Simulation Engine for Analysis and Comparison between Cristian's and Berkeley clock synchronization algorithms

Rahul Sehgal,
Department of Computer Science, Kent State University
rsehgal@cs.kent.edu

Abstract— **An experimental setup is proposed for comparing and analyzing clock synchronization algorithms in distributed system. Clock synchronization is required for transaction processing applications, process control applications etc. This experimental setup generates transmission delays and synchronization errors for processes and the clock synchronization algorithms try to synchronize the clocks in the system under the effect of these barriers. Two centralized clock synchronization algorithms are used for experiment - Cristian's and Berkeley clock synchronization algorithms.**

Keywords— **Clock Synchronization, Coordinator, Distributed System, Global Time, Simulation Engine, Synchronization Error, Transmission Delay, Time Server.**

I. Introduction

The two clock synchronization algorithms used for experiment in this report are Cristian's and Berkeley clock synchronization algorithms. A distributed system consists of set of processes and these processes communicate by exchanging messages. In distributed system synchronization between processes is required for various purposes, for example in transaction processing and process control operations. For processes to be synchronized and have a common view of global time, clock synchronization algorithms are applied for ensuring that physically dispersed processes have a common knowledge of time.

The clock synchronization algorithms are of following types:

1) Distributed Algorithm: NTP (Network Time Service Protocol)
2) Centralized Algorithm:
    a) Cristian's clock synchronization algorithm.
    b) Berkeley clock synchronization algorithm.

Clock synchronization algorithms can be used to synchronize clocks with respect to an external time reference (Cristian's algorithm) or to synchronize clocks among themselves. In the first approach a time server shows real time and all other clocks try to be as close to this time as possible. In the second approach (Berkeley Algorithm) real time is not available from within the system, and the goal is then to minimize the maximum difference between any two clocks. An internal clock synchronization algorithm enables a process to measure the duration of distributed activities that start on one process and terminate on another one.

In this report we will analyze and compare performances of Cristian's and Berkeley clock synchronization algorithms on a set of processes having same set of variables and instructions and are asynchronous (each process execute actions with arbitrary speeds). A simulation engine is used for generating transmission delays and synchronization errors. The synchronization algorithms try to minimize effect of these delays and errors.

The experiment is done on the basis of these parameters on varying number of processes in the system. Algorithm runs for a finite number of iterations in order to minimize the effect of delays and errors.

II. Clock Synchronization Problem

Physical clock maintained by each process differ from the reference time. This is called clock drift. This drift is caused due to physical parameters like heat and temperature. As the time passes the drift keeps on increasing. Due to this behavior of clocks there exists a clock skew in a distributed system. Two clocks are said to be synchronized if they differ from each other by a specified value. So, we need algorithms which can make these clocks to differ from each other by not more than a specified value.

III. Simulation Engine

Simulation represents key characteristics or behaviors of a selected system. Simulation Engine extends this idea, for measuring the performance of clock synchronization algorithms. It generates transmission delays and synchronization errors for clock synchronization algorithms. The algorithms work on these parameters and try to synchronize processes in a system. Simulation engine maintains a message queue for adding request messages from the processes.

A. *Concept of Time Simulation*

Time simulation does not have a predefined unit for time. It does not have a base value for starting of the time. The value of simulation time is current time in the model. Simulation time advances on the basis of change of state in the model. It never goes back, it only advances. In our experiment simulation engine maintains a global time represented by "gtime". The global time increases when the state of the system changes. Sending "request message" to the message queue, replying to messages in the message queue, transmission delay generated for "reply messages" and finding average for internal synchronization of clocks. This global time is used by algorithms for synchronizing clocks.

B. Processes, Transmission Delays and Synchronization Error

Processes are generated randomly for the system. The system is asynchronous and each process has same set of variables and instructions. They have unique identifiers. The processes communicate only by exchanging messages. In a system a process is randomly set as Coordinator (Berkeley System) or a Time Sever (Cristian System). Processes send requests to these Coordinator or Time Sever for synchronizing themselves.

Delay is generated for each process. Once the global clock reaches this value the request in delivered in the message queue and when the requests are delivered a transmission delay is calculated for each process.

Synchronization Error is calculated in Berkeley algorithm when the Coordinator sends reply about the correction to each process in the system.

IV. EXPERIMENT

A. Apparatus

In Cristian's algorithm each process sends a request and a delay is generated at each process, after which the request will be delivered to the message queue. The simulation engine removes the message from queue head, calculates a random transmission delay and sends a reply message to the destination message by adding destination identifier to the message and message delay. The calculated delay is represented as delay_at_rqst_queue in the equation.

Each process makes 30 requests to the Time Server and then averages the delay values which it gets in each "reply message" from the time server.

A difference between the current process and the global time (Time Server) is calculated. These differences are displayed in the results.

The run () method for Cristian's Algorithm:

- The messages are delivered in the increasing order of delay, to Time Server.
- Time Server computes *message_queue_delay* (states for which the message was in queue) sends a reply message to requesting process.
- Process sends 30 requests to the Time Server and gets a value for delay at request queue.
- Calculates the average on 30 delay values and calculates its local time.

In Berkeley algorithm the Simulation Engine (Coordinator) polls processes and measures the clock difference between its time and time of other process in the system. It selects a largest set of processes that do not differ from its value by more than a fixed value (in the experiment fixed value is selected as 20 milliseconds). It then averages the differences of these processes. It also calculates a synchronization error for each process clock. The Coordinator asks each process to correct its clock by a quantity equal to the difference between the average value and the previously measured difference between the clock of the Coordinator and that of a process.

The run () method for Berkeley Algorithm:

- Coordinator calculates time difference between itself and other processes in the system.
- Coordinator polls processes with a bound on difference (in experiment this value is 20 milliseconds)
- Calculates the average
- Calculates an error which represents error in approximation of other clocks in the system.
- Inform all the processes about the correction.
- Does 10 iterations of above step.

B. Assumptions

In Cristian's Algorithm, all the process send request for synchronizing its time. All processes suffer transmission delay. Each process makes 30 [1] requests to Time Server. There are no faulty processes in the system (Time Server never crashes). Processes are asynchronous and delays are generated randomly. A process sends request after waiting random amount of time.

In Berkeley Algorithm, all the processes get message in each iteration, they are asynchronous. The Coordinator never crashes. The error calculated by Coordinator is between 1-2 milliseconds and generated randomly.

C. Equations

In Cristian's Algorithm,

Current Time = gtime + (delay_at_rqst_queue)/2

gtime = global time (Time Server), (1)

delay_at_rqst_queue = averaged delay value for 30 replies.

In Berkeley Algorithm,

Average value for correction = (Total of time difference of each clock from Coordinator)/ (Total number of process polled by the coordinator). (2)

D. Results

The experiment is conducted on 5, 10, 20 and 30 processes in the system.

In Cristian algorithm each process makes 30 requests to the Time Server and then calculates an average on these delay values. In results all the processes of the system are shown and the difference between them and global time are also shown.

In Berkeley algorithm an average is calculated by the coordinator on the basis of equation (2). The graph is generated for one process by showing its difference at each iteration.

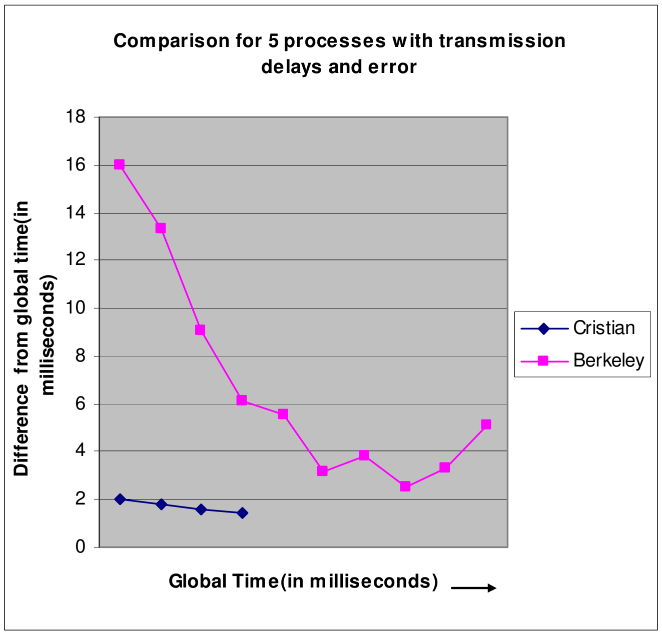

Figure 1. Comparison of Cristian and Berkeley algorithm for a system of 5 processes.

Note: For all diagrams, the Berkeley curve shows the difference curve for one process on which 10 iterations of Berkeley algorithm was performed. The value of the 4 processes on which Berkeley Algorithm were performed is shown in Table 1. Cristian curve shows the difference of all the processes in the system from the global time. Also in a system of five processes one process is Time Server, therefore only four process are shown in the curve.

Observation: In Cristian curve shows tendency to converge as the global time increases. Berkeley algorithm converge a process at a very fast rate with every iteration.

Processes	Itr 1	Itr 2	Itr 3	Itr 4
p0	16	13.2857	5.57985	3.16838
p1	17	12.2857	2.57985	6.16838
p2	15	14.2857	5.57985	6.16838
p3	17	13.2857	4.57985	4.16838

Table 1. Four iterations for a system of five processes, running Berkeley Algorithm for synchronizing their clocks.

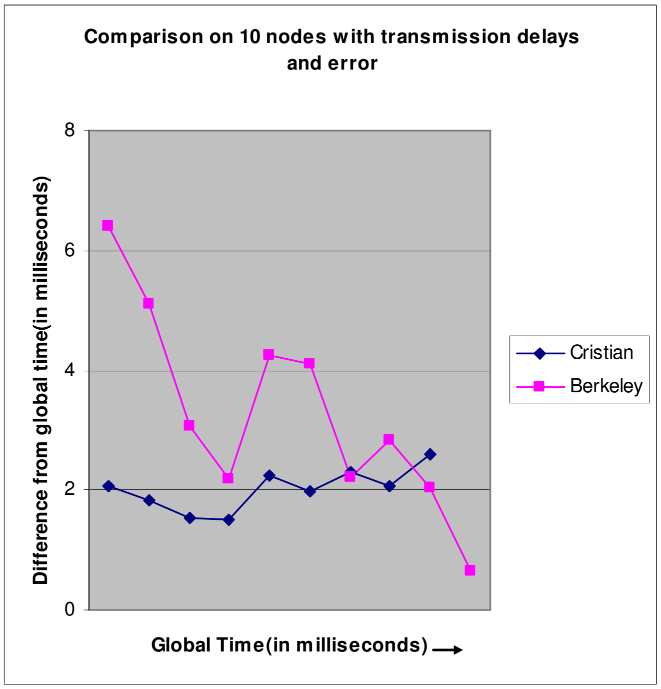

Figure 2. Comparison of Cristian and Berkeley algorithm for a system of 10 processes.

Observation: Cristian curve shows that as the number of processes increase the difference of each process from global time varies in a very small interval. In Berkeley Algorithm the process gets closer to global time value.

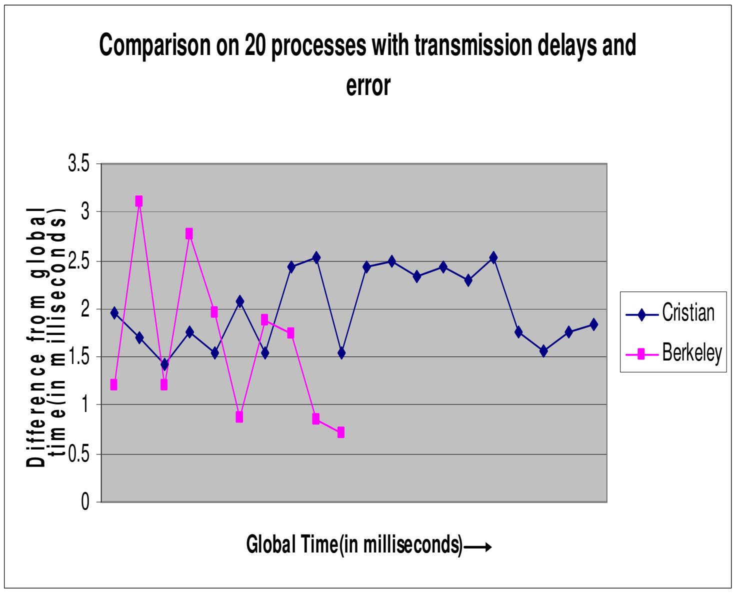

Figure 3. Comparison of Cristian and Berkeley algorithm for a system of 20 processes.

Observation: There is randomness in processes of Cristian Curve due to variable transmission delay. But this randomness is in a finite range. Berkeley curve shows that clock of the process converges closer to the global time after 10 iterations.

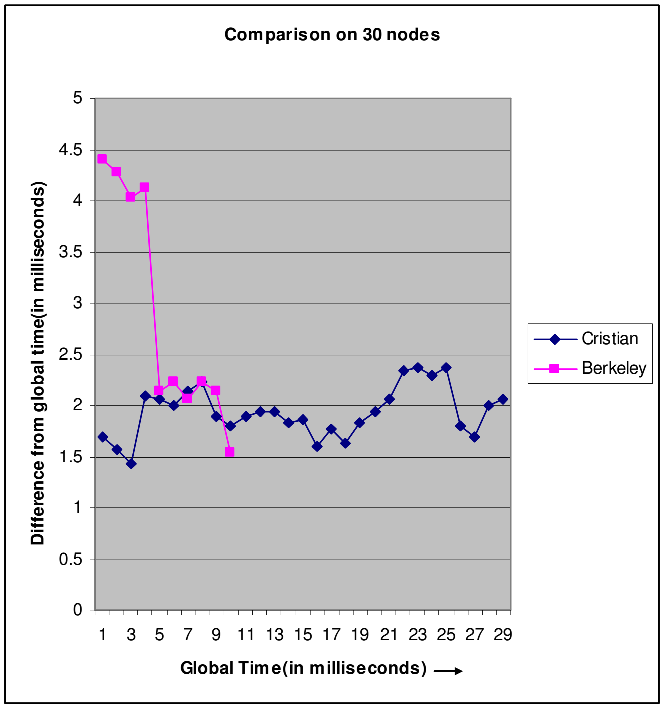

Figure 4. Comparison of Cristian and Berkeley algorithm for a system of 30 processes. Berkeley curve shows time difference of a process from global time. Cristian curve shows time difference all the processes.

Observation:

Cristian curve shows that as the number of processes increase the difference of each process from global time varies in a very small interval. Berkeley curve shows that number of iterations brings clock value closer to global time.

V. Future Work

Modify the experimental setup for faulty processes and imply a polling algorithm if the time server or coordinator crashes. Implement Cristian Algorithm over Berkeley algorithm because Berkeley algorithm shows better performance when internal synchronization is performed. But the system doesn't synchronize itself with external resources. Or improve Cristian's Algorithm because external synchronization leads to internal synchronization. That means externally synchronized processes are internally synchronized, too.
Results with clock drift and clock skew could give a better understanding of performance of these algorithms.

VI. Conclusion

Clock synchronization is required for internal and external synchronization of clocks for various transaction processes and process controls. A more efficient algorithm will lead to a better convergence.

References

[1] F. Cristian Probabilistic clock synchronization. In *Distributed Computing,* volume 3, pages 146-158. Springer Verlag, 1989
[2] R. Gusella and S. Zatti, "TEMPO-A network time controller for a distributed Berkeley UNIX system."IEEE Distributed Processing Tech. Comm. Newslett., vol. 6, no. S1-2, pp. 7-15, June 1984.
[3] J.Y. Halpern et al., "Fault-Tolerant Clock Synchronization,"Proc. Third Ann. ACM Symp. Principles of Distributed Computing, ACM, New York, 1984, pp. 89-102.
[4] T. Clouser, R. Thomas, M. Nesterenko "Emuli: Emulated Stimuli for Wireless Sensor Network Experimentation", technical report TR-KSU-CS-2007-04, Kent State Univesity .