CMPSCI 677 Operating Systems **Spring 2022**

Lecture 12: March 07

Lecturer: Prashant Shenoy *Scribe: **Ritvika Pillai***

12.1 Overview

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

Clock Synchronization: Cristian's algorithm, Berkeley algorithm, NTP, GPS

Logical Clocks: Event ordering

12.2 Clock Synchronization

12.2.1 The motivation of clock synchronization

Just like usual clocks, systems have a clock that tells time to applications running on the system. Centralized machines have just 1 clock, but in the case of distributed systems each machine has its own clock. All these clocks might not be in sync and may drift over time. Hence the time will be dependent on which local clock is being checked.

For example, you modify files and save them on one machine A, and use another machine B to compile the files modified. If machine B has a faster clock than machine A, you may not correctly compile the files modified because the time of compiling files on machine B may be later than the time of editing files on machine A according to local timestamp on different machines, thus leading to errors.

12.2.2 How physical clocks and time work

1) One approach is to use astronomical metrics (solar day) to tell time. For example, solar noon is the time that sun is directly overhead. "Noon" on our clocks is different from solar noon. Noon depends on time zone while solar noon is a fixed physical fact. 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) Atomic clocks use the properties of atoms to measure time. Atomic clocks are the most accurate clocks and other clocks derive from this time. Typically, you will have some centralized atomic clock broadcast its time. The receivers, which may use less accurate mechanisms, are then synchronized with the atomic clock. For example, cell-phone clock also uses atomic clock to synchronize time with cell-phone broadcast tower. Some satellites also broadcast time based on atomic clocks.

3) Coordinated universal time (UTC) is based on noon in Greenwich (UK). All time zones are offset by UTC. Most of the atomic clocks broadcast UTC time regardless of timezone using wireless channels, satellites, FM radios, etc. Receivers listen to this and set local time based on the broadcast time.

4) Mechanical clock are less accurate—the accuracy is roughly one part per million. Computers typically use mechanical clocks. This small amount of inaccuracy results in clock drift because the property of the physical mechanism (usually quartz) can change with environmental properties such as temperature or 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

Actual clocks have drift and hence need synchronization once in a while. This drift needs to be calculated to determine how frequently synchronization is needed.

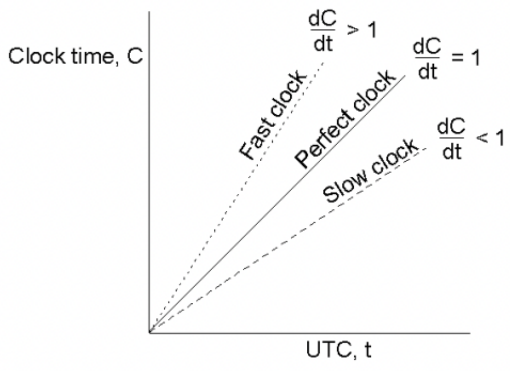

Figure 12.1: Clock drift relative to a perfect clock.

In Figure 12.1, Here, t is UTC time, C is clock time, and slope, dC/dt, is the rate of advancement of the clock. ρ indicates the inaccuracy of the clock. Consider the following cases: a. If the $dC/dt = 1$, the real time and clock advances proportionally and are in sync. b. If the $dC/dt < 1$, real time advances by 1 second and the clock will advance by $(1 - \rho)$ second, i.e., the clock runs slower. c. If the $dC/dt > 1$, real time advances by 1 second then clock will advance by $(1 + \rho)$ second, i.e., the clock runs faster. To limit the error in clock to δ, we need to synchronize every $\delta/2\rho$ seconds.

12.3 Centralized clock synchronization algorithms

12.3.1 Christian's Algorithm

In Cristian's Algorithm, is a master machine called *time server* which is the authoritative clock for telling time. It is in sync with the atomic clock via a UTC receiver. Other machines in the system synchronize with the time server.

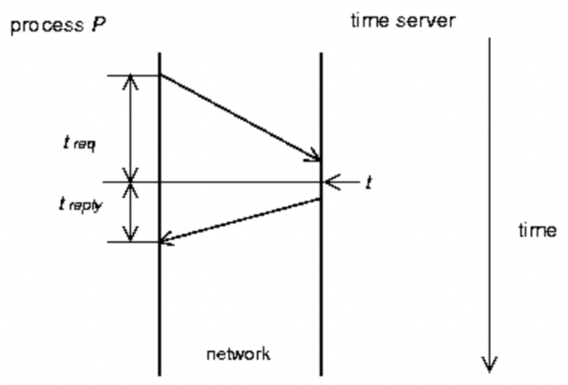

Figure 12.2: Cristian's Algorithm.

Machine P sends message to the time server to check the current time. After taking some time (t_{req}) to propagate, the request reaches the time server and will then be processed. The time server then returns the current time (t) and machine P uses this time to reset its clock. The machine P will set its time as ($t+t_{reply}$) and not just t. This is done so as to take the propagation delay from server to machine P into account. We can use $(t_{req} + t_{reply})/2$ as an estimation of t_{reply}. The better the estimate, the better the synchronization.

12.3.2 Berkeley Algorithm

This algorithm doesn't use a time server. Instead, clocks are synchronized with one another in a group, and no machine in this group synchronize with external atomic clock. We use leader election to select a "master" in a group to run clock synchronization while others are "slaves." This master clock is known as the coordinator. Each machine sends their local time to the coordinator. The coordinator then calculates an average of these times. Based on the value of average, the time of all clocks are adjusted. 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 $3:00 + 5(5 = (0 - 10 + 25)/3)$. This is a relative clock synchronization algorithm, not an absolute synchronization algorithm. The propagation times are estimated in the same way as Cristian's algorithm.

12.4 Distributed clock synchronization approaches

Both Cristian's and Berkley are centralized algorithms. Apart from these, there are also decentralized algorithms using resynchronized intervals. In a decentralized version of Berkley, the role of coordinator is eliminated. Instead, all machines broadcast their times to all other machines at the start of the interval. At every machine, suppose n clock values are received within the interval. Then at the end of period S, their average is calculated which is then used to set their local time. For the outliers, machines can throw away few highest and lowest values to avoid negative influence of extremely fast or slow clocks relative to the average time.

There are two decentralized approaches in use today. One approach is using NTP which is used by most computers. It uses a time server and advanced techniques to deal with network propagation delays. The accuracy is typically between 1 and 50ms. 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.

12.4.1 Network Time Protocol (NTP)

NTP is widely used standard which based on Cristian's algorithm. In NTP clock synchronization, you also want to find out network propagation delay (dT_{res}). NTP clock synchronization uses a hierarchical protocol and unlike Cristian's algorithm, it does not let the clock be set backward. Since the fast clock cannot go backward, it is synchronized by slowing it down. Letting a clock go backward can have many negative consequences (such as two files having the same timestamp). This is the reason why NTP is widely used compared to Cristian's algorithm.

12.4.2 Global Positioning System (GPS)

GPS is a technology that allows any device to figure out its location. It requires clock synchronization to accurately figure out where the device is located. For example, a phone has a GPS chip that listens to satellite broadcasts. Uses the principle of triangulation to know where you are with respect to the known position of the satellite. These known positions are called landmarks. GPS achieves high accuracy because it is synchronized with satellites which use atomic clocks without a heirarchical protocol.

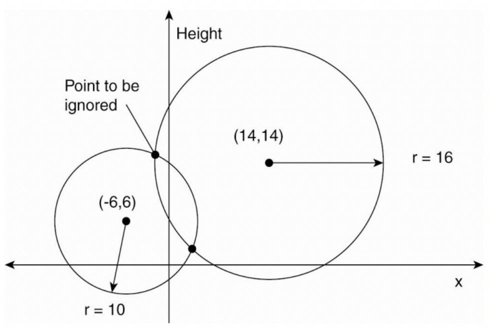

Figure 12.3: Global Positioning System (GPS)

2D space: Let the 2 landmarks be (14,14) and (-6,6). A device somewhere in this space will measure its distance with respect to these landmarks. Say, it is 16 units from the first landmark and 10 units from the second, this means that the device is on the intersection of the 2 circles because it has to satisfy both the distance constraints. If a 3rd landmark is added, then the exact position of the device can be known.

3D space: We assume GPS landmark A with its position (x_1, y_1, z_1) and its timestamp t_1, and GPS receiver B (e.g. a car) with its unknown position (x, y, z) and the timestamp t receiving broadcast t_1 message from a GPS landmark. Then the distance between A and B is $di = \sqrt{x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2}$, and di also equals $c(t_2 - t_1)$, where c is the speed of light. If we assume the receiver has a drift time dr from landmark A, then we can use Equation (1) to show that $c(t_2 + dr - t_1) = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2}$. From Equation (1), we can see that there are 4 unknowns, x, y, z and dr, thus we need minimum 4 satellites to compute the location of a GPS receiver as well as its time 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 quickly narrow the solutions. Therefore, GPS does clock synchronization as well as computing the receiver's location.

12.5 Logical clock

The above approaches use timestamps 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. In some cases, if processes need to know the order in which the events occurred instead of the exact time, then logical clocks should be used. Hence absolute time isn't important and clock synchronization isn't needed.

12.5.1 Event Ordering

In logical clocks, there is no global clock and local clocks may run faster or slower. The ordering of events needs to be figured out in such a situation. There are some key ideas of logical clocks proposed by the scientist Lamport: we can use send/receive messages exchanged between processes/machines to order events, and if 2 processes never communicate with each other, then in such cases we don't need to find order.

The happened-before relation: We use the fundamental property that the send event occurs before the receive event. The relation is transitive, i.e, if A occurs before B and B occurs before C, then A occurs before C.

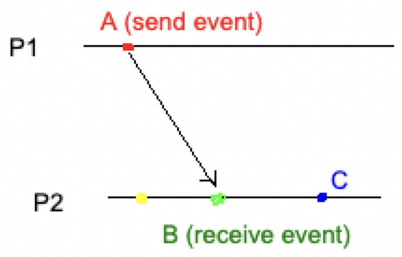

Figure 12.4: The "happened-before" relation. Note that we cannot say anything about the relation with the yellow event.

Each processor has a logical clock which gets incremented whenever an event occurs. Suppose when process i sends a message to process j, it piggybacks its local timestamp (say LCi=3) along with the message. The receiver takes this timestamp and its local timestamp (say LCj=4). Then the maximum of both these values is calculated and incremented by 1, i.e, max(LCi, LCj) + 1. This makes sure that the timestamp assigned to the receiver event is higher than the sending event. This technique was invented by Leslie Lamport.

The above algorithm only solves half the problem as it gives only forward property and not the reverse property (i,e if $timestampA < timestampB$ then A has occurred before B). Hence further changes are needed in this approach.