# Simulation of Berkley's Algorithm and Lamport Clocks

By: **Anushka H Patil**

## Aim:

To use the principles of clock consistency, drifts and inter-process communication. To create a pseudo-distributed simulation (i.e., running on a single processor) involving a master object (MO) and four process objects (PO). The MO and each PO will contain a logical clock, a concept first proposed by Lamport. The concept of the logical clocks along with the techniques, which is based on the Berkeley Algorithm, will attempt to resolve the clock consistency.

## Model:

My model consists of 4 parts:

1) Simulation
2) Process
3) Master
4) Message

**Simulation.java :**

This file is responsible for creating a pseudo- distributed simulation and runs on a single processor. It involves Master object and 4 Process objects. The threads for these objects are created in the start() function of this class file.

**Process.java:**

This file extends the Thread class and is responsible for 4 processes. The run function calls 3 other functions: sendMessage(), recieveMessage() and internalEvent(). I have used switch case to decide the function for each iteration, a random number is created at run time and it makes this decision.

1) sendMessage(): This function is responsible for sending messages. It sends an offset to Master Object when 't' counter time is passed using a masterQueue. It uses this function to send messages to other process by making use of processQueue. These processQueue and masterQueue are of type blocking queue. Every time it sends a message it increments its logical clock. The message that it puts in the Queue consist of type message.
2) receiveMessage(): This function is responsible for receiving messages. If the sender is a master object, it removes the element from the masterQueue and adjust its clock using the offset. If it receives message from other process, it uses the concept of lamport to adjust its clock. "Happened Before Relation"
3) internalEvent(): This function is independent and doesn't affect the working of any other processes

**Master.java:**

This file extends the Thread class and is responsible for master object. The run function calls 3 other functions: sendMessage(), recieveMessage() and internalEvent(). For every iteration, if conditions fulfilled all the functions run.

1) receiveMessage(): This function is responsible for receiving messages. Master object receives messages only from the process. Once, it receives the logical clock, it uses an arraylist mastetimeclock to store the logical clocks. Based on the concept of Lamport it then adjusts its logical clock.
2) sendMessage(): This function is responsible for executing the Berkley's Algorithm. It uses the mastertimeclock arraylist to maintain the logical clocks for each process. Once, all the processes send their logical time, it computes the average and later the offset. It sends offset to each process to update their logical clock. It then updates, its own logical clock. If the condition i.e. all logical clocks are obtained isn't satisfied it doesn't compute the offset. I have made use of a Boolean function check() to check whether all process have send their logical clock.

3) internalEvent(): This function is independent and doesn't affect the working of any other processes

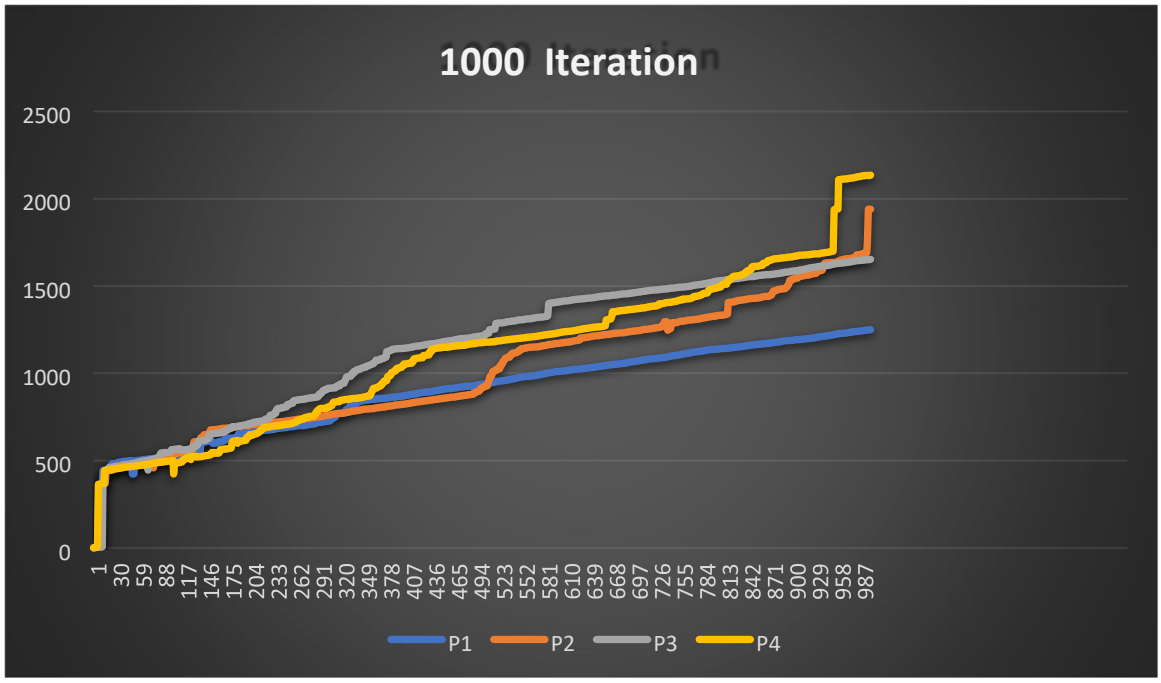
**Message.java:**

Message has two parameters: time, sender. This is used to maintain the logical clock and sender information for a message.

**Byzantine or arbitrary failures**

To exhibit this, I have made use of an if statement which checks whether the counter is divisible by a random number that is generated at run time. If this condition is satisfied it reduces the value of logical clock to half. This is how byzantine failure occurs in this model (Refer graph 2)

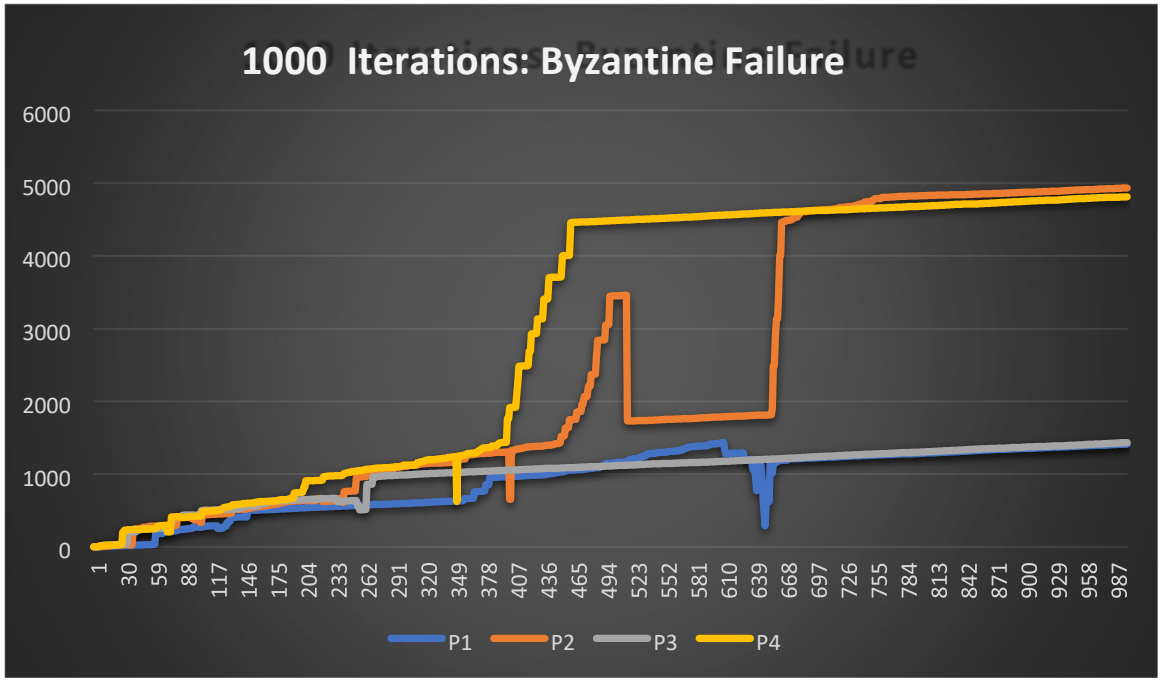
## Graphs




Graph 1: 1000 iteration

This graph (graph 1) was computed based on the results obtained from running the process for 1000 iterations. In this case, I had commented the Byzantine or arbitrary failures section to show that no failure occurred while this was in process. Here, we can see how the clock consistency and drift takes place.

The t value was set to 50, but you might notice that the master object adjusting the clocks varies. This is because for every process we randomly select the event. The probability for selecting a receive event varies for each of the 4 process. Thus, every process receives the offset at different iteration and adjust it likewise. Thus, we can notice that some processes might synchronize at same time while one take time in case of iteration 929 P2, P3 and P5 have synchronized while P1 is out of synch.




Graph 2: 1000 iteration- Byzantine failure

This graph (graph 2) was computed based on the results obtained from running the process for 1000 iterations. In this case, Byzantine or arbitrary failures occurred. Here, we can see how the clock consistency is affected and drift takes place when byzantine failure occurs in the system.

In, this case the byzantine failure occurs 5 times:

Process 1: Iteration 639

Process 2: Iteration 407, 523

Process 3: Iteration 262

Process 4: Iteration 349

In my case, byzantine failure occurs at receiving end and it reduces the logical value by half, every time the failure is caused. I have kept it half to properly show the failure in the system. In normal cases byzantine failure doesn't reduce half it could change the logical clock to anything. Thus, these failures are displayed clearly in the graph and show how they change the clock consistency.

Rest mechanism of Berkeley's algorithm is same as graph 1.

## Pros:

1) The master and process successfully communicate with each.
2) The “Happens before relationship” is acquired by using the concept by Lamport.
3) The failure doesn’t affect the working of other processes in the system. 4) All threads run properly and effectively execute their functionalities.
5) Provides a mechanism for clock consistency

## Cons:

1) It fails to detect the timeouts. This parameter will be looked in case of RMI implementation of this model.
2) The model must have a mechanism to roll back the process’s logical clock after byzantine failure.

## Future work:

1) For RMI implementation, my work involves adding encryption to the logical clock and message.
2) Furthermore, implementing practices to detect the timeout failures.

## Conclusion:

Thus, using this model, I was able to effectively implement a psedu distributed simulation. The logical clocks worked successfully based on the concept by Lamport and the techniques based on Berkley’s algorithm. This model efficiently used the principles of clock consistency, drifts and inter-process communication.