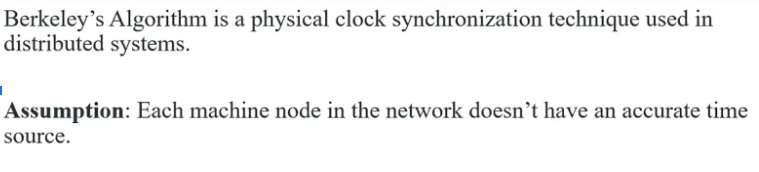
//Practical : 4 . Implement Berkeley

//Aim – . Implement Berkeley algorithm for clock synchronization.

//Program to demonstrate Berkeley clock synchronization algorithm



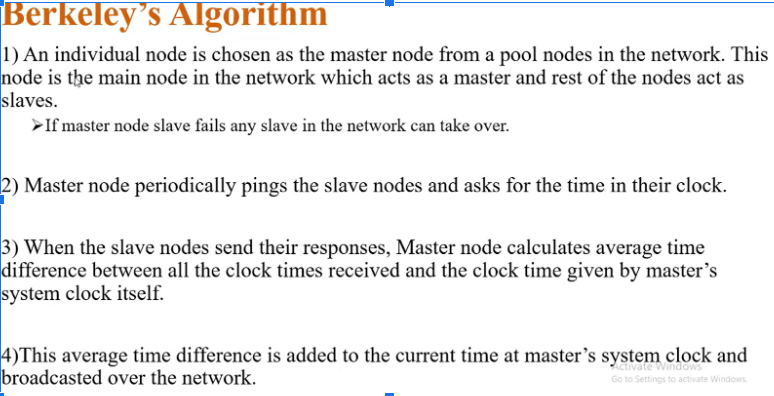
Pass 1 : Requests the time stamp from all the slave nodes.

Pass 2 : Save node responds the timestamps to their master

Pass 3 : Master node computes fault tolerant average

Avg i.e 1+2+3+4/4

Pass 4 : This avg time difference is added to the current time at master’s system clock and broadcasted over the network



There is other more advance algorithm such as the Network Time Protocol (NTP) which use a more complex algorithm and also consider the network delay and clock drift to get a more accurate time.

In distributed systems, there are different types of clocks used to manage time and coordinate events across multiple nodes. Here are some commonly used types of clocks:

Physical Clock:

A physical clock measures real-time based on the physical properties of the underlying system, such as the CPU clock or a hardware clock.

Physical clocks provide a notion of absolute time and are synchronized with a reference time source, such as a network time protocol (NTP) server.

Physical clocks are subject to clock drift and may not run at the exact same rate due to variations in hardware or environmental factors.

Logical Clock:

A logical clock captures the order of events in a distributed system, irrespective of real-time.

Logical clocks are often implemented using algorithms such as Lamport timestamps or vector clocks.

Each process or node in the distributed system maintains its own logical clock, which is incremented as events occur.

Logical clocks provide a partial ordering of events and are used to establish causality between events, even if they occur concurrently or on different nodes.

Lamport Clock:

Lamport clocks are a type of logical clock proposed by Leslie Lamport.

Each process in a distributed system maintains its own Lamport clock, which is incremented whenever an event occurs.

The Lamport clock value represents the logical time of an event and is used to order events across different processes.

Lamport clocks provide a total ordering of events if there is a causal relationship between them.

Vector Clock:

Vector clocks are another type of logical clock used in distributed systems.

Like Lamport clocks, each process maintains its own vector clock, but it consists of an array of logical clocks, one for each process.

Vector clocks capture causality by including information about the relative ordering of events across different processes.

Vector clocks are useful for determining causality and detecting concurrent events.

These are some of the commonly used clocks in distributed systems. Each type of clock has its own characteristics and is suitable for different purposes, such as ordering events, enforcing consistency, or detecting causality violations. The choice of clock depends on the specific requirements and characteristics of the distributed system being designed.

Difference between logical and physical clock

They serve different purposes and have different characteristics:

Physical Clock:

A physical clock is a clock that measures real-time based on the physical properties of the underlying system, such as the CPU clock or a hardware clock.

Physical clocks provide a notion of absolute time and are synchronized with a reference time source, such as a network time protocol (NTP) server.

Physical clocks are subject to clock drift, which means they may not run at the exact same rate due to variations in hardware or environmental factors. Over time, these clocks may diverge from each other.

Physical clocks are typically used for timestamping events, measuring time intervals, and establishing a global time reference in distributed systems.

Logical Clock:

A logical clock is a clock that captures the order of events in a distributed system, irrespective of real-time.

Logical clocks are typically implemented using algorithms such as Lamport timestamps or vector clocks.

Each process or node in the distributed system maintains its own logical clock, which is incremented as events occur.

Logical clocks provide a partial ordering of events and are used to establish causality between events, even if they occur concurrently or on different nodes.

Logical clocks do not require global synchronization or coordination among nodes, and they are not affected by clock drift.

Logical clocks are often used for enforcing consistency, determining the order of message delivery, and detecting causality violations in distributed systems.

In summary, physical clocks provide a notion of real-time and synchronization, while logical clocks focus on establishing a causal order of events in a distributed system. Both types of clocks are important in managing time and coordination in distributed systems, each serving specific purposes depending on the requirements of the system.

Berkeley's algorithm, also known as the Berkeley algorithm or Berkeley time synchronization algorithm, is a clock synchronization algorithm designed for distributed systems. It aims to synchronize the clocks of multiple computers in a network by electing a coordinator and adjusting the clocks based on the coordinator's time.

Here's a step-by-step explanation of the Berkeley algorithm:

1. Coordinator Election:
   * One of the nodes in the network is elected as the coordinator. This can be done using various leader election algorithms or based on a predetermined rule.
2. Time Request:
   * The coordinator periodically sends a time request message to all other nodes in the network.
3. Local Clock Adjustment:
   * Upon receiving the time request, each node records its local clock time and sends it back to the coordinator in a time response message.
4. Coordinator Calculation:
   * The coordinator collects the time response messages from all nodes.
5. Calculation of Time Discrepancy:
   * The coordinator calculates the time discrepancy (difference) between its own clock time and the recorded clock times of the participating nodes.
6. Time Adjustment:
   * The coordinator computes an average time adjustment value based on the time discrepancies.
7. Time Update:
   * The coordinator broadcasts the time adjustment value to all nodes in the network.
8. Local Clock Adjustment:
   * Each node receiving the time adjustment value adjusts its local clock accordingly by applying the computed time adjustment value.
9. Synchronization Period:
   * The above steps are repeated periodically to maintain clock synchronization as the clocks in the network may drift over time due to various factors.

The Berkeley algorithm aims to minimize the time discrepancies between the clocks of participating nodes by adjusting the local clocks based on the coordinator's time. By periodically synchronizing the clocks, it helps maintain a reasonable level of time consistency across the distributed system.

It's important to note that the Berkeley algorithm assumes that the clock drift rates of the participating nodes are relatively small and that the network communication delay is relatively constant. If the clock drift rates are significant or the network delay varies widely, additional techniques or algorithms may be required to achieve more accurate clock synchronization.

The Berkeley algorithm is one of the basic approaches to clock synchronization in distributed systems and has been widely used in various applications where approximate time synchronization is sufficient.

import java.io.\*;

import java.util.\*;

public class Assignment4

{

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:\nTime Server ---> "+h+" : "+m+" : "+s);

for(int i=0;i<n;i++)

{

System.out.println("Node "+(i+1)+" ---> "+nh[i]+" : "+nm[i]+" : "+ns[i]);

}

}

@SuppressWarnings("deprecation")

public static void main(String[] args) throws IOException

{

Assignment4 b = new Assignment4();

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("Time 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:\javac](about:blank) Berkley.java

[c:\java](about:blank) Berkle

