International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 05 | May 2021 www.irjet.net p-ISSN: 2395-0072

Analysis of Timing and Synchronization Algorithms in Distributed Systems

Jennifer Mento Clemens[1], Sourab A.[2]

[1]*Student, Dept. of Computer Science and Engineering, Vellore Institute of Technology (VIT) Chennai, TamilNadu, India*
[2]*Student, Dept. of Computer Science and Engineering, VIT Chennai, TamilNadu, India*

Abstract - *Distributed systems have become an ever-growing domain in today's life. New and upcoming technologies are making use of Distributed Systems. However, they come with their own challenges. Each node in a Distributed System has its own internal clock. There would be no problem in synchronizing these nodes if each of their clocks maintained its frequency and all started together at the exact same time. However, this is not the case in real life. In real life, a clock may tend to slow down or speed up as time goes by. This is known as drift. To overcome this drift, there exist several algorithms and protocols. Each of these algorithms have its own pros and cons. Some may be more suitable than others in certain scenarios. In this paper we analyze the different algorithms and protocols proposed for implementing timing and synchronization in Distributed Systems. Finally, the analysis of our research describe which algorithms and protocols are suitable in which cases.*

Key Words: **Distributed Systems, Clock Synchronization, Mutual Exclusion algorithm, Lamport's algorithm, Cristian's algorithm, Berkeley's algorithm, PCS Algorithm**

1.INTRODUCTION

What is it about Distributed operating systems that is making it the current trend among various technologies? There are many obvious advantages to the use of these systems. When one node fails, the rest of the network is unaffected. Another advantage is that the scalability of the network becomes much simpler. Addition and removal of computers in the network is possible without much disturbance to the rest of the network. In contrast to distributed systems, centralized systems may get overwhelmed when too many nodes are connected to it.

This does not mean that there are no challenges when it comes to a distributed system. In fact, there are plenty. These include distribution transparency, i.e., the extent to which a system appears as a whole, monitoring user behaviour without being intrusive, etc. [1]. One challenge of utmost importance which is discussed in this paper is the clock synchronization of every node in a distributed system. When talking about clock synchronization in a Distributed System, one must understand the certain concepts.

First, we have logical and physical clocks. Every computer has its own internal crystal clock that does not require electricity to work. This clock is always ticking and keeping track of time. This is how the time in our computers is always running whether or not it is connected to the Internet. This internal crystal clock is the physical clock of the system. On the other hand, the logical clock of a computer is the maintenance of the order of processes taken place in it. It allows the processor to understand which process happened before which.

Now, it may seem obvious that we can start all the physical clocks in a distributed system at the same exact time. This way all the clocks are always in sync. However, the physical clocks in each system undergo a phenomenon called 'drift'. Drift is the gradual change in speed of the physical clock in a system with respect to time. The clocks may go slower or they may go faster. Either case results in unsynchronized clocks.

There are two types of synchronizations – external and internal synchronization. External synchronization is where an external entity looks over the clocks of the system and make sure they are in sync. In internal synchronization, the nodes of the distributed system themselves keep a regular check with the other systems to make sure its own clock is in sync.

There is the clear solution of allowing all computers to use a network time source (for getting the UTC time) to allow all the nodes on a distributed system to synchronize their clocks. This way the distributed system allows the use of an external synchronization of each node's physical clocks. However, the problem with this is that the clocks will be in sync only when connected to the Internet. Since, a stable and noiseless Internet connection cannot be promised forever, using UTC time is not an efficient solution.

Having made clear the need of synchronization in a Distributed System, in the following sections of this paper, we have studied the various algorithms that have been proposed for synchronising the clocks in a distributed system for each node. Each algorithm has its pros and cons. This paper gives an in-detail explanation of Lamport's algorithm using Lamport's mutual exclusion, Berkeley' algorithm, Cristian's algorithm, and finally the PCS algorithm.

2. Lamport's Algorithm

Leslie Lamport had introduced the idea of mutual exclusion in 1987[2].

This kind of algorithm was proposed by Lamport as an illustration of his synchronization scheme for distributed systems. It is based on permission.

In these permission-based algorithms, timestamp is used to resolve the conflicts between requests and for ordering critical section requests. The critical section request executed by timestamp is done in the increasing order of the timestamps. Hence a request with smaller timestamp will be executing critical section first rather than a request with higher timestamp.

2.1 Algorithm

This algorithm consists of 3 kinds of messages:(REQUEST, REPLY, and RELEASE). The communication channels are considered to follow First In First Out (FIFO) order. A given site (S1) will send a REQUEST to all the other sites to get their permission to enter the critical section. Then one of the other sites (S2) sends a REPLY to the requested site (S1) to grant that site's permission to enter critical section. The allowed site then RELEASES a message to all the sites after they have exited from the critical section.

The critical section requests are stored by each site in a queue.

Ex: request_queue(i) = queue of site(i)

Using Lamport's logical clock a timestamp is given to each critical section. As said earlier the requests are accepted in the increasing order of timestamps.

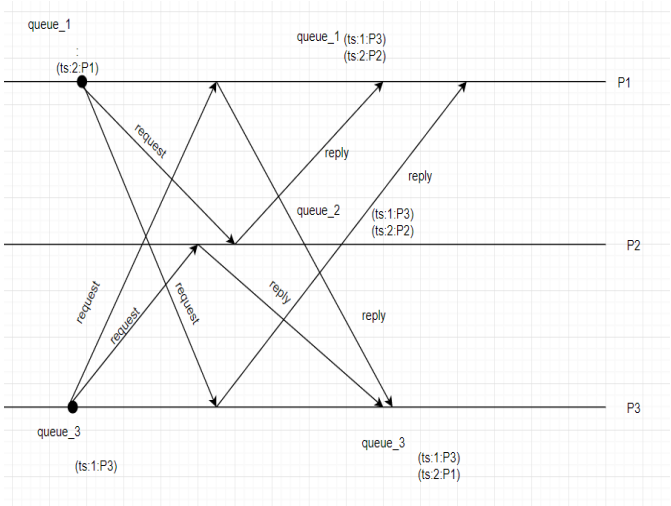

Fig -1: Demonstration of Lamport's algorithm.

2.1.1 Entering the critical section

When a site s(i) wants to enter a critical section, it sends a request message request(ts(i),i) to all the sites. Here, ts(i) refers to the timestamp of s(i). If the site s(j) receives this request message request(ts(i),i), it returns a timestamped reply to s(i) and places that on the request_queue(j).

2.1.1 Executing the critical section

The site s(i) can enter the critical section if its own request is at the top of the request_queue(i) and if it has received the message with timestamp larger than (ts(i),i).

2.1.1 Leaving from critical section

The site s(i) removes its own request from its request queue and sends a timestamped release message to all other sites. Hence after receiving this release message from s(i), the site s(j) also removes the request of s(i) from its queue.

2.2 Drawbacks of Lamport's algorithm

One of the main disadvantages is its high message complexity. The algorithm requires invocation of 3(n-1) messages per critical section execution. Progress of the entire system will be stopped if any one of the processes fails.

3. Cristian's Algorithm

Round-trip time is an important concept in Cristian's algorithm. In simple words, round-trip time is the time duration between the start of a request and end of that same request. Clock synchronization which is used to synchronize time with a time server by client process is called Cristian's algorithm. Low latency networks go well with these algorithms where round-trip time is short in context of accuracy.

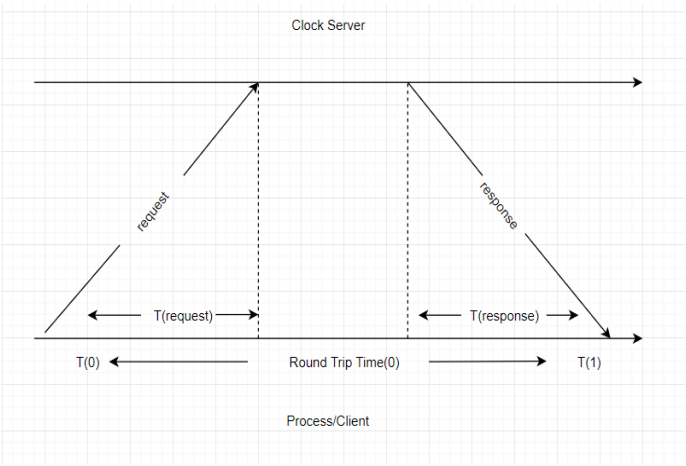

Fig -2: Demonstration of Cristian's algorithm

3.1 Algorithm

The process on the clock machine sends a request to the clock server shown above at T(0) for getting the time at server. Hence the clock server returns the clock server time as response to the process. Then the client process receives this clock server time at T1 and hence the synchronized client clock time is calculated.

T(client) = T(server) + (T1 – T0)/2

T(client) = synchronized clock time

T(server) = server returned clock time

T1 – T0 refers to the time taken by the network and server to transfer request, process and returning to the client. Network latency T0 and T1 are almost equal. To define a minimum transfer time using which we can formulate an improved synchronization clock time, using iterative testing over the network.

Defining a minimum transfer time, we can say the server time will be generated after T0 + T(min) and T(server) will be generated before T1 – T(min), here T(min) is minimum transfer time which is the minimum value of T(request) and T(response) during iterative tests.

Synchronization error = [-((T1 – T0)/2-T(min)), ((T1 – T0)/2 – T(min))]

If T(request) and T(response) differ by a considerable amount of time, substitute T(min) by T(min1) and T(min2), where T(min1) is the minimum observed request time and T(min2) refers to the minimum observed response time over the network.

Synchronized clock time:

T(client) = T(server) + (T1 – T0)/2 + (T(min2) – T(min1))/2

Hence just by keeping response and request time as separate time latencies, we can improve clock synchronization time and eventually decrease the total synchronization error.

4. Berkeley's Algorithm

This clock synchronization technique used in distributed systems. In this algorithm each machine node in the network either do not have an accurate time source or an UTC server.

A node is taken as the master node from the collection of nodes. This node acts as the main node of the network and acts as the master and the rest acts as the slaves. The master node in this case is taken by a leader election algorithm. The master node fetches the clock time by periodically pinging slave nodes using the help of Christian's algorithm.

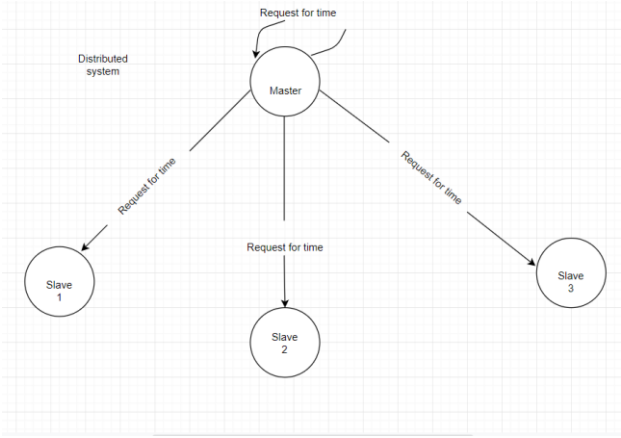

Fig –1: Sending request to salve nodes

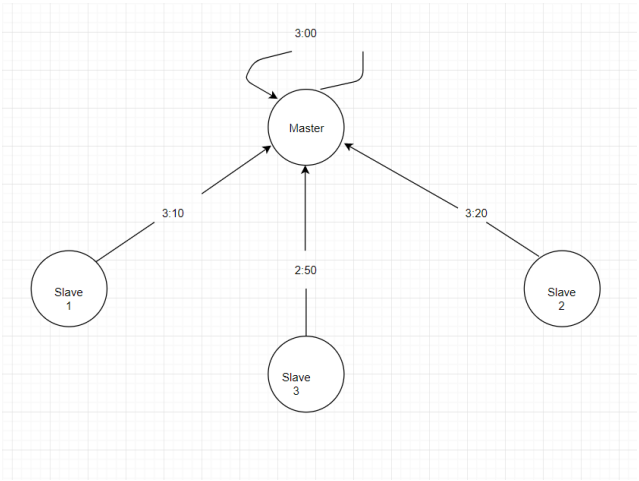

Fig –1: Slave nodes send back time from their system clock

Average time difference between all the clock times received and the master's system clock is calculated by the master node. This average time is added to current time in the master's clock and hence broadcasted over the network.

4.1 Pseudocode

```
#Receiving time from all slave nodes

repeat_for_all_slaves:

time_at_slave_node = receive_time_at_slave()

#Calculating time difference

time_difference = time_at_master_node - time_at_slave_node
```

5. PCS algorithm

This algorithm is based on TTP i.e., time transmission protocol. Each node in this algorithm uses time transmission protocol to transfer the time on its clock to a target node. TTP uses timestamped messages for the transmission of messages to the target node. The time stamps on the messages and as well as the message delay statics are the factors that help in estimating the time on the transmitting clocks node by the target node.

When a node S wants to transfer its time on its clock to a target node M, it sends n synchronization messages to target node. The $(i)^{th}$ message msg(i) sent at time T(i) on S clock is = "Time is T(i)." One of the main parameters of TTP is the separation between successive messages. The target node M records time R according to what time it receives the message on its own clock. After all the n messages from S has reaches M, it calculates the current time on S's clock.

$$T(est) = R(n) - R1(n) + T1(n) + d1$$

$$T1(n) = 1/n(\text{sum of all } T(i))$$

$$R1(n) = 1/n(\text{sum of all } R(i))$$

d1 = estimated value of message delay

To derive the clock synchronization algorithm PCS, the TTP can be incorporated into a master slave scheme. One node will be taken as the master node and the rest are synchronized to the master node as slaves.

Here also the master node communicates the time on its clock with the slave node with the help of TTP. The slave estimates the time on master clock in the same way as mentioned previously. It finds out the difference between estimated and the current time on its own clock. Then it adjusts its time to match it up with that of the master clock time. This synchronization procedure is periodically repeated. The duration of the interval between successive repetitions of synchronization procedure is called resynchronization interval (R(sync)).

The code for master nodes has 2 timers in it. The first one is resynchronization timer used for the resynchronizing at each interval. The second one is message timer used to separate successive messages by a specified interval. The interval is chosen large for independence of synchronization message delays.

If there are many slaves, the master makes sure that each slave's time is synchronized with its time. This eventually makes the clock time of the slaves are mutually synchronized. The skew between any 2 clocks is less than or equal to twice the maximum skew between master's clock and any of the slaves' clock.

The master needs to transmit only a single set of messages per synchronization interval if the system is based on any broadcast network such as ethernet, even if there are multiple slaves. Hence no of messages will be independent of the nodes in the system.

5.1 Clock synchronization Process at the Master node

```
PROCESS Master(% W, R(synch)

/* n = Number of messages; W = Interval between messages, R(synch) = R(synch) interval */

BEGIN

Set the resynch. timer to time-out immediately;

LOOP FOREVER

Wait for a time-out event to occur;

IF time-out went is due to the resynch. timer

THEN
```

ENDIF

Set the resynch. timer to time-out after R(synch) seconds;

ENDIF

Transmit time stamped message to slave;

IF fewer than n messages have been transmitted in the current resynch. interval

THEN

Set the message timer to time-out after W seconds;

ENDIF

END LOOP

End

Clock Synchronization process at slave node:

PROCESS Slave(n)

BEGIN

I <- 0 /* initialize message sequence number */,

LOOP FOREVER

Wait for the arrival of a synch message from the master;

Record the time-stamp on the message m T(i);

Record time of receipt of the message in R(i);

IF (1 = n)

THEN /* compute estimate */

T(est) <- R(n), -1/n (R(i)) + 1/n(T(i)) + d;

Adjust clock based on the estimate;

i <- 0. /* rest sequence number */

ENDIF

END LOOP

END

#Average time difference calculation

average_time_difference = sum(all_time_differences) / number_of_slaves

synchronized_time = current_master_time + average_time_difference

6. CONCLUSIONS

In conclusion, we have successfully studied the implementations of Lamport's algorithm, Cristian's algorithm, Berkeley's algorithm and PCS algorithm. The various approaches and logics have been analyzed. The importance of synchronization had been understood and discussed. With the rise of Distributed Systems in today's world, there are more up and coming algorithms and protocols to maintain the synchronization in Distributed Systems. For further work, we hope to come up with our own efficient and accepted algorithm for keeping the clocks of each node in sync.

ACKNOWLEDGEMENT

First and foremost, we would like to thank God. Our sincere gratitude goes to our dear VIT University founder Dr, G. Vishwanathan, without whom we would not have this opportunity. We cannot express enough thanks to the Computer Science department at VIT Chennai, especially Dr. Amit Kumar Tyagi – for giving us the wonderful opportunity of conducting our research. It was with his guidance and motivation that we have successfully completed this paper. Our completion of this project would not have been possible without the constant love and support from our family and friends.

REFERENCES

[1] Van Steen, Maarten, Guillaume Pierre, and Spyros Voulgaris. "Challenges in very large distributed systems." Journal of Internet Services and Applications 3.1 (2012): 59-66.

[2] Lamport, Leslie. "A fast mutual exclusion algorithm." ACM Transactions on Computer Systems (TOCS) 5.1 (1987): 1-11.

[3] Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." Concurrency: the Works of Leslie Lamport. 2019. 179-196.

[4] Gusella, Riccardo, and Stefano Zatti. "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD." IEEE transactions on Software Engineering 15.7 (1989): 847-853.

[5] Cristian, Flaviu. "Probabilistic clock synchronization." Distributed computing 3.3 (1989): 146-158.

[6] Holger Karl; Andreas Willig, "Time Synchronization," in Protocols and Architectures for Wireless Sensor Networks, Wiley, 2005, pp.201-229, doi: 10.1002/0470095121.ch8.

[7] Williams, R. N. (2018, May 8). What is the Best Source of UTC Time? Galleon Systems. http://www.galsys.co.uk/news/what-is-the-best-source-of-utc-time/.

[8] Rollins, S. (2008, January 15). Time Synchronization. http://www.cs.usfca.edu/~srollins/courses/cs686-f08/web/notes/timesync.html.

[9] Rollins, S. (2018, January 15). Time and Global States. http://www.cs.usfca.edu/~srollins/courses/cs682-s08/web/notes/timeandstates.html.

[10] UKEssays. (November 2018). Importance Of Time In Distributed Systems. Retrieved from https://www.ukessays.com/essays/philosophy/importance-of-time-in-distributed-systems-philosophy-essay.php?vref=1

[11] Honour, J. (2020, March 30). Distributed Systems: Physical, Logical, and Vector Clocks. Medium. https://levelup.gitconnected.com/distributed-systems-physical-logical-and-vector-clocks-7ca989f5f780.