# DISTRIBUTED COMPUTING(CS5320)

# ASSIGNMENT 1

# REPORT

-SAAHIL SIROWA
-CS16BTECH11030

- **Design and Implementation**

Class Node{

**read**() //returns unsigned long long system clock in nanoseconds

**update**() //update errrorFactor

**incrementDriftFactor**() //update driftFactor

**synchronizeRequest**(ull pid){ //function to synchronize clock values

```
For i in range(K)
    For j in range(N)
        T1 = time();
        sendRequest();
        Get T2, T3 from the packet received
        T4 = time();
        Delta += (T2-T4-T1+T3)/2
        sleep()
    Delta /= (N-1);          //take mean
    Read clock value after a round of synchronization
```

}

**sendRequest**(){

```
create socket and request time
Receive response from server node and update T2, T3 accordingly
T4 = time();
```

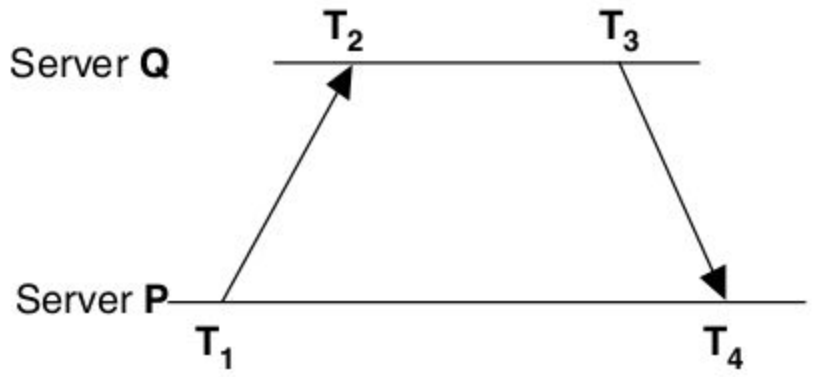
}

**receive**(){ //function to receive sync request from client nodes

```
Create socket and listen on the port
```

```
                Accept connection. System generates a new socket
                T2 = time();
                sleep();
        T3 = time()
        send(packet with info about T2 and T3)
}
```

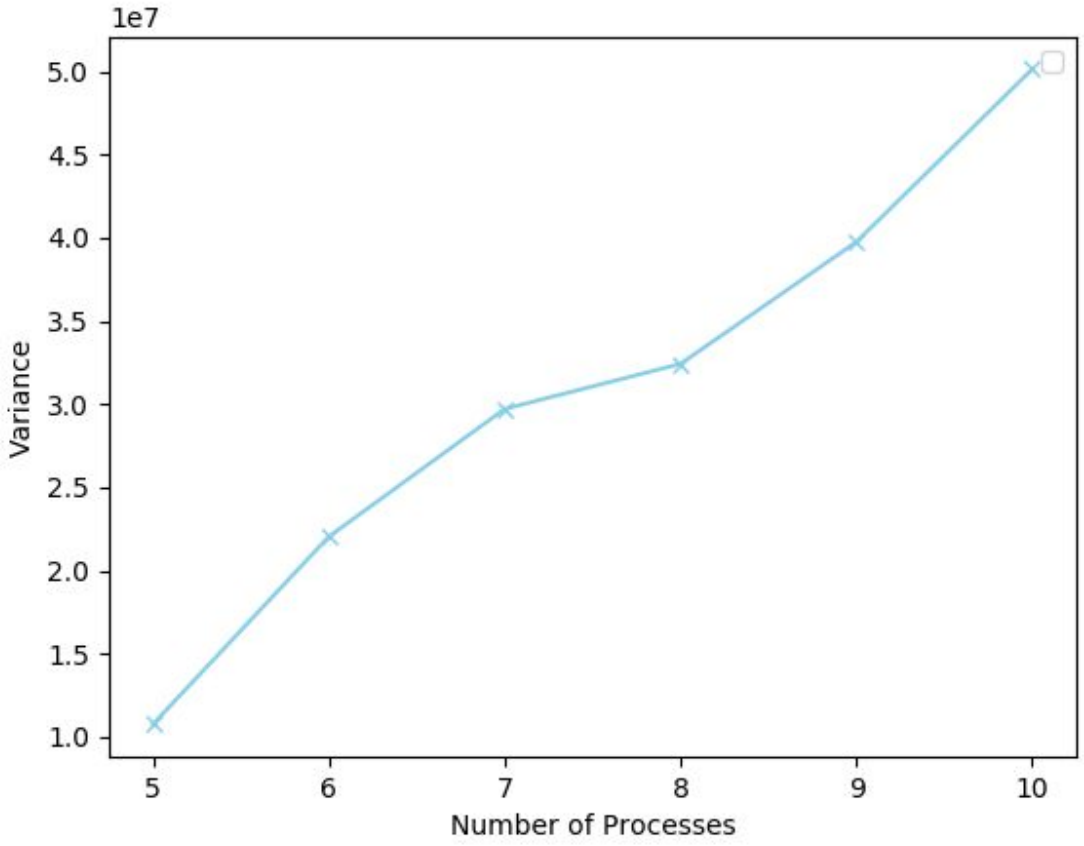
- **P2P Connection**
  Consider the exchange of a pair of messages between two time-servers as shown in Figure.Define the offset between two servers P and Q as the difference between their clock values. For each message, the sending time is stamped on the body of the message, and the receiver records its own time when the message was received. Let $T_{PQ}$ and $T_{QP}$ be the message propagation delays from P to Q and Q to P, respectively. If server Q's time is ahead of server P's time by an offset δ




$$T_2 = T_1 + T_{PQ} + \delta$$
$$T_4 = T_3 + T_{QP} - \delta$$
$$2\delta = (T_2 - T_4 - T_1 + T_3) - (T_{PQ} - T_{QP})$$
$$\delta = (T_2 - T_4 - T_1 + T_3)/2 - (T_{PQ} - T_{QP})/2$$
Assuming symmetric network delay
$$\delta = (T_2 - T_4 - T_1 + T_3)/2$$

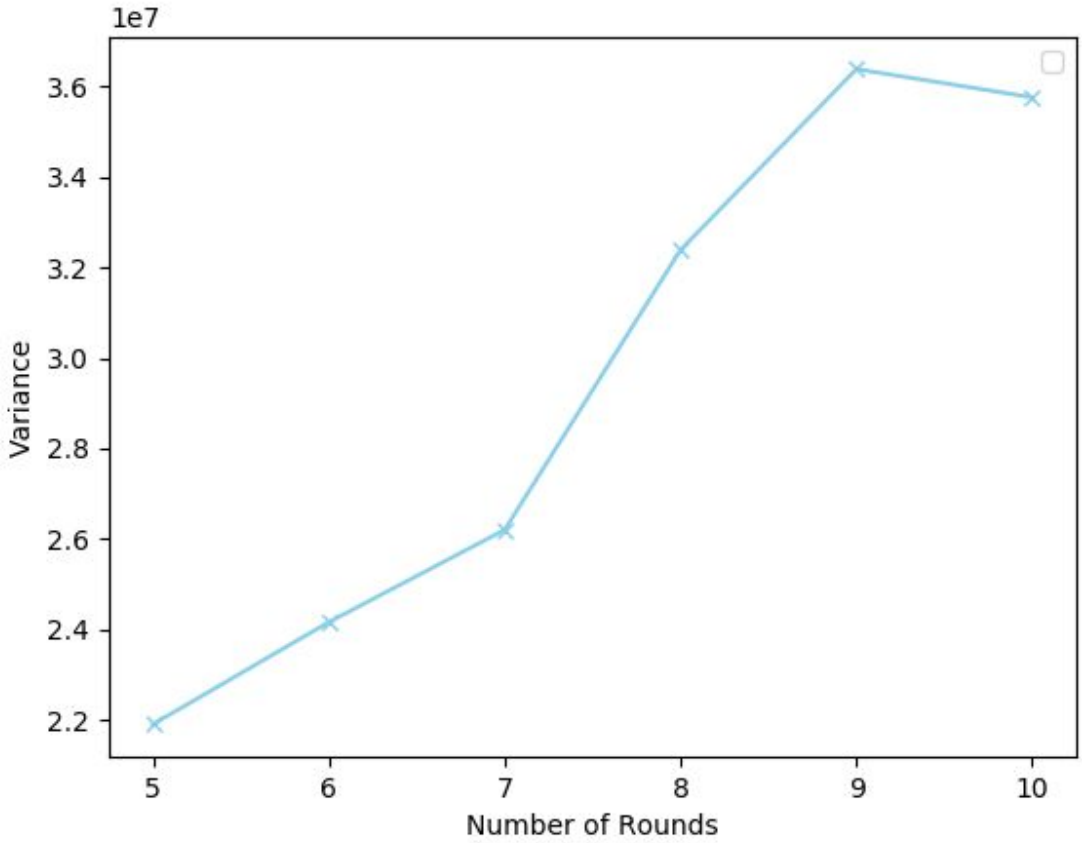
- **Testing methodology**
  For each experiment, a mean of 50 readings is reported. Each of the cross marks in the graph is a mean of 50 readings with that particular set of input parameters. A script is written for this purpose.

- **Graph of Number of Processes vs Variance after synchronization**
  Number of synchronization rounds are fixed as 10 and the number of processes varies from 5 to 10 with an increment of 1.




  **Analysis:** With increase in number of processes the number of threads spawned by program increases by 3. As the number of physical threads is fixed, increasing number of threads spawned will result in extensive contention for system resources. This results in unnatural message delays and thread scheduling issues. Due to these reasons variance is  increasing with an increasing number of threads.
  Theoretically more processes should give better synchronization results.

- **Graph of Number of Synchronization Rounds vs Variance after synchronization**
  Number of Processes are fixed as 10 and the number of synchronization rounds varies from 5 to 10 with an increment of 1.




**Analysis:** There is no fixed relation between the number of rounds and variance of time servers. Theoretically Variance of time servers is supposed to go through a cycle of convergence and divergence as rounds progresses. Same can be observed in the graph. We observe a dip in variance at 10th round as compared to 9th round. If extrapolated it will keep falling for some rounds and then rise again.