Distributed Algorithms (UAI/503): Berkeley Algorithm

Eldhose Poulose

January 7, 2023

Abstract

In this report, the Berkeley Algorithm and its implementation using Java RMI technology is explained. The source files can be viewed under Github.

1 Introduction

In the Distributed Systems (DS) the nodes are communicating with each other using message passing. To achieve real time applications working in ordered manner the reference parameter used is time. Therefore synchronisation of time is essential for allocating the available resources. Synchronization in DS can be achieved by using physical clock of the node. For synchronization purpose, each node in the system needs to share their local clock time with another node in the system.

2 Berkeley Algorithm

The Berkeley Algorithm follows a master slave communication operation. There is a central computer that serves as the master or a time server. The master master periodically sends a request message to all other slaves or nodes which asks the time of the destination nodes. The master will receive the round trip time value (RTT) from the slaves, and the master will average the time values including its own clock value and readjusts its own clock accordingly. The master will also eliminate readings from faulty clocks that is values far outside the range. It does this by taking a subset of the returned times with a small variance. After this, the master will then send the amount by which each individual's clock requires adjustment to each individual clock. This value can be positive or negative. If the master fails at any point, then a new master is elected to take over and function exactly like its predecessor.

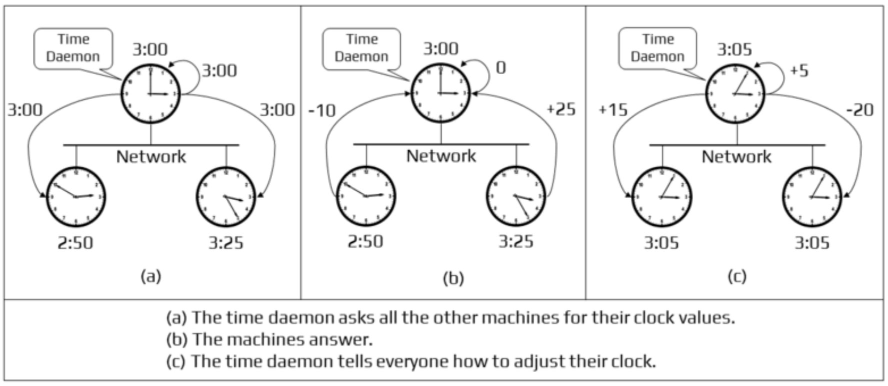


Figure 1: Berkeley Algorithm Workflow[source: see below]

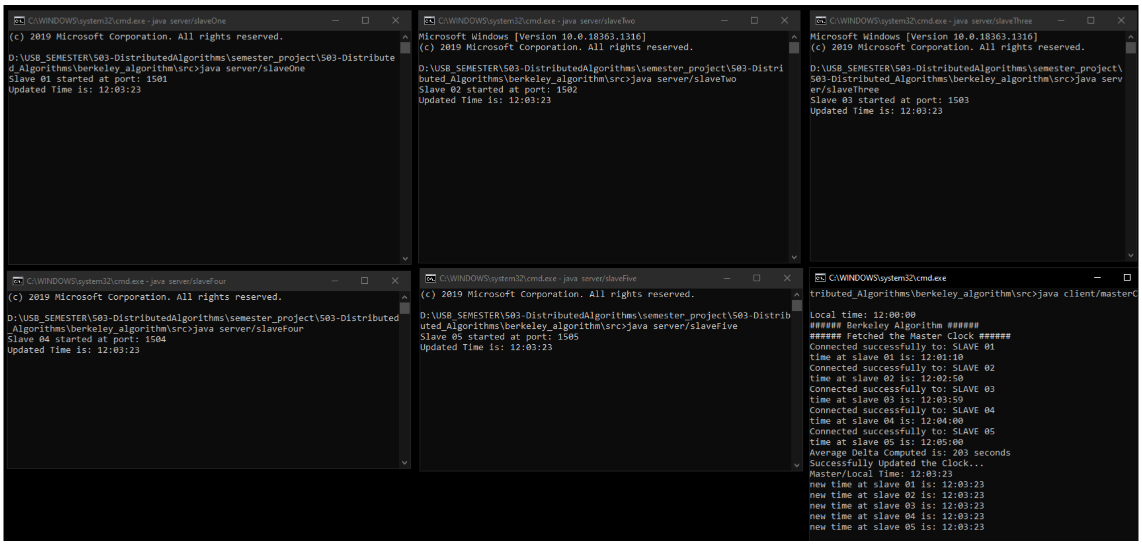
3 Methods

The basic idea in implementing this algorithm is to test with some random clock time with some slaves or servers. Here I implemented a master machine and 5 other server machines, then the clock values are fetched from the each machines and average the time difference with master clock time. Later the changes are made to the client and slaves and displayed the new clock values. After compilation first run all the slaves/server files (slaveOne, slaveTwo, slaveThree, slaveFour, slaveFive) then run master clock (masterClock).

- *client*: contains client side (master) implementation files
- *server*: contains server side (master) implementation files
- *global*: contains the global parameter used in this algorithm. For instance the port values of slaves.

4 Results

For experimenting I used the 12:00:00 at master node and for slaves I used slightly different values. The algorithm managed to computed the average and update the error value, in this case it is 203 seconds. Figure 2. The simulation result is provided in Figure 2 and the resources can be accessed from Github.


C:\WINDOWS\system32\cmd.exe - java server/slaveOne
C:\WINDOWS\system32\cmd.exe - java server/slaveTwo
C:\WINDOWS\system32\cmd.exe - java server/slaveThree
C:\WINDOWS\system32\cmd.exe - java server/slaveFour
C:\WINDOWS\system32\cmd.exe - java server/slaveFive
C:\WINDOWS\system32\cmd.exe

```
(c) 2019 Microsoft Corporation. All rights reserved.

D:\USB_SEMESTER\503-DistributedAlgorithms\semester_project\503-Distribute
d_Algorithms\berkeley_algorithm\src>java server/slaveOne
Slave 01 started at port: 1501
Updated Time is: 12:03:23
```

```
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\USB_SEMESTER\503-DistributedAlgorithms\semester_project\503-Distri
buted_Algorithms\berkeley_algorithm\src>java server/slaveTwo
Slave 02 started at port: 1502
Updated Time is: 12:03:23
```

```
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\USB_SEMESTER\503-DistributedAlgorithms\semester_project\
503-Distributed_Algorithms\berkeley_algorithm\src>java serv
er/slaveThree
Slave 03 started at port: 1503
Updated Time is: 12:03:23
```

```
(c) 2019 Microsoft Corporation. All rights reserved.

D:\USB_SEMESTER\503-DistributedAlgorithms\semester_project\503-Distributed
_Algorithms\berkeley_algorithm\src>java server/slaveFour
Slave 04 started at port: 1504
Updated Time is: 12:03:23
```

```
(c) 2019 Microsoft Corporation. All rights reserved.

D:\USB_SEMESTER\503-DistributedAlgorithms\semester_project\503-Distrib
uted_Algorithms\berkeley_algorithm\src>java server/slaveFive
Slave 05 started at port: 1505
Updated Time is: 12:03:23
```

```
tributed_Algorithms\berkeley_algorithm\src>java client/masterC

Local time: 12:00:00
###### Berkeley Algorithm ######
###### Fetched the Master Clock ######
Connected successfully to: SLAVE 01
time at slave 01 is: 12:01:10
Connected successfully to: SLAVE 02
time at slave 02 is: 12:02:50
Connected successfully to: SLAVE 03
time at slave 03 is: 12:03:59
Connected successfully to: SLAVE 04
time at slave 04 is: 12:04:00
Connected successfully to: SLAVE 05
time at slave 05 is: 12:05:00
Average Delta Computed is: 203 seconds
Successfully Updated the Clock...
Master/Local Time: 12:03:23
new time at slave 01 is: 12:03:23
new time at slave 02 is: 12:03:23
new time at slave 03 is: 12:03:23
new time at slave 04 is: 12:03:23
new time at slave 05 is: 12:03:23
```

Figure 2: Updating 5 Slaves and a master clock time using Berkeley Algorithm

A source code: clock server

```
package server;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.time.LocalTime;


public interface clockServer extends Remote{

        LocalTime getTime() throws RemoteException;
        void adjustTime(LocalTime slaveTime, long delta) throws RemoteException;

}
```

B source code: clock server Implementation

```
package server;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.time.LocalTime;
import global.berkeleyParams;

public class clockServerImplementation extends UnicastRemoteObject implements clockSe

        private LocalTime time;
        public clockServerImplementation(LocalTime time) throws RemoteException {
                this.time = time; // fetch the time
        }

        @Override
        public LocalTime getTime() throws RemoteException {
                return time;
        }

        @Override
        public void adjustTime(LocalTime slaveTime, long delta) throws RemoteExceptio
                long localTime = slaveTime.toSecondOfDay(); // fetch the time from sl
                long serverTime = this.getTime().toSecondOfDay(); // fetch the time fr
                var deltaTime = serverTime - localTime; // compute the difference in
                deltaTime = deltaTime * -1 + delta + serverTime;
                LocalTime updatedClock = LocalTime.ofSecondOfDay(deltaTime);
                System.out.println("Updated␣Time␣is:␣" + berkeleyParams.dateformatter
                this.time = updatedClock;
        }

}
```

C source code: global

```
//
package global;

import java.time.format.DateTimeFormatter;

public interface berkeleyParams {
        public final String servername01 = "localhost";
        public final int serverport01 = 1501;

        public final String servername02 = "localhost";
        public final int serverport02 = 1502;

        public final String servername03 = "localhost";
        public final int serverport03 = 1503;

        public final String servername04 = "localhost";
        public final int serverport04 = 1504;

        public final String servername05 = "localhost";
        public final int serverport05 = 1505;

        public final DateTimeFormatter dateformatter = DateTimeFormatter.ofPattern("H

}
```

D source code: main

```
// This file includes the client side implementation of the algorithm
// 5 server/ slave machines time is fetched and the time is adjusted.
// time difference and the average of them is computed and adjusted


package client;
import java.rmi.*;
import java.rmi.registry.*;
import static global.berkeleyParams.dateformatter;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.time.LocalTime;
import java.util.ArrayList;
import global.berkeleyParams;
import server.clockServer;
import server.clockServerImplementation;

//master/Client Side Implementation

public class masterClock {

        public static void main(String[] args) {
                try {
                        var slavetimes = new ArrayList<LocalTime>();

                        LocalTime masterTime = LocalTime.parse("12:00:00", berkeleyPar
                        slavetimes.add(masterTime);
                        System.out.println("Local␣time:␣" + dateformatter.format(maste

                        System.out.println("######␣Berkeley␣Algorithm␣######");
                        System.out.println("######␣Fetched␣the␣Master␣Clock␣######");

                        // Connect to slave 01 and get Time
                        Registry registry01 = LocateRegistry.getRegistry(berkeleyParan
                        clockServer ts01 = (clockServer) registry01.lookup(clockServer
                        System.out.println("Connected␣successfully␣to:␣SLAVE␣01");
                        LocalTime slaveTime01 = ts01.getTime();
                        slavetimes.add(slaveTime01);
                        System.out.println("time␣at␣slave␣01␣is:␣" + dateformatter.for

                        // Connect to slave 02 and get Time
                        Registry registry02 = LocateRegistry.getRegistry(berkeleyParan
                        clockServer ts02 = (clockServer) registry02.lookup(clockServer
                        System.out.println("Connected␣successfully␣to:␣SLAVE␣02");
                        LocalTime slaveTime02 = ts02.getTime();
                        slavetimes.add(slaveTime02);
                        System.out.println("time␣at␣slave␣02␣is:␣" + dateformatter.for


                        // Connect to slave 03 and get Time
                        Registry registry03 = LocateRegistry.getRegistry(berkeleyParan
                        clockServer ts03 = (clockServer) registry03.lookup(clockServer
                        System.out.println("Connected␣successfully␣to:␣SLAVE␣03");
                        LocalTime slaveTime03 = ts03.getTime();
                        slavetimes.add(slaveTime03);
                        System.out.println("time␣at␣slave␣03␣is:␣" + dateformatter.for

                        // Connect to slave 04 and get Time
                        Registry registry04 = LocateRegistry.getRegistry(berkeleyParan
                        clockServer ts04 = (clockServer) registry04.lookup(clockServer
                        System.out.println("Connected␣successfully␣to:␣SLAVE␣04");
                        LocalTime slaveTime04 = ts04.getTime();
                        slavetimes.add(slaveTime04);
                        System.out.println("time␣at␣slave␣04␣is:␣" + dateformatter.for

                        // Connect to slave 04 and get Time
                        Registry registry05 = LocateRegistry.getRegistry(berkeleyParan
                        clockServer ts05 = (clockServer) registry05.lookup(clockServer
                        System.out.println("Connected␣successfully␣to:␣SLAVE␣05");
                        LocalTime slaveTime05 = ts05.getTime();
                        slavetimes.add(slaveTime05);
                        System.out.println("time␣at␣slave␣05␣is:␣" + dateformatter.for

// COMPUTATION of TIME DIFFERENCE AND AVERAGING THEM

                        var localTimeSeconds = masterTime.toSecondOfDay();

                        var delta01 = slaveTime01.toSecondOfDay() - localTimeSeconds;
                        var delta02 = slaveTime02.toSecondOfDay() - localTimeSeconds;
                        var delta03 = slaveTime03.toSecondOfDay() - localTimeSeconds;
                        var delta04 = slaveTime04.toSecondOfDay() - localTimeSeconds;
                        var delta05 = slaveTime05.toSecondOfDay() - localTimeSeconds;

                        var delta_average = (delta01 + delta02 + delta03 + delta04 +
                        System.out.println("Average␣Delta␣Computed␣is:␣"+delta_average

                        // Assign new time
                        ts01.adjustTime(masterTime, delta_average);
                        ts02.adjustTime(masterTime, delta_average);
                        ts03.adjustTime(masterTime, delta_average);
                        ts04.adjustTime(masterTime, delta_average);
                        ts05.adjustTime(masterTime, delta_average);
                        masterTime = masterTime.plusSeconds(delta_average);
                        System.out.println("Successfully␣Updated␣the␣Clock...");
                        // Verifying the time on all machines
                        System.out.println("Master/Local␣Time:␣" + dateformatter.forma
                        System.out.println("new␣time␣at␣slave␣01␣is:␣" + dateformatter
                        System.out.println("new␣time␣at␣slave␣02␣is:␣" + dateformatter
                        System.out.println("new␣time␣at␣slave␣03␣is:␣" + dateformatter
                        System.out.println("new␣time␣at␣slave␣04␣is:␣" + dateformatter
                        System.out.println("new␣time␣at␣slave␣05␣is:␣" + dateformatter

                } catch (Exception ex) {
                        System.out.println(ex);
                }
        }

}
```

E source code: slave

```
package server;

import static global.berkeleyParams.dateformatter;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.time.LocalTime;
import global.berkeleyParams;

// Initiate the Slave 01

public class slaveOne {

        public static void main(String[] args) {
                try {
                        clockServer ts01 = new clockServerImplementation(LocalTime.pa
                        Registry registry01 = LocateRegistry.createRegistry(berkeleyPa
                        registry01.rebind(clockServerImplementation.class.getSimpleNam
                        System.out.println(String.format("Slave␣01␣started␣at␣port:␣%
                } catch (Exception ex) {
                        System.out.println(ex);
                }
        }

}
```