CMPSCI 677 Operating Systems **Spring 2019**

Lecture 12: March 18

Lecturer: Prashant Shenoy *Scribe:* ***Jaskaran Singh, Abhiram Eswaran(2018)***

Announcements:
Midterm Exam on Friday Mar 22,
Lab 2 will be released today, it is due after the exam.

12.1 Overview

The topic of the lecture is "Time ordering and clock synchronization". This lecture covered the following topics.

Clock Synchronization : Motivation, Cristians algorithm, Berkeley algorithm, NTP, GPS

Logical Clocks : Event Ordering

12.2 Clock Synchronization

12.2.1 The motivation of clock synchronization

In centralized systems and applications, it is not necessary to synchronize clocks since all entities use the system clock of one machine for time-keeping and one can determine the order of events take place according to their local timestamps.

However, in a distributed system, lack of clock synchronization may cause issues. It is because each machine has its own system clock, and one clock may run faster than the other. Thus, one cannot determine whether event A in one machine occurs before event B in another machine only according to their local timestamps. For example, you modify files and save them on machine A, and use another machine B to compile the files modified. If one wishes to compile files in order and B has a faster clock than A, you may not correctly compile the files because the time of compiling files on B may be later than the time of editing files on A and we have nothing but local timestamps on different machines to go by, thus leading to errors.

12.2.2 How physical clocks and time work

1) Use astronomical metrics (solar day) to tell time: Solar noon is the time that sun is directly overhead. Noon is different from solar noon. Noon depends on time zone while solar noon is exactly the same time. We typically use the notion of solar day to tell there are 24 hours between the time that sun is directly overhead on a particular location. Although this method was used for centuries, it is not accurate since it based on the length of a day.

2) Accurate clocks are atomic oscillators: Such a clock uses the property of atoms and their accuracy is

1 part in 1013. It is the most accurate type of clock used today. If you have an atomic clock and broadcast clock values, then the receivers can synchronize with the atomic clock. For example, in US, there is a shortwave radio station that broadcasts the atomic clock value, so that the receiver can listen and match their times. Cell-phone clocks are synchronized by using atomic clock located in cell-phone broadcast towers. Some satellites also broadcast atomic clock.

3) Coordinated universal time(UTC): International standard based on atomic time. UTC broadcast on radio and receivers accurate to 0.1-10ms.

4) Mechanical clock: less accurate, accuracy is one part in million. Computers use mechanical clock. Their lack of precision results in clock drift, since the properties of quartz(the material used to keep time) would change with changes in temperature and humidity. To avoid clock drift, we need to synchronize machines with a master or with one another.

12.2.3 Drift tolerance and frequency of synchronization

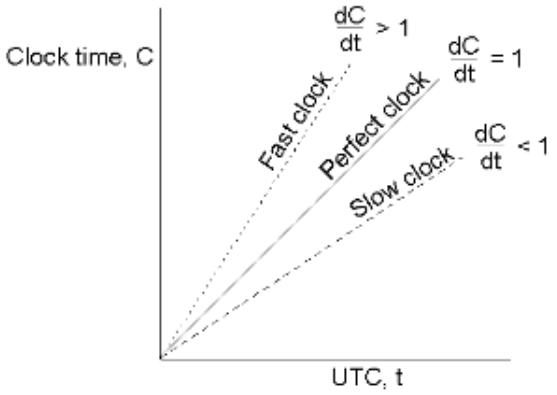

Figure 12.1: Clock Drift

If t stands for UTC time and C stands for clock time, then dC/dt equals to 1 for a perfect clock as 1s in UTC time corresponds to 1s increase in clock time. When dC/dt is less than 1, the clock gains $1-\rho$ seconds for 1s increases in UTC time (it is slower). When dC/dt is more than 1, the clock gains $1+\rho$ seconds for 1s increases in UTC time (it is faster). Here ρ is the drift rate, which tells us how slowly/quickly the clock drifts away from "perfect time". This rate depends on the type of clock used.

If there are two machines, each clock has a maximum drift rate ρ, then $1-\rho \leq dC/dt \leq 1+\rho$. Hence, the two clocks in two machines may drift by $2\rho\ \delta t$ in time δt in the worst case (one has a fast clock while the other has a slow clock). If the systems are to limit time drift to δ, we need to resynchronize every $\delta/2\rho$ seconds.

12.2.4 Centralized clock synchronization algorithms

12.2.4.1 Cristians algorithm

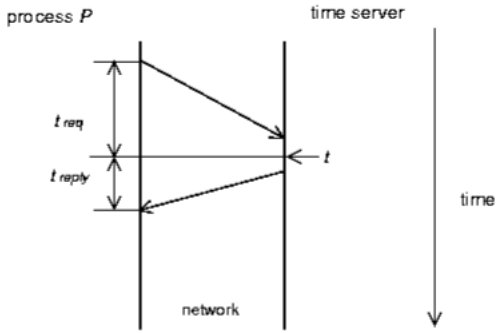

Figure 12.2: Cristian's Algorithm Example

It assumes there is a master machine called time server, which somehow synchronizes with an atomic clock via a UTC receiver and has a timestamp for the system. Other machines of the system synchronize with the time server. Every $\delta/2\rho$ seconds, machine P sends network messages to time server to check what is the current time, and time server returns the current time and machine P uses the time to reset the clock of the machine P.

For example, machine P requests time from time server. After t_{req} time server receives the request, does some processing, and returns time t to machine P. After t_{reply} machine P receives a message with time t from server. However, it must account for the time it took for the request to reach the server, and the time taken by response to come back. So, setting local time to t would be inaccurate. Ideally, we would like to set the clock to (t + t_{reply}). But we cannot find the exact value of t_{reply}.

So, we use (t_{req} + t_{reply})/2 = (the time of machine P receives reply the time of machine P sends request) / 2 as an estimation of t_{reply}. In such estimation, you assume the time from machine P to server is identical to the time from server to machine P though it is usually not the case in practice, but we can use this assumption to give an estimation. To improve accuracy, we can estimate by making a series of measurements.

Question: After synchronization by this method, what is the order of magnitude of the difference between local time and server time?
Answer: Depends on how accurate your estimate of t_{reply} was.

Question: Would it be helpful to send your current time to the timserver as part of the request?
Answer: Not really. What we need to calculate to synchronize the local clock is the network delay and your current time value would not feature in that calculation. You can just keep track of when you sent the request and when you get the response locally to estimate network delay.

Question: Can the timeserver also be off from UTC time?
Answer: It definitely can be, but we assume that the timeserver is authoritative.

12.2.4.2 Berkeley algorithm

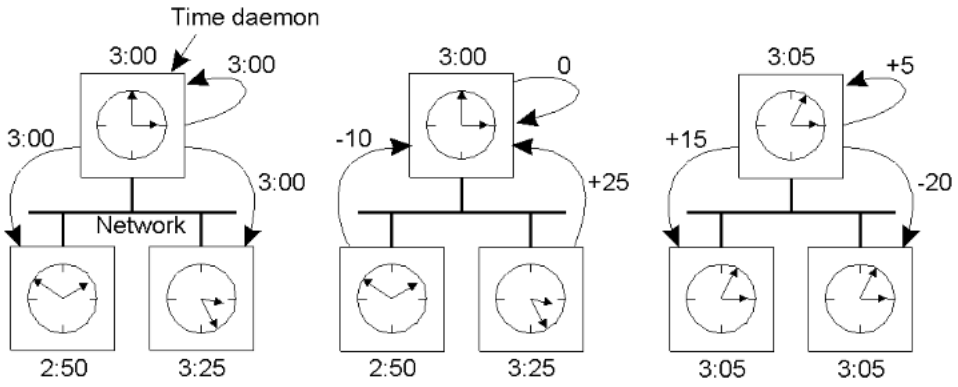

Figure 12.3: Berkeley Algorithm Example

This algorithm just keeps clocks synchronized with one another in a group, and no machine in this group synchronize with external atomic clock. For this algorithm, the absolute time value is not important and we want to know clock differences between machines in a certain system. In this algorithm, we use leader election to select a master in a group to run clock synchronization while others are slaves. Synchronization can be done in the following steps:

a) Master asks all the other machines for their clock values by sending the time value on master;
b) The other machines answer;
c) Master tells everyone how to adjust their clock.

Since some clock are fast and some clock are slow, this algorithm takes average time difference to adjust their clock. For example, three machines reply with their clock values as time difference of 0, -10, +25 at 3:00, then the master will tell all those machines to set their clock at 3:00 + 5 (5 = (0 - 10 + 25) / 3). We still need to account for network delays, as in Cristian's algorithm.

Question: Wouldn't setting faster clocks back cause problems?
Answer: This does create some problems. For example, if you modified a file at 3:25 and save it, but then the clock goes back to 3:05, you would have a file saved in the future which doesn't make sense. Another problem that might occur is that two non-concurrent events might get the same timestamp.

To avoid issues like these, you generally do not set clocks back. Instead, you can make the faster clocks run slower, i.e. their time increases by 1s for every 1.2s (for example). This would hopefully ensure that eventually they are consistent with actual time.

12.2.5 Distributed clock synchronization approaches

Cristians algorithm and Berkeley algorithm are both centralized approaches since they need time sever or time demon that run clock synchronization.

There are also decentralized algorithms that use resynchronization intervals. Each machine in a certain system broadcasts time at the start of the interval and collects all other broadcast that arrive in a period S, then uses average value of all reported times. For the outliers, machines can throw away few highest and

lowest values to avoid negative influence of extremely fast or slow clocks to the average time.

There are two decentralized approaches in use today. One approach is using NTP which is used in most computers to synchronize clock. NTP uses advanced techniques for accuracies of 1-50ms. If the time difference in the system is smaller than accuracy of NTP clock synchronization, then you cannot use NTP since you cannot say which event happens before another. The other approach is rdate which synchronizes a machine with a specified machine. In many cases, you can run rdate with the argument of the name of server and just synchronize clock with that server.

(At this time, there was a short demonstration of how one can set the NTP server for a Mac.)

12.2.5.1 Network Time Protocol(NTP)

NTP is widely used standard which based on Cristians algorithm. In NTP clock synchronization, you also want to find out network propagation delay (dT_{res}). The difference between NTP and Cristians algorithm is NTP clock synchronization uses hierarchical protocol and does not let the clock to be set backward. For Cristians algorithm, fast clock and slow clock can just resynchronize with time server to adjust their time. However, for NTP clock synchronization, fast clock cannot go backward, and it is synchronized by slowing down fast clock. Letting clock go backward can cause many negative consequences (Like two files having the same timestamp). This is the reason why NTP is widely used compared to Cristians algorithm.

Question: How is Daylight Savings Time(DST) accounted for?
Answer: DST is related to your timezone, not the universal time(UTC). It essentially changes just your offset from UTC. The server always keeps track of UTC and can send the time to a machine, calculating the current offset/timezone, be it -5hrs or -6hrs.

12.2.5.2 Global Positioning System(GPS)

GPS not only does clock synchronization, but also figures out the location of the block and the location of the receiver. GPS achieves high accuracy because it is synchronized with satellites which use atomic clock without hierarchical protocol.

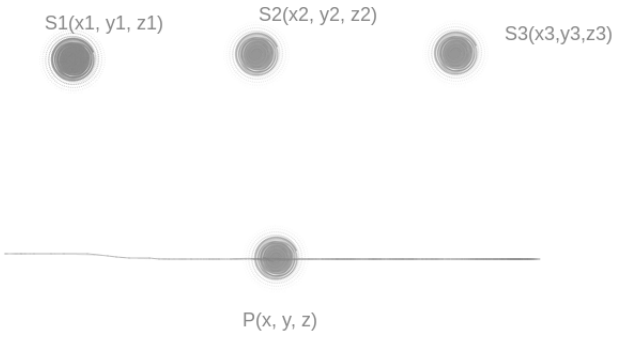

Figure 12.4: GPS Example

GPS landmarks broadcast their locations and an unknown node will estimate its location according to landmarks locations.

How GPS works: We assume GPS landmark S1 with its position (x1, y1, z1) and its timestamp t1, and

GPS receiver P (e.g. a car) with its unknown position (x, y, z) and the timestamp t receiving broadcast t1 message from GPS landmark.
Then the distance between S1 and P is

$$d_i = \sqrt{(x1 - x)^2 + (y1 - y)^2 + (z1 - z)^2}$$

at the same time

$$d_i = c * (t - t1)$$

c being the speed of light. We assume the receiver has a drift time d_r from landmark S1. Taking time drift into account and combining the two equations gives us:

$$c(t + d_r - t1) = \sqrt{(x1 - x)^2 + (y1 - y)^2 + (z1 - z)^2}$$

We can see that there are 4 unknowns x, y, z, d_r, thus we need a minimum of 4 satellites to compute the location of a GPS receiver as well as its time drift value. If we get 4 satellites, then we can get multiple solutions of the location of the receiver. If we have 6 or 8 satellites, we can get more accurate solutions. In this way, GPS does clock synchronization as well as computing receivers location.

Question: Is the position of satellites fixed?
Answer: Satellites used for this are geostationary, i.e their relative position with respect to the Earth is fixed.

12.3 Logical clock

The above clock synchronization approaches assume synchronize local clock and use timestamp to reason the order of events. If the time difference between two events is smaller than the accuracy, then we cannot say which event happens first, thus problems may be caused. For many problems, we care about the internal consistency of clocks, not absolute time. The approach using logical clocks is proposed. Logical clocks do not need clock synchronization and take the order in which events occur rather than the time at which they occurred into account. If the processes only care about event A happens before event B, but dont care about the time difference exactly, they can use logical clock.

12.3.1 Event Ordering and Algorithm

Logical clocks are to solve the problem that define a total ordering of all events that occur in a system. In a distributed system, logical clocks do not have global clock and local clocks may be unsynchronized. Besides, you cannot order events on different machines using local times. There are some key ideas of logical clocks proposed by scientist Lamport: can use send/receive messages exchanged between processes/machines to order events since messages must be sent before received. We then use the transitivity property to reason about order of events.

For example, machine A sends a message to machine B along with its logical clock value 4, and machine B receives this message at its local logical clock value 3, then we can say that the events happens before logical clock value 4 in machine A occur before the events happens after logical clock value 3 in machine B without clock synchronization. However, we cannot say the order of the events happens after logical clock value 4 in machine A and the events happens before logical clock value 3 in machine B. This indicates that

this algorithm only gives us a partial ordering of events.

Rules for updating logical clock values:
1) Whenever a local event occurs at machine i, it updates LC_i as $LC_i += 1$.
2) When machine i wants to send a message, it includes LC_i in the message.
3) When machine j receives a message, it updates its logical clock as $LC_j = max(LC_j, LC_i) + 1$.

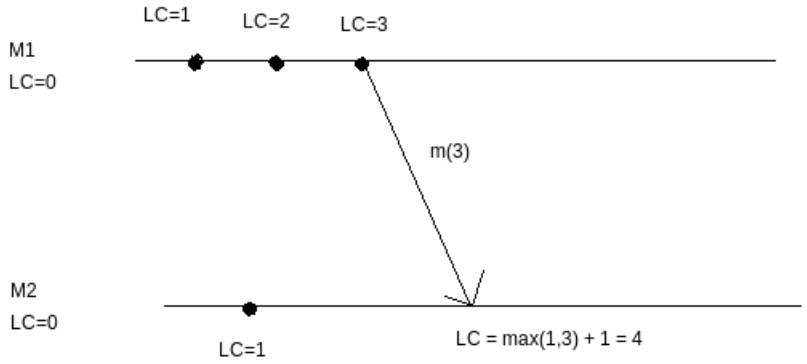

Figure 12.5: Logical Clock Example

Important to note that if we know the order of two events A and B, this algorithm ensures that their timestamps will be in order(i.e if $A \rightarrow B$, then $ts(A) < ts(B)$). However, we cannot be certain about the order of events, given just their timestamps i.e. $ts(A) < ts(B) \not\Rightarrow (A \rightarrow B)$ because all we have is a partial ordering.