1 Cristian Algorithm

Cristian's Algorithm is a centralized clock synchronization algorithm used to synchronize time with a time server by client processes. This algorithm works well with a low latency network where the round-trip time — time duration between the start of request and end of corresponding response — is short as compared to the accuracy. It is an approach in which the client approaches the server.

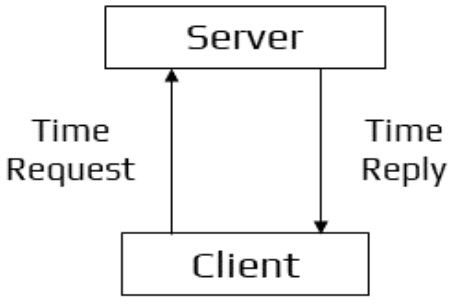
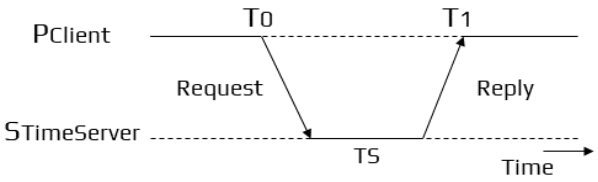


Figure 1: Cristian's Algorithm Workflow

A client sends a request to a time server for its current value of the UTC time (□s). The client records the time its request was submitted (□0) and the time it got the response (□1). Then, the client changes its current time at □1 with the value received from the server plus its estimate of the delay in obtaining this value, resulting in the total time required to submit the query and get the

response, which is (𝑇1−𝑇0)/2. The new time value is thus 𝑇s+(𝑇1−𝑇0)/2.



$$Tnew = Tserver + T1 - T0 / 2$$

Figure 2: Cristian's Algorithm Working

Algorithm:

- Let S be the time server and Ts be its time.
- Process P requests the time from S.
- After receiving the request from P, S prepares a response and appends time Ts from its own clock and then sends it back to P.

3.2 Berkeley Algorithm

The Berkeley Algorithm is a centralized clock synchronization mechanism in a distributed system that implies no computer has a precise timing source. The algorithm was developed by Gusella and Zatti at the University of California, Berkeley in 1989.

This algorithm is an example of an active time server approach: the time server periodically sends a message to all the computers in the group. When the message is received, each computer then sends back its own clock value to the time server. It is an approach in which the server approaches the client.

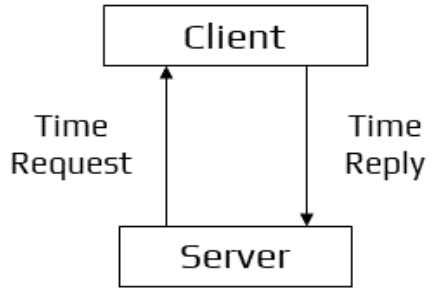


Figure 3: Berkeley's Algorithm Workflow

Here the time server has prior knowledge of the approximate time required for the propagation of the message which is used to readjust the message. The time server then takes the average of all clock values of all the computers. All clocks should be readjusted to the current time which is the calculated average. The time server readjusts its own clock value to this value and instead of sending the current time to other computers, it sends the amount of time each computer needs for readjustment. The value may be positive or negative.

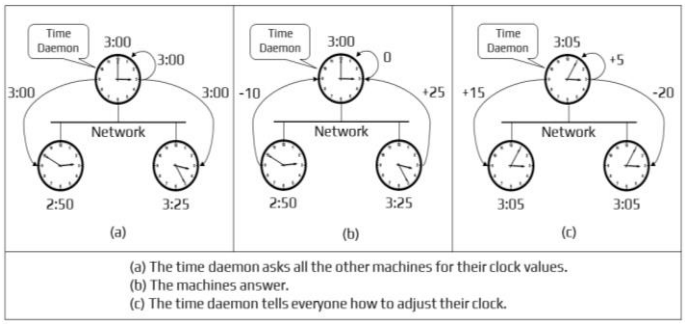


Figure 4: Berkeley's Algorithm Working

3.3 Network Time Protocol

Network Time Protocol is a standard followed by synchronization clocks on the internet. It is a decentralized algorithm.

The Network Time Protocol (NTP) is a commonly employed Internet Engineering Task Force (IETF) standard (RFC 1305). (RFC 1305). The main servers are directly linked to a precise and dependable UTC time source. They are the foundations of hierarchical time service, with additional servers becoming operational as we go away from the roots. The common configuration includes UTC time servers at big government institutions at stratum 1, institutional time servers or Internet service providers' time servers at stratum 2, and most users linking to academic time servers at stratum 3.

NTP may synchronize computers in three modes: first is the client-server mode, the second is the multicast mode, and the last is the symmetrical (peer) mode. In the client-server mode, the client makes queries to the server upon startup and on a regular basis thereafter. In a way similar to Cristian's technique, it tracks the time at which the request and response are delivered and received in order to factor out network latency as much as feasible. Because the server multicasts its time value on a regular basis, the multicast mode is frequently more efficient. On a local area network with multicast capabilities, time resynchronization can be accomplished in a single

message rather than two messages per client (e.g., Ethernet). However, in order to assess the network latency and adjust for it, the clients must first conduct a few client-server queries. However, if the network parameters change over time, the multicast mode's accuracy will be inferior to that of the client-server mode.

The Simple Network Time Protocol (RFC 2030) is a Network Time Protocol adaptation that supports operation in a stateless remote procedure call mode or multicast mode. It is designed for use in contexts where a complete NTP implementation is neither required nor warranted. SNTP is intended to be utilized at the endpoints of the synchronization subnet (high stratum) rather than for time server synchronization.

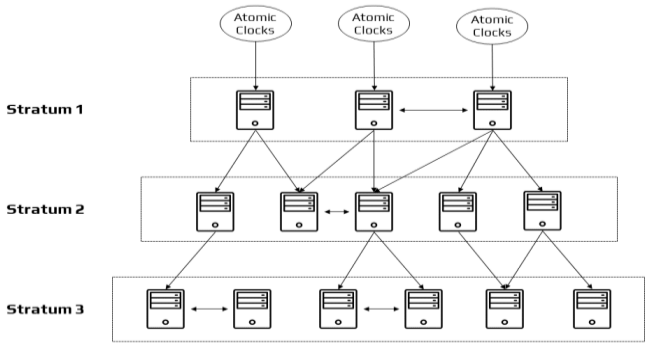


Figure 5: Architecture of Network Working Protocol

3.4 Lamport's Clock

In a distributed system, it is not necessary for the clocks to be absolutely synchronized. If two processes do not interact with each other, it is not necessary that their clocks need to be synchronized because the lack of synchronization would not matter. It is not important for all processes to agree on what the current time is but they should agree on the order in which events occur.
In a distributed system, Lamport Clocks [5] are a basic mechanism for identifying the sequence of events. It gives a "Happened-Before" sequencing of occurrences. If there is no "happened-before" relationship, then the events are considered concurrent.

Algorithm:
The "Happened before" relation between a and b is a -> b, which means 'a' happened before 'b'.
The criteria for logical clocks are:

- Clock 1: Ci (a) < Ci(b), Ci -> Logical Clock, if 'a' happened before 'b', then the time of 'a' will be less than 'b' in a particular process.
- Clock 2: Ci(a) < Cj(b), Clock value of Ci(a) is less than Cj(b).

3.5 Vector Clock

In Lamport's clock, if x -> y, then T(x)<T(y). But this does not tell about the relationship between events x and y. That's because Lamport's clock do not capture causality. The causal relationship between messages is captured through vector clocks.

Vector Clock [9] is an algorithm that creates a partial ordering of occurrences and identifies causality breaches in a distributed system. Such clocks extend on vector time to provide for a logically coherent picture of the distributed system; they identify if a contributed activity has triggered another activity. It essentially captures all the causal relationships. This approach assists in labelling each process within the system with a vector (a list of numbers) including an integer for each local clock. As a result, for every N process, there will be a vector of size N.

Algorithm:

- All of the clocks are initialized to zero.
- When an internal event happens in a process, the number of the process's logical clock in the vector is increased by one.
- Also, every time a process sends a message, the value of the process's logical clock in the vector is incremented by one.
- Every time a process receives a message, the value of the process's logical clock in the vector is incremented by one, and moreover, each element is adjusted by calculating the maximum value of the vector clock and the vector value in the incoming message.
-

3.6 Election Algorithms

Algorithms used in distributed systems necessitate the usage of a coordinator who performs duties required by other processes in the system. Election algorithms [7] are meant to select a coordinator.

Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process fails for whatever reason, another processor chooses a new coordinator. The election algorithm decides where a new copy of the coordinator should be begun.

The election method is based on the assumption that each active activity in the system has a distinct priority number. As a new coordinator, the process with the utmost priority will be picked. Hence, when a coordinator fails, this algorithm elects that active process that has the highest priority number. Then this number is sent to every active process in the distributed system.

3.6.1 Bully Algorithm:

Bully Algorithm [7] was proposed by Garcia Molina. This algorithm is planned on assumptions: a. Each process in a situation has a process identifier that may be used to identify it uniquely.

Each process should know the process numbers of all the remaining processes.

The process with the highest process number is elated as coordinator.

Algorithm:

- A process P notices that the coordinator is no longer responding, it will initiate an election.
- P will send the election to all other processes with a higher process id than its. If no one responds, process P becomes the coordinator.
- If one of the higher processes answers, P's job is done and the higher process will take over.
- When process P gets a message from one of its lowered id, it sends an OK message to the sender that it will take over and that the process is alive.
- Eventually, all processes will give up apart from one, which is the coordinator. The coordinator finally wins and announces its victory by sending a message to everyone.

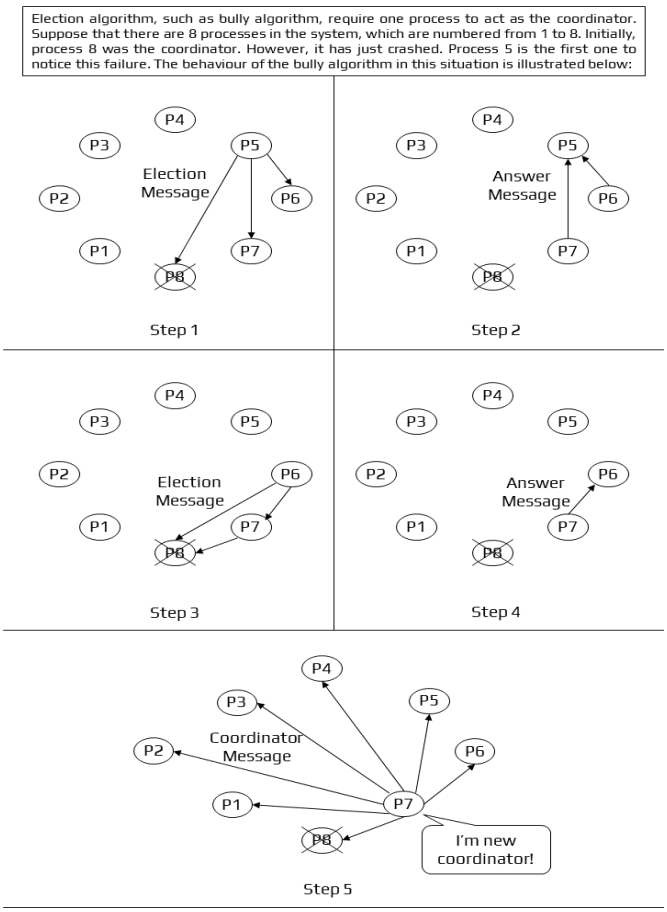


Figure 6: Working of Bully Algorithm

3.6.2 Ring Algorithm:

The ring algorithm [8] is another example of an election algorithm. This algorithm assumes that the processes are arranged in a logical ring and each process knows the order of the ring of processes. The processes are able to 'skip' faulty systems - systems that don't respond in a fixed amount of time.

Algorithm:

- Here, when the process notices that the coordinator is dead, it builds and sends an election message to other processes.
- At every step, processes keep on adding their own id at the end of this.
- This stops when the initiator -- a process that started the election -- receives the message it sent.
- After this, the process with the higher id is declared to be a coordinator.
- The initiator then announces the coordinator by sending the message to the nodes.
- Here the maximum number of initiators is 2.

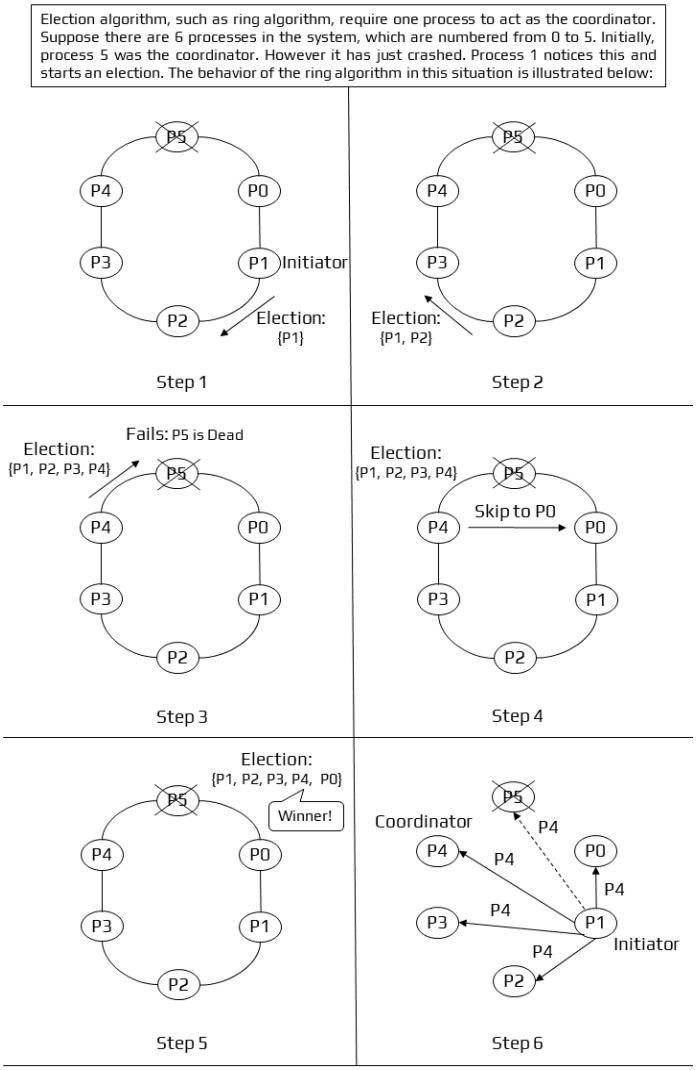


Figure 6: Working of Ring Algorithm

CONCLUSION

Several synchronization algorithms in distributed systems have been studied in this paper. In terms of algorithms, we can conclude that for clock synchronization, both centralized and distributed algorithms must account for the propagation time of messages among each node. The sequencing of processes and the preservation of resource status requires clock synchronization. When it comes to the concept of time in distributed systems, the most essential element is to get the events in the right sequence. Events can be positioned either in chronological order with Physical Clocks or in a logical order with Lamport's Logical Clocks and Vector Clocks along the execution timeline.

REFERENCES

[1] Latha, C. A., and H. L. Shashidhara. "Clock synchronization in distributed systems." In *2010 5th International Conference on Industrial and Information Systems*, pp. 475-480. IEEE, 2010.

[2] Horauer, Martin. "Clock synchronization in distributed systems." PhD diss., 2004.

[3] Sampath, Amritha, and C. Tripti. "Synchronization in distributed systems." In *Advances in Computing and Information Technology*, pp. 417-424. Springer, Berlin, Heidelberg, 2012.

[4] Biradar, Shripad, Santosh Durugkar, and Subhash Patil. "Handling Clock synchronization Anomalies in Distributed System."

[5] Simons, Barbara. "An overview of clock synchronization." *Fault-Tolerant Distributed Computing* (1990): 84-96.

[6] Welch, Jennifer Lundelius, and Nancy Lynch. "A new fault-tolerant algorithm for clock synchronization." *Information and computation* 77, no. 1 (1988): 1-36.

[7] Arghavani, A., E. Ahmadi, and A. T. Haghighat. "Improved bully election algorithm in distributed systems." In *ICIMU 2011: Proceedings of the 5th international Conference on Information Technology & Multimedia*, pp. 1-6. IEEE, 2011.

[8] Soundarabai, Paulsingh & Thriveni, J. & Manjunatha, H. & K R, Venugopal & Patnaik, Lalit. (2013). Message Efficient Ring Leader Election in Distributed Systems.

[9] Baldoni, Roberto, and Michel Raynal. "Fundamentals of distributed computing: A practical tour of vector clock systems." *IEEE Distributed Systems Online* 3, no. 2 (2002): 12.