International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459,ISO 9001:2008 Certified Journal, Volume 3, Issue 8, August 2013)

Handling Clock synchronization Anomalies in Distributed System

Shripad Biradar[1], Santosh Durugkar[2], Subhash Patil[3]
[1]*RMD Sinhgad School Of Engineering (RMDSSOE) Warje*
[2,3]*Late G.N.Sapkal College ofEngineering, Nasik, University of Pune*

Abstract:

Distributed system:

A distributed system is software systems in which components located on networked computers communicate and coordinate their actions by passing messages.[1] The components interact with each other in order to achieve a common goal.

In this paper we have suggested a solution which will overcome the time required to process the request which are remain in waiting queue.

Instead of sending and Receiving Request and Response messages to each and every process, all the processes that are involved into the association directly send a request message to a critical section.

It will reduce the time required to broadcast the message to each other. Instead of this at critical section side we can prepare the waiting queue which will hold all the requesting processes and will allow them one by one .

Keywords: **distributed system , clock synchronization, ricarta and agrawal's algorithm, process, logical clock , virtual clock.**

I. Introduction

1.1 Introduction

There are many alternatives for the message passing mechanism, including RPC-like connectors and message queues. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.

An important goal and challenge of distributed systems is location transparency. Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

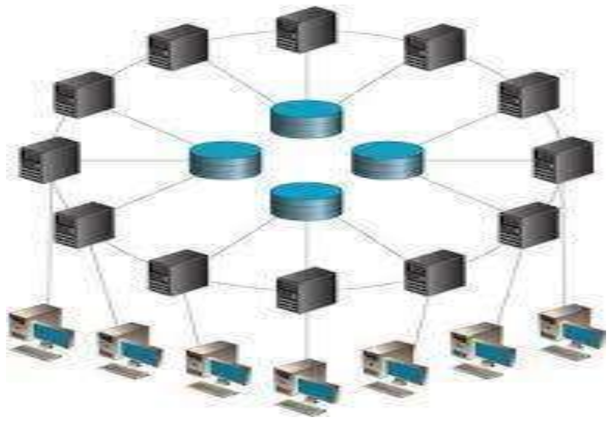

Fig. 1.1 Distributed system structure

1.2 Clock synchronization

It is a problem from computer science and engineering which deals with the idea that internal clocks of several computers may differ. Even when initiallyset accurately, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates.

There are several problems that occur as a repercussion of clock rate differences and several solutions, some being more appropriate than others in certain contexts.

Fig 1.2 Synchronization of clocks

Clock synchronization deals with understanding the temporal ordering of events produced by concurrent processes. It is useful for synchronizing senders and receivers of messages, controlling joint activity, and the serializing concurrent access to shared objects. The goal is that multiple unrelated processes running on different machines should be in agreement with and be able to make consistent decisions about the ordering of events in a system.

Clock synchronization is one of the most basic building blocks for many applications in a Distributed system. Synchronized clocks are interestingly important because they can be used to improve performance of a distributed system. The purpose of clock synchronization is to provide the constituent parts of a distributed system with a common notion of time. There are several algorithms for maintaining clock synchrony in a distributed multiprocessor system where each processor has its own clock. In this paper we consider the problem of clock synchronization with bounded clock drift. We propose a clock synchronization algorithm which does a two level synchronization to synchronize the local clocks of the nodes and also it exhibits fault tolerant behavior. Our approach should be for combining external clock synchronization and internal clock synchronization.

1.3 Cristian's algorithm

Cristian's algorithm relies on the existence of a time server. The time server maintains its clock by using a radio clock or other accurate time source, then all other computers in the system stay synchronized with it. A time client will maintain its clock by making a procedure call to the time server. Variations of this algorithm make more precise time calculations by factoring in network radio propagation time.

1.4 Berkeley algorithm

A mutual network synchronizationprotocol and algorithm that allows for use-selectable policy control in the design of the time synchronization and evidence model. NTP supports single inline and meshed operating models in which a clearly defined master source of time is used ones in which no penultimate master or reference clocks are needed.

In NTP service topologies based on peering, all clocks equally participate in the synchronization of the network by exchanging their timestamps using regular beacon packets.

In addition NTP supports a unicast type time transfer which provides a higher level of security.

A **logical clock** is a mechanism for capturing chronological and causal relationships in a distributed system.

We can construct a logical clock algorithm that will assign logical times to all the events in the system in a way that is consistent with the *happens before* relation, as follows.

- Each process keeps an integer, initially 0, that represents its internal logical clock.
- Whenever a process takes a local step, it increments its logical time by 1, and the incremented time is considered to be the time of the local event.
- Whenever a process sends a message (send event), it increments its logical time by 1, and sends that new time with the message. This time is considered to be the logical time of the send event
- Whenever a process receives a message (receive event), it first compares its own logical clock time to the logical time sent with the message, and sets its own logical clock to be the maximum of the two times. Then, it increments its logical time by 1, and the incremented time is considered to be the time of the receive event.

II. LAMPORT TIMESTAMPS

2.1 Introduction to Lamport timestamps

The algorithm of **Lamporttimestamps** is a simple algorithm used todetermine the order of events in a distributed computer system.

Fig 2.1 Time Stamp

As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for the more advanced vector clock method.

Distributed algorithms such as resource synchronization often depend on some method of ordering events to function. For example, consider a system with two processes and a disk.

The processes send messages to each other, and also send messages to the disk requesting access. The disk grants access in the order the messages were sent. Now, imagine process 1 sends a message to the disk asking for access to write, and then sends a message to process 2 asking it to read. Process 2 receives the message, and as a result sends its own message to the disk.

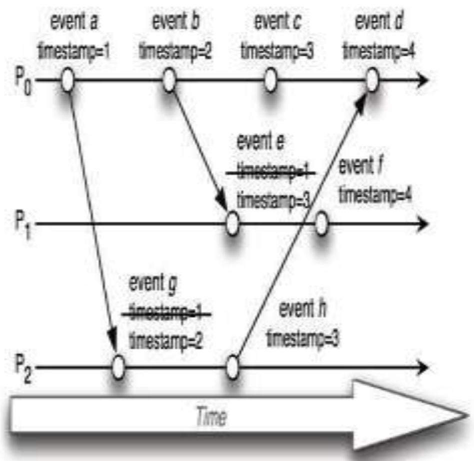

Fig 2.2 lamport's Timestamp

2.2 Ricart-Agrawala Algorithm

The **Ricart-Agrawala Algorithm** is an algorithm for mutual exclusion on a distributed system. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm, by removing the need for $\mathit{release}$messages. It was developed by Glenn Ricart and Ashok Agrawala.

Algorithm

Requesting Site:

Sends a message to all sites. This message includes the site's name, and the current timestamp of the system according to its logical clock (*which is assumed to besynchronized with the other sites*)

Receiving Site:

- Upon reception of a request message, immediately send a time stampedreply message if and only if:
  - the receiving process is not currentlyinterested in the critical section OR
  - the receiving process has a lowerpriority (usually this means having a later timestamp)
- Otherwise, the receiving process will defer the reply message. This means that a reply will be sent only after the receiving process has finished using *the critical section itself.*

Critical Section:

- o Requesting site enters its critical section only after receiving all reply messages.
- o Upon exiting the critical section, the site sends all deferred reply Messages.

A critical section is a piece of code that only one thread can execute at a time. If multiple threads try to enter a critical section, only one can run and the others willsleep.

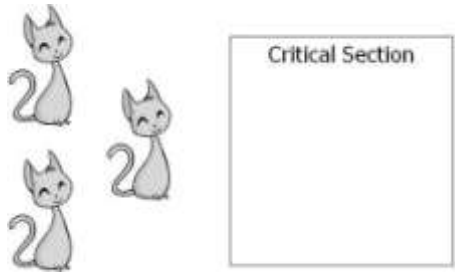

Fig 2.3 Critical section

Once the thread in the critical section exits, another thread is woken up and allowed to enter the critical section.

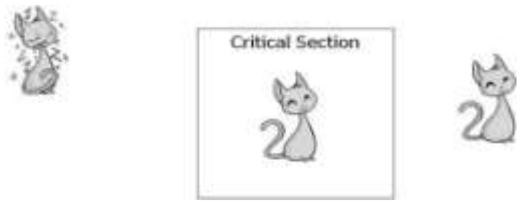

Fig. 2.4 A Process is in Critical Section

2.3 Critical point about Ricart –Agarwal algorithm:

This is not applicable in case where there are „n" no.of processes. Because failed process cannot "reply" to request message which can be interpreted as "denial of permission".

And may cause all requesting processes to wait indefinitely.

III. Proposed System For HandlingThese Anomalies By ApplyingFollowing Solutions

3.1 Solution 1:

We can propose a system in which we can keep track of "failed" processes.

Instead of delay we can maintain a list of "Live" processes.

Fig 3.1 List of Failed Processes

Meaning is that a queue can be maintained of those processes.

We can avoid the delay in such case. so that live processes will reply to requesting process so that it can get into the critical section.

3.2 Solution 2:

Instead of sending and Receiving Request and Response messages to each and every process, all the processes that are involved into the association directly send a request message to a critical section.

The critical section is to maintain a table that contains the entry of each process along with its time stamp , when new process wants to enter a critical section then its timestamp is compared with existing timestamp and based on that it will assigns a priority.

After exiting from a critical section the process need not to inform all otherprocess that areinvolved into theassociation, instead of that the processsimply exits from critical section and remaining thing is going to be take care by critical section.

The above solution will not consume much time compared to agarwala"s algorithm but the drawback is that critical section has to maintain separate algorithm to assign priority and some CPU memory to store these information.

The Algorithm is as follows

- If(Timestamp of new Process >= list of existing Time stamps)
- {
- Enter into the critical section "Process Id"
- }
- Else
- {
- Wait for the duration as per the new Priority
- }

In the Above Algorithm the Time Stamp value is not only critical section requesting time but it may also the execution time of process in critical section.

IV. Conclusion

Hence after observing the time required to reply to the message from each of the processes will be greater and also it creates the complexity if the process is failed one. It will be assumed that "denied" reply that process is giving even though it is failed.

So instead of this we can go for implementing the queue for maintaining the failed processes list.

So in implementation at the time of accessing permission from every process wecan go to only those processes which are not in the "failed processes list" i.e. the queue maintained for the same.

REFERENCES

[1] Paul Krzyzanowski, "Clock Synchronization"

[2] "Distributed Operating Systems", By Andrew Tanenbaum, © 1995 Prentice Hall.

[3] "Modern Operating Systems", By Andrew Tanenbaum, ©1992 Prentice Hall.

[4] Time, Clocks, and the Ordering of Events in a Distributed System, Leslie Lamport, Communications of the ACM, July 1978, Volume 21, Number 7, pp. 558-565.