Cristian Algorithm

Subject - Distributed Algorithms
Semester - WS 2022-23
Submitted by - Aman Sahani
Bilal Raza

Contents

Introduction

Cristian Algorithm

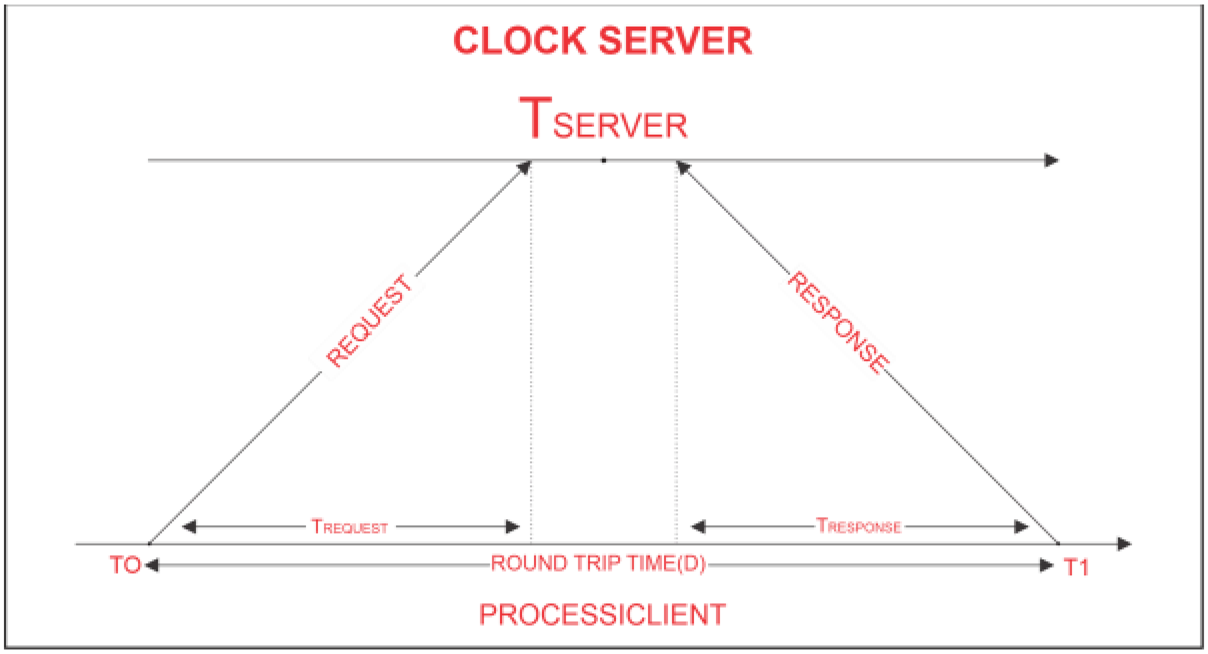
Cristian’s Algorithm is a clock synchronization algorithm used to synchronize time with a time server by client processes.
It is a method for synchronizing the time of client processes with a time server. It is most effective in networks with low latency, where **Round Trip Time** is short compared to the desired level of accuracy. However, it is not well-suited for use in distributed systems prone to redundancy, as these systems do not typically require high levels of accuracy.
The time interval between the beginning of a Request and the corresponding response's conclusion is called Round Trip Time.

Advantages and Disadvantages

1. Improve accuracy by sending multiple spaced requests and using response with smallest (T1 - T0).
2. Server failure: Use multiple synchronized time servers.
3. Cannot handle faulty time servers.

Algorithm Description

An example mimicking the operation of Cristian's algorithm is provided below:



1. The process on the client machine sends the clock server a request at time T0 for the clock time (time at the server).
2. In response to the client process request, the clock server listens and responds with clock server time.
3. The client process retrieves the response from the clock server at time T1 and uses the formula below to determine the synchronized client clock time.

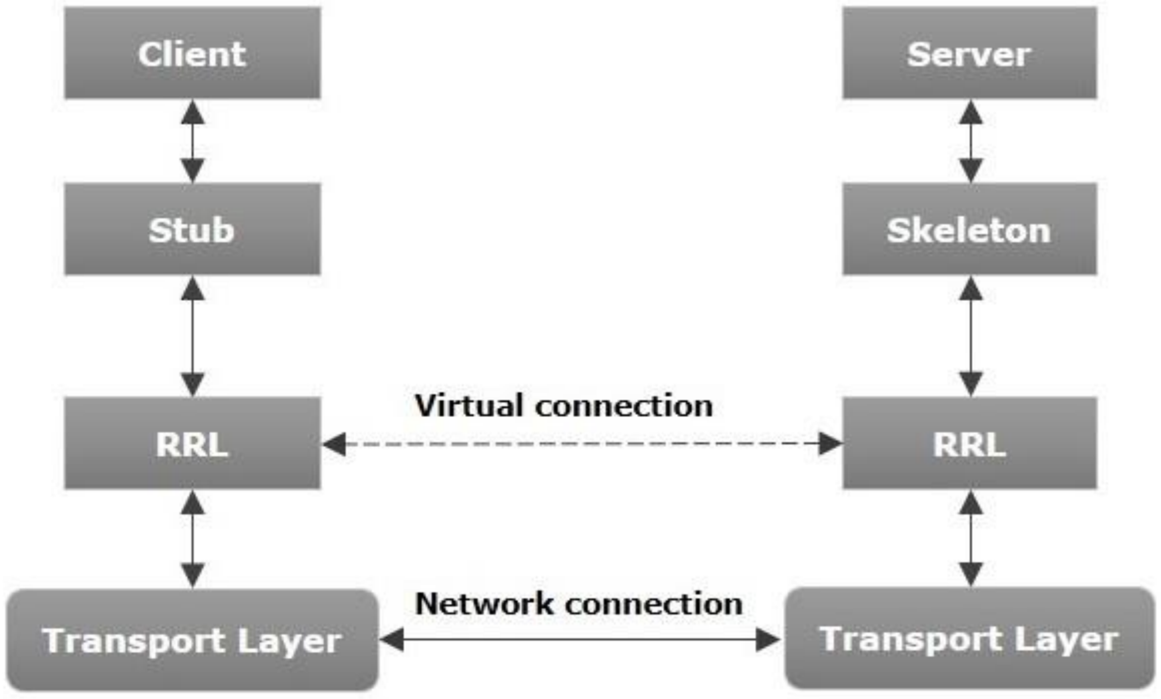
Formula: TCLIENT = TSERVER + (T1 - T0)/2.
where TCLIENT denotes the synchronized clock time, TSERVER denotes the clock time returned by the server, T0 denotes the time at which the client process sent the request, and T1 denotes the time at which the client process received the response

The difference between the client's time and the actual time is at most (T1 - T0) / 2 seconds, where T0 is the time at which the request was sent by the client and T1 is the time at which the response was received by the client. This means that the error in synchronization can be at most (T1 - T0) / 2 seconds, or within the range [-(T1 - T0) / 2, (T1 - T0) / 2].

Implementation Description

We have used Java RMI for communication between different processes within the same system.

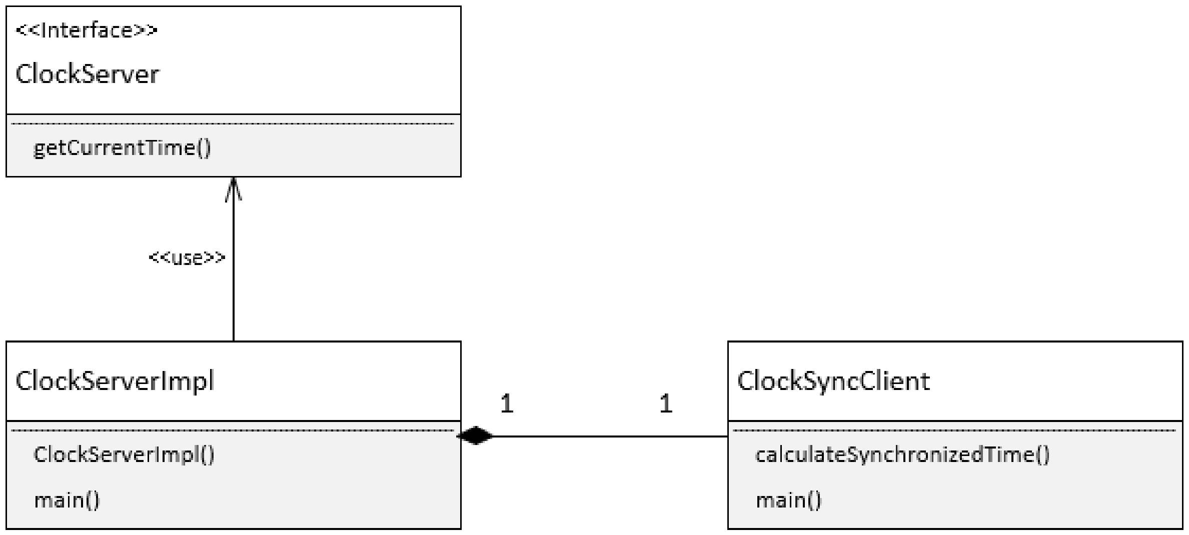
Java RMI



As per the Java RMI architecture we have created -

1. **ClockServer** Interface containing methods that needs to be implemented
2. **ClockServerImpl** class that implements the methods defined in the interface and export the method to RMI registry.
3. **ClockSyncClient** class that is the client class which request for server time.

UML Diagram



Detailed Class and Method description

ClockServer
This is an interface class that declares a method **getCurrentTime**() and throws RemoteException

```
public interface ClockServer extends Remote {
    Date getCurrentTime() throws RemoteException;
}
```

ClockServerImpl
This class extends the interface ClockServer and defines the method getCurrentTime(). The main method exports the ClockServerImpl object as a remote object and then creates a registry and binds the remote object to it. The Registry class is used to create a registry if it does not already exist.

getCurrentTime – This method returns and server time using Calendar. We have made the thread sleep for 10000 milliseconds deliberately in order to observer the algorithm with some delay.

```
public class ClockServerImpl implements ClockServer {

    public ClockServerImpl() {}
    public Date getCurrentTime() throws RemoteException {
        Calendar calendar = Calendar.getInstance();
        // using sleep to increase response time
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return calendar.getTime();
    }

    public static void main(String[] args) {
        try {
            ClockServerImpl clockServer = new ClockServerImpl();
            ClockServer stub = (ClockServer)
UnicastRemoteObject.exportObject(clockServer, 0);

            Registry registry = LocateRegistry.createRegistry(8000);
            registry.bind("ClockServer", stub);

            System.out.println("Clock server ready.");
        } catch (Exception e) {
            System.out.println("Clock server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

ClockSyncClient
This is the client class that will make a connection with the server and use the method defined at server to request the time from the server.
The main method first locate the registry using port and then connect to the server with help of lookup.

Then we first note the current system time i.e. T0 and then request for the server time (TServer) using the getCurrentTime() method defined at server, after the response from server we note the response time T1.

calculateSynchronizedTime - This methods takes request time T0, response time T1 and server time TServer to calculate the synchronized time and the error between actual time and synchronized time.
We calculate the synchronized time using the formula
TSyncTime = TSERVER + (T1 - T0)/2

We then proceed to find the error between the actual client time and the synchronized time calculated to find error in the Cristian algorithm.

```java
public class ClockSyncClient {

    public static void calculateSynchronizedTime(long requestTime, long
responseTime,Date serverTime){
        // Get the actual time on the client's clock
        Calendar calendar = Calendar.getInstance();
        Date actualTime = calendar.getTime();

        // Calculate the process delay latency i.e. (T1-T0)
        long processDelayLatency = responseTime - requestTime;

        // Synchronize the client's clock with the server's clock
        // Tclient = Tserver + (T1-T0)/2
        Calendar milliCalendar = Calendar.getInstance();
        milliCalendar.setTime(serverTime);
        milliCalendar.add(milliCalendar.MILLISECOND, (int)processDelayLatency);
        Date synchronizedTime = milliCalendar.getTime();

        // Calculate the synchronization error
        long error = actualTime.getTime() - synchronizedTime.getTime();

        //Set Date Format
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSS");

        // Print the results
        System.out.println("Request Time: " + dateFormat.format(requestTime));
        System.out.println("Response Time: " + dateFormat.format(responseTime));
        System.out.println("Process Delay Latency: " + processDelayLatency + "
milliseconds");
        System.out.println("Time returned by server: " +
dateFormat.format(serverTime));
        System.out.println("Actual clock time at client side: " +
dateFormat.format(actualTime));
        System.out.println("Synchronized process client time: " +
dateFormat.format(synchronizedTime));
        System.out.println("Synchronization error: " + error + " milliseconds");
    }

    public static void main(String[] args) throws RemoteException,
NotBoundException {
        // //Set Date Format
        // SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSS");

        // Connect to the clock server
        Registry registry = LocateRegistry.getRegistry("localhost", 8000);
        ClockServer clockServer = (ClockServer) registry.lookup("ClockServer");

        // Get the current time T0
        long requestTime = System.currentTimeMillis();

        // Request the current time from the server
        Date serverTime = clockServer.getCurrentTime();

        // Get the response Time T1
        long responseTime = System.currentTimeMillis();

        calculateSynchronizedTime(requestTime,responseTime,serverTime);


    }

}
```

Results and observations

We observed that the error between the actual time at client and the synchronized client time is usually within -10 to +10 milliseconds which is relatively low indicating that Cristian algorithm is working correctly.

Bibliography

Saini, Harshit. “Cristian's Algorithm.” *GeeksforGeeks*, 10 Nov. 2021, https://www.geeksforgeeks.org/cristians-algorithm/.

Cristian's algorithm - javatpoint. www.javatpoint.com. (n.d.). Retrieved January 4, 2023, from https://www.javatpoint.com/cristians-algorithm

Thakur, Amey *“Clock synchronization in distributed”* https://www.researchgate.net/publication/359685729_Clock_Synchronization_in_Distributed_Systems (Accessed: January 4, 2023).