ANISH. ADNANI

D17B / 01

DC
(1)

EXPERIMENT 7

Aim: Write a program to demonstrate Berkeley clock synchronization algorithm.

Theory:

Clock synchronization deals with understanding the temporal ordering of event produced by concurrent processes. It is useful for synchronizing senders and receivers of messages, controlling joint activity, and the serializing concurrent access to shared objects. The goal is that multiple unrelated process running on different machine should be in agreement with each other and be able to make consistent decision about the ordering of events in a system.

Berkeleys algorithm is a solution to the clock synchronization problem in a centralized server environment. In a distributed system the problem takes more complexity because a global time is not easily known. The most used clock synchronization solution on the internet is the NTP which is layered client-server architecture based on UDP message passing.

Berkeley Algorithm

- It does not assume that every machine has an accurate time same with which to synchronize. Instead, it opts for obtaining an average time from the participating computers and synchronizing all machines to that average.

1) An individual node is chosen as the master node from a pool nodes in the network. This node is the main node in the network which acts as a master and rest of the nodes act as slaves. Master node is chosen using a election process / leader election algorithm.

2) Master nodes periodically pings slave nodes and fetches clock time at them.

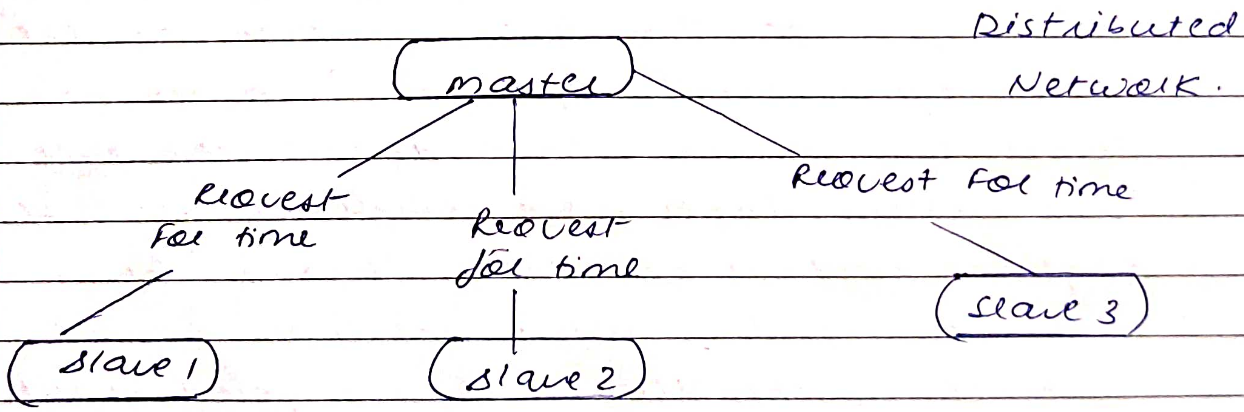
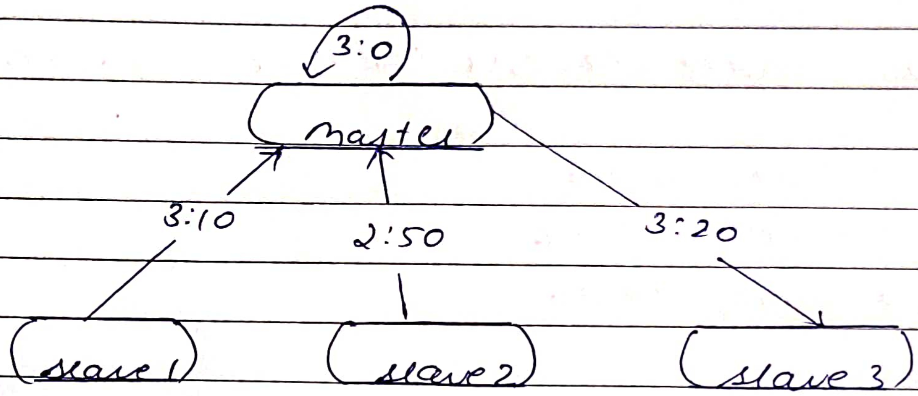


Diagram below illustrates how slave nodes and back time given by their system clock.



3) Master node calculates average time difference between all the clock time received and the clock time given by master's system clock itself. This average time difference is added to the current time at master system clock and broadcasted over the network

Example

Nodes in the distributed system with their clock timings

N1 → 14:00 (master node)
N2 → 13:46
N3 → 14:15

Step 1: The leader is elected, nodes N1 is the master in the system.

Step 2: Leader requests for time from all nodes.

N1 → time : 14:00
N2 → time : 13:46
N3 → time: 14:20

Step 3: The leader averages the times and sends the correction time back to the nodes.

N1 → corrected time 14:02 (+2)
N2 → corrected time 14:02 (+16)
N3 → corrected time 14:02 (+8)

This shows how the synchronization of nodes of a distributed system is done using berkeleys algorithm

Conclusion:

The algorithm also has provision to ignore readings from clocks whose skew is too great. The master may compute a fault tolerant averaging values from machine. If the master machine fails, any other slave could be elected to take over.

ANISH ADNANI D17B ROLL NO:01

```
import java.io.*;
import java.util.*;
public class BerkeleyAlgo
{
float diff(int h,int m,int s, int nh,int nm,int ns)
{
int dh = h-nh;
int dm = m-nm;
int ds = s-ns;
int diff = (dh*60*60)+(dm*60)+ds;
return diff;
}
float average(float diff[],int n)
{
int sum=0;
for(int i=0;i<n;i++)
{
sum+=diff[i];
}
float average = (float)sum/(n+1);
System.out.println("The average of all time differences is "+average);
return average;
}
void sync(float diff[], int n, int h, int m, int s, int nh[], int nm[], int ns[], float average)
{
for(int i=0;i<n;i++)
{
diff[i]+=average;
int dh = (int)diff[i]/(60*60);
diff[i]%=(60*60);
int dm = (int)diff[i]/60;
diff[i]%=60;
int ds = (int)diff[i];
nh[i]+=dh;
if(nh[i]>23)
{
nh[i]%=24;
}
nm[i]+=dm;
if(nm[i]>59)
{
nh[i]++;
nm[i]%=60;
}
```

```
ns[i]+=ds;
if(ns[i]>59)
{
nm[i]++;
ns[i]%=60;
}
if(ns[i]<0)
{
nm[i]--;
ns[i]+=60;
}
}
h+=(int)(average/(60*60));
if(h>23)
{
h%=24;
}
m+=(int)(average/(60*60*60));
if(m>59)
{
h++;
m%=60;
}
s+=(int)(average%(60*60*60)); if(s>59)
{
m++;
s%=60;
}
if(s<0)
{
m--;
s+=60;
}
System.out.println("The synchronized clocks are:\n Time Server "+h+":"+m+":"+s);
for(int i=0;i<n;i++)
{
System.out.println("Node "+(i+1)+" "+nh[i]+":"+nm[i]+":"+ns[i]);
}
}
public static void main(String[] args) throws IOException
{
BerkeleyAlgo b=new BerkeleyAlgo(); Date date = new Date();
BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter number of nodes:");
```

```
int n = Integer.parseInt(obj.readLine());
int h = date.getHours();
int m = date.getMinutes();
int s = date.getSeconds();
int nh[] = new int[n];
int nm[] = new int[n];
int ns[] = new int[n];
for(int i=0;i<n;i++)
{
System.out.println("Enter time for node " +(i+1)+"\n Hours:");
nh[i] = Integer.parseInt(obj.readLine());
System.out.println("Minutes:");
nm[i] = Integer.parseInt(obj.readLine());
System.out.println("Seconds:");
ns[i] = Integer.parseInt(obj.readLine());
}
for(int i=0;i<n;i++)
{
System.out.println("Server sent time "+h+":"+m+":"+s+" to node " +(i+1)); }
float diff[] = new float[n];
for(int i=0;i<n;i++)
{
diff[i] = b.diff(h,m,s,nh[i],nm[i],ns[i]);
System.out.println("Node " +(i+1)+" sent time difference of " +(int)diff[i]+" to Time Server.");
}
float average = b.average(diff,n);
b.sync(diff, n, h, m, s, nh, nm, ns, average);
}
}
```

C:\Windows\System32\cmd.exe

```
C:\Users\User\Desktop\sem8-exps-anish\DC\exp7>javac BerkeleyAlgo.java
Note: BerkeleyAlgo.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\User\Desktop\sem8-exps-anish\DC\exp7>java BerkeleyAlgo
Enter number of nodes:
3
Enter time for node 1
 Hours:
10
Minutes:
56
Seconds:
23
Enter time for node 2
 Hours:
1
Minutes:
7
Seconds:
55
Enter time for node 3
 Hours:
1
Minutes:
10
Seconds:
42
Server sent time 15:34:31 to node 1
Server sent time 15:34:31 to node 2
Server sent time 15:34:31 to node 3
Node 1 sent time difference of 16688 to Time Server.
Node 2 sent time difference of 51996 to Time Server.
Node 3 sent time difference of 51829 to Time Server.
The average of all time differences is 30128.25
The synchronized clocks are:
 Time Server 23:35:39
Node 1 23:56:39
Node 2 23:56:39
Node 3 23:56:39

C:\Users\User\Desktop\sem8-exps-anish\DC\exp7>
```