CSCI537
Introduction to Distributed Computing

Report on

Lamport's Logical Clock A1

Dept. of Computer & Information Science,
IUPUI

Name: Keyur Kirti Mehta

Contents

1. Introduction

The assignment is to learn the concept of the clock consistency in distributed system. Due the absence of the global clock, it will be difficult to track the events between distributed nodes. So, this assignment will simulate the distributed nodes and maintain the correct event occurrence using Lamport's logical clock concept. The logically distributed nodes will inter communicate with each other as events. Also analyze drift and synchronization of the logical clock values due to change in the randomness of event occurrence.

2. Interaction Model

The simulation is implemented as distributed system consisting of 4 Process Object (PO) and 1 Master Object (MO). The PO and MO are implemented by using Java Thread concept. Each PO and MO have its own message queue. Each process will contain the reference of all the processes. The process who wants to communicate with any other process will put the message in the destination process's queue. And the destination process will take the message out of the queue to receive the same. This send and receive will acts as independent events for the processes. Apart from these events, processes will have their internal events as well.

Each process will have its own logical clock, which will keep the track of event occurrences. The description the event and logical clock consistency will be explained in later section.

Advantage:

The communication between the processes will efficient as each process have its own separate message queues. Failure of message transmission will be minimum.

Disadvantage:

If more number of processes are involved in the simulation then each processes have to maintain long list of references which would be inefficient. This model is asynchronous model which means it does not receive the acknowledgement from the receiver.

3. Failure Model

The simulation implemented the arbitrary failure (Byzantine failure). The processes when receives the message from the master object, can decide to ignore the offset value or readjust its logical clock. This will be considered based on random event. Not every time process will follow or reject the message. Because of this its arbitrary failure.

Also the system handles any interrupt exception caused during the transmission process. This is implemented using the try catch block of Java.

Advantage:

Processes runs in distributed manner. So, failure of one process won't affect the execution of the other processes.

Disadvantage:

Delivery of the messages are not acknowledged. So, processes won't be able to confirm that the master received the message successfully and vice versa. Also security mechanism is not provided in the simulation, so the content of messages are not attack proof.

4. Implementation

The simulation is implemented using Java Thread communication. Four Process Objects are created and 1 Master object thread is created. Each thread has the reference of all other threads for message communication. The entire simulation will be carried out for N iterations with each iteration have M events. This event can ne internal, send or receive event. Each event will increment the logical clock of the process.

Process object will send the logical clock value once M number of events are executed by the process. Along with this randomly it will send the logical clock value to any of the remaining process to adjust its own value. Or else it will carry out receive or internal event. This will be based on the random number generation. On receiving the message, process can randomly decide not to adjust its clock value (Byzantine failure). Internal event will be thread sleep event. Sleep time, logical clock time incremental will be depend on the random value generated earlier. Once it receives, the message from MO, one iteration will be complete.

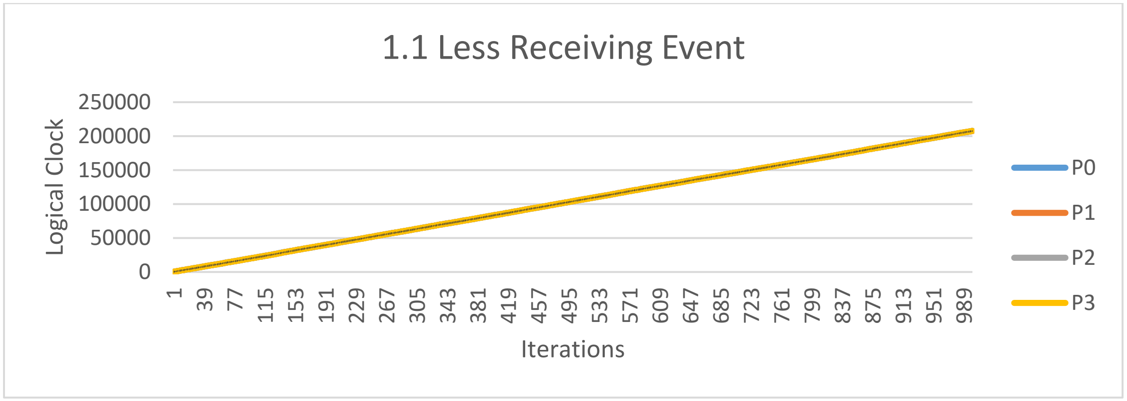
Master Object will keep checking its queue for the logical clock value from other POs. Once it receive the message it will store the thread id along with its clock value. Once MO receives all 4 value from all POs it will calculate the average and then correct his own clock. Also sends the offset calculated from the correct clock value to corresponding POs. It will store the logical clock value of each processes in the text file for future analysis purpose. MO will also carry out his own internal event on random basis.

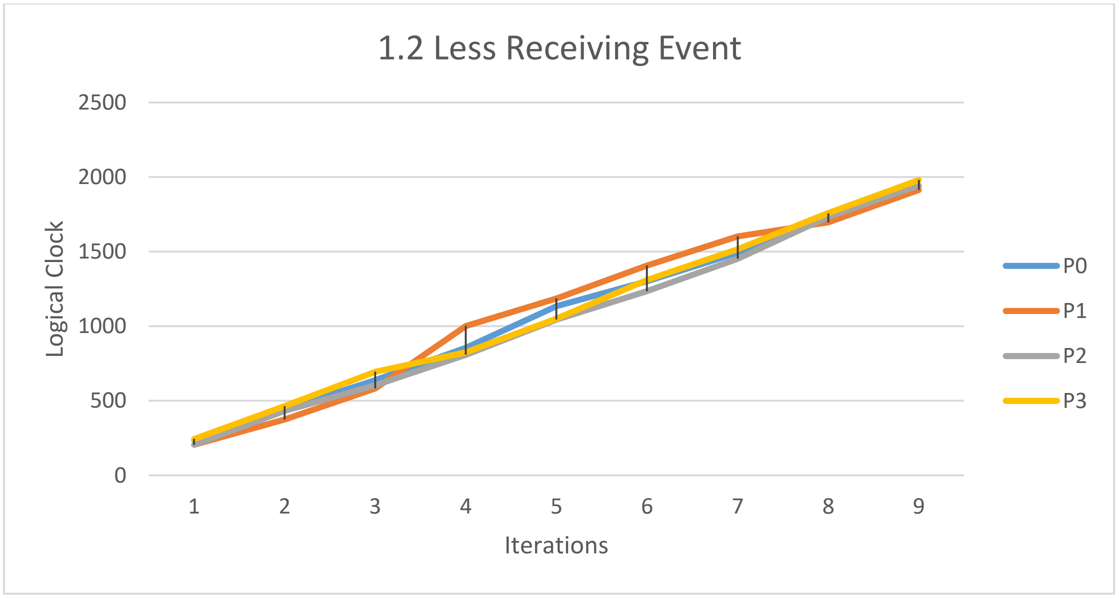
5. Result Analysis

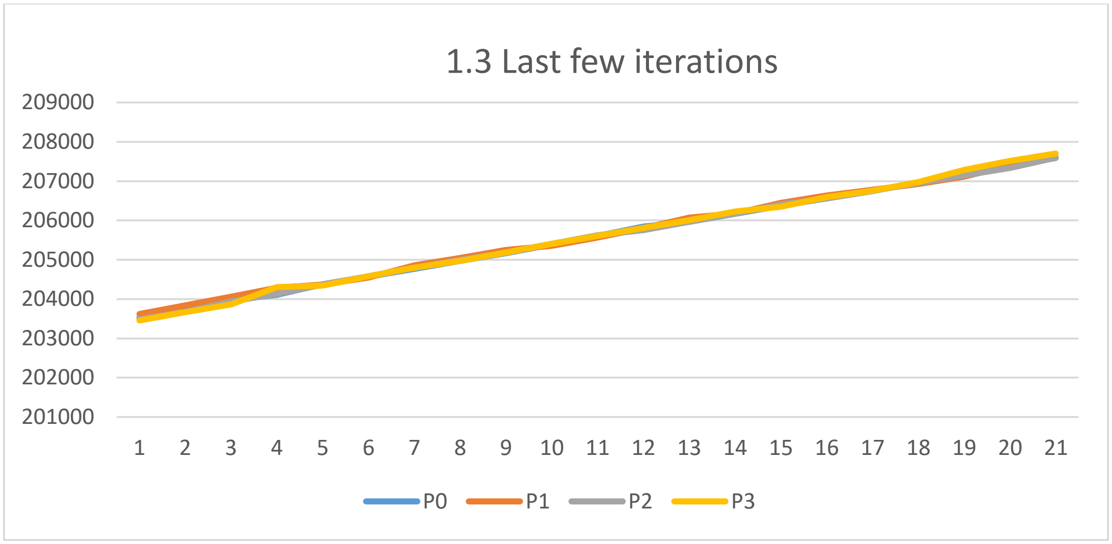
The simulation was carried out with different number of iterations and probability values. For each of the scenarios the logical clock of the PO drifts at the beginning of the scenarios and at end of the simulation they tend to move towards each other i.e. logical clock sync for all the processes.

1. The probability difference of the events was large between receive event and other events. The number of event per iteration is also kept high (t=70) with large number of iterations (n=1000). This means the correction of logical clock will occur after the long interval. The result of simulation with these parameter shows clocks drift apart from other processes clock.

   The result of all processes logical clock value is analyzed using graph with number of iteration on X-axis and logical clock value on Y-axis.
   The graph (1.1) shows straight line of all processes but this is due to large number of iterations. The graph (1.2) shows only few initial iterations which clearly shows the drift of the logical clock values and later on all the clocks sync to each other at the end of the simulation as shown in graph 1.3

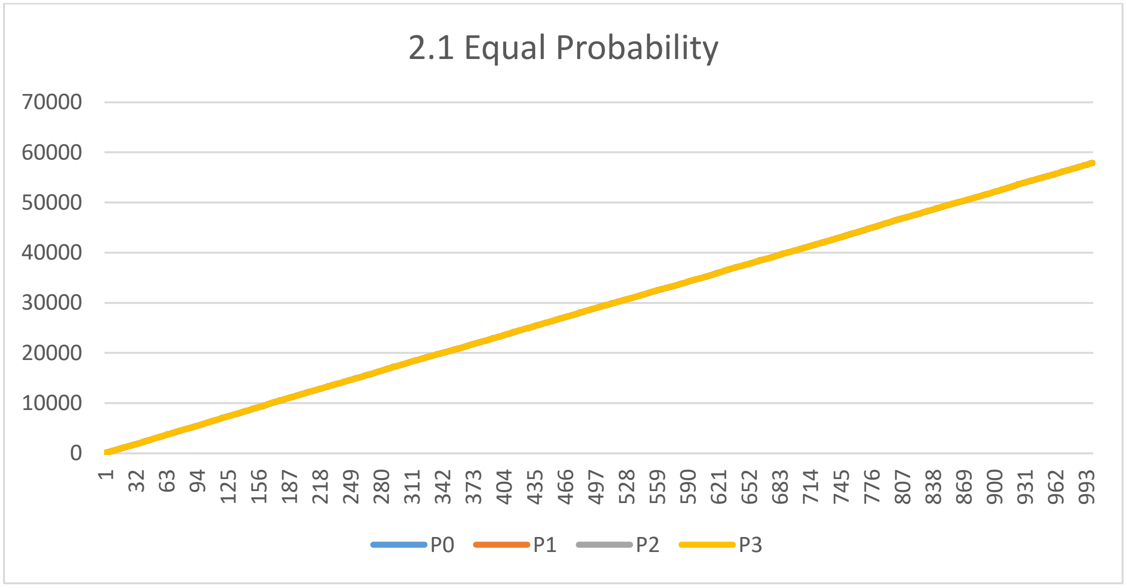


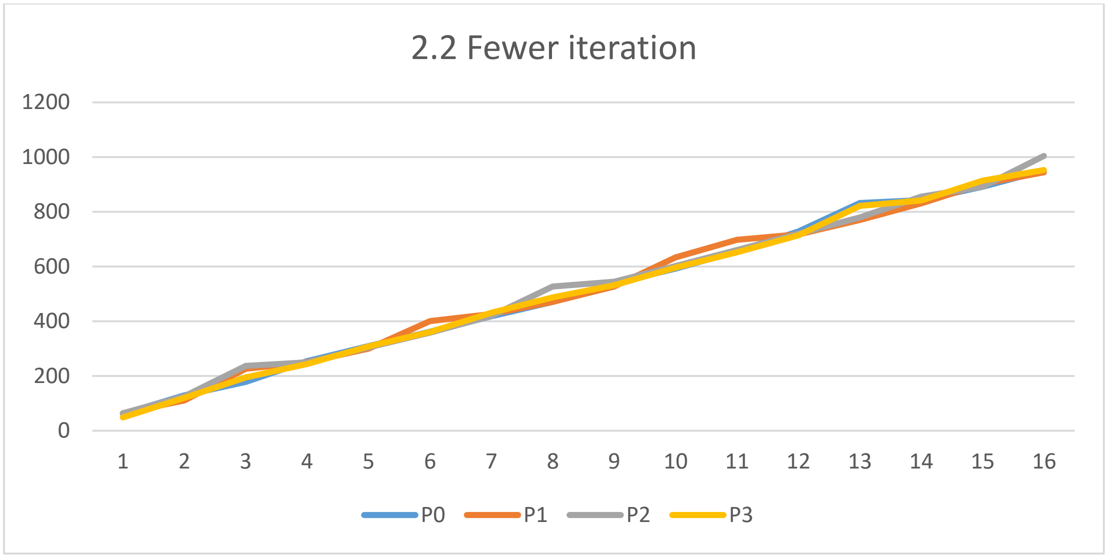




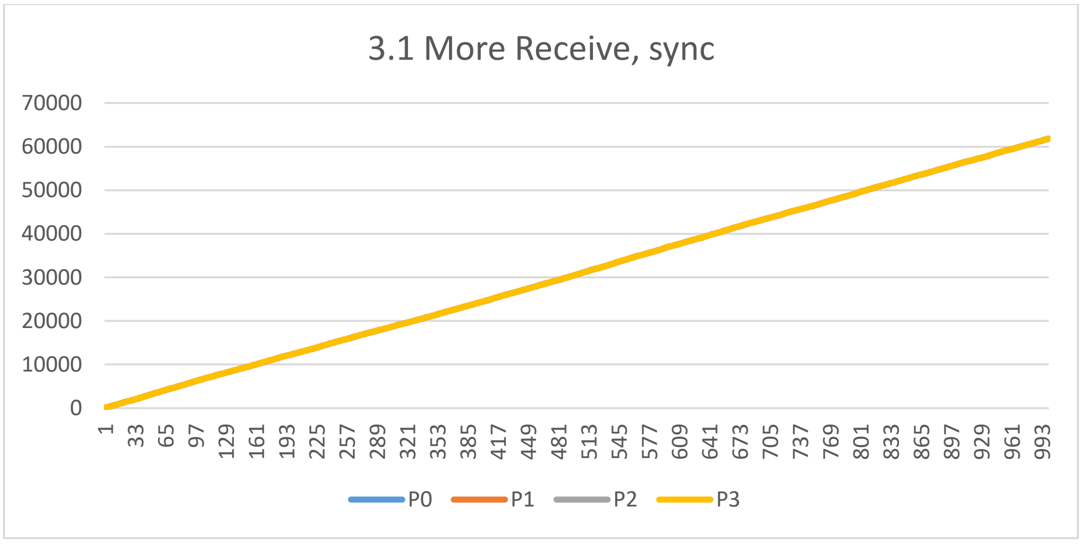
2. In this analysis, we keep the probability of all events to be equal. Send, receive and internal will take place at regular interval. Send will take place after each t events. The number of events per iteration is kept regular.

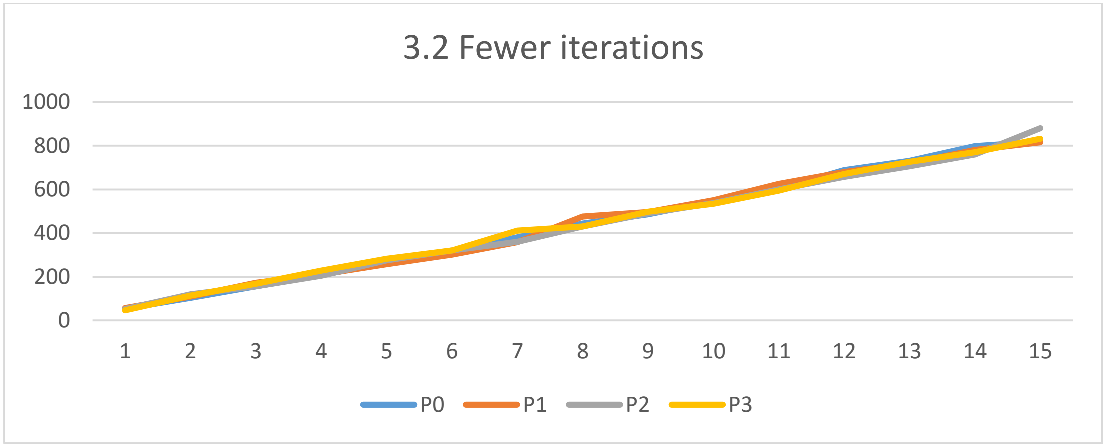
   Graph 2.1 shows the analysis when all the event have equal probability. The graph 2.2 gives better perspective of this scenario where all the clocks are almost in sync to each other.





3. In final analysis we wanted to all the processes sync their clocks at the end of the simulation. This will occur when the correction of logical clock will occur frequently and sync of clocks with other clocks are done after short duration.
   So, in this scenario, we kept the probability of the receive event high compare to other events. So, clocks were syncing more frequently. The graph 3.1 shows the entire simulation. But the graph 3.2 will show all the clocks are in sync to each other due to higher frequency of the correction in logical clocks.





6. Conclusion

The logical clock can be implemented in distributed system to keep the track of the event occurrence at different nodes in the system. This is overcome the absence of physical clock. The simulation shows the different behavior of the logical clock based on the number of iterations, events per iterations and the probability of the occurrence of the events.

References

1. Leslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System
2. https://web.mit.edu/6.005/www/fa14/classes/20-queues-locks/message-passing/
3. https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html