Distributed Systems		Date:	January 11, 2021
Instructor: *Lini Thomas*		Scribes:	Kalp Shah, Vivek Puar, Aditya Morolia

Lecture 2

NTP

We gave a presentation on the NTP (Network Time Protocol), which is a protocol which is used to synchronise time over a complete distributed system.

Requirement

Ordering of events is very important in a distributed system and thus, time is an important concept as one can order events only when a notion of time is established.

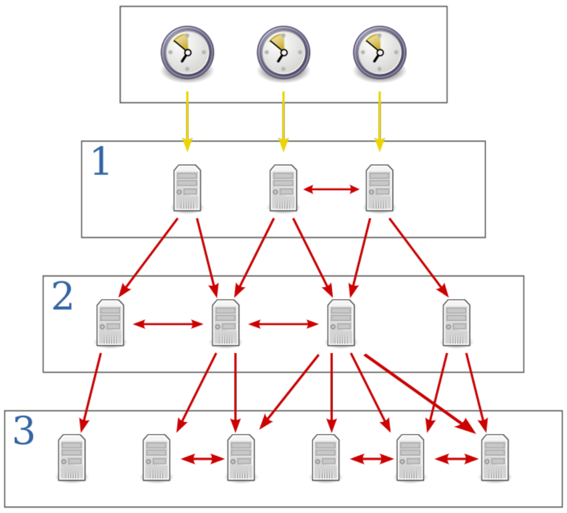
An example that can be taken is of a bank where one might have a balance of \$100 and makes a deposit of \$50 and withdrawal of \$150. The bank should not stop the second transaction as the amount to withdraw is equal than the balance. But if there is no notion of time, the bank would not know which request to process as is does not know which happened earlier, and thus if the withdrawal is handled earlier, the request would be cancelled.

A common notion of time is also required for implementation of timeouts in protocols and algorithms. It also helps in reducing communication as every node of the system, having the same time, does not need to ask each other of time whenever a time critical task is to be accomplished.

Concept

NTP protocol is based around two main ideas, the first being that not every node in a system requires an extreme precision in time and the other being that the amount of communications required to get the time must be as minimal as possible.

The schematic of the NTP algorithm is as follows :



It can be seen that the nodes are divided into multiple layers known as strata, which are divided by the accuracy required. The lower the number of the strata, more is the precision of the time.

Any node in the system can interact with its parents and in between itself, so thus practically it is at most as precise as the strata above it and thus the precision goes down while going down strata. Due to this design though, the amount of communications for a system are greatly reduced as it only communicates with a part of the system rather than the complete network.

It can be thus seen that the the main two ideas greatly complement each other and thus influence the design to be what it is.

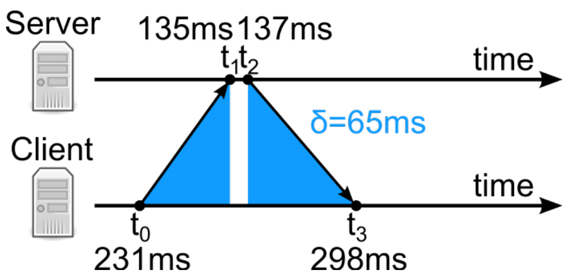
Advantages

There are two major advantages to using the NTP algorithm for time keeping. The first one is that if there is a node, which is not in sync, the first few changes in time are massive and thus the node comes back to some semblance of common time in a very small time, with other smaller changes taking a bit longer.

The other advantage is that it uses UDP which is a protocol where packets are broadcasted in one direction without the requirement to confirm the receiving of the packet. Due to not requiring confirmation, the communication time decreases greatly and thus in turn making NTP faster.

Algorithm

The algorithm of the system is very simple, where the client polls all the servers its allowed to poll, to get two quantities known as time offset (θ) and round trip delay (δ). The client then approximates the time from the three servers with least δ. The scematic for an individual client and server interaction is as follows :



The calculation of θ and δ is as follows :

$$\theta = \left|\frac{(t_1 - t_0) + (t_2 - t_3)}{2}\right|$$

$$\delta = (t_3 - t_0) - (t_2 - t_1)$$

All the values are communicated to the client after which it computes the δ_i and θ_i, thus all the computation is done by client.

Questions

There were two questions asked after the presentation. The first one was about the amount of pairs (θ, δ) kept by the client for approximation of time. The presentation had the figure 8 given while also telling that the implementation was important and the value will change depending on it.

Ques : The first question asked about the parameters that help in determination of the amount of pairs (θ, δ)

Ans : The answer is that it depends on how critical is the computation of time. As proved earlier, the lower the number of strata, the more precise is its computation of time. Thus the client on numerically lower strata requires a more precise definition of time, and hence there the amount of samples taken is higher to ensure that the node has a precise definition.

On the flip side, a node with higher numerical strata does not require, nor does it have that concise definition of time ,and thus the samples taken are lower, as the noise there is sufficiently high to make the amount of sample negligible.

Ques : The second question asked is the consequence of a fluctuating reading of (θ, δ), due to unreliable connection.

Ans : The answer is that is does not matter, as the system is not taking multiple samples to compute a single value but rather going around to every server once to approximate the time. This is due to the fact that the precision in time is not as important as the time in which is is computed and thus it would have no consequence.

Lecture

The lecture was on logical time, with revision of its definition and explanation of causality. The lecture then continued on to teach logical concurrency, followed by consistency and ended with beginning the topic of implementations of consistent networks.

Logical Time

Logical time is a mechanism where casual order in which events occurred at different processes is defined. A process occurring before another is given by $e_i \rightarrow e_j$, if and only if event e_i occurred before e_j.

Three rules are used to define this order :

- Two process occurring at the same process happen in the order in which they were observed.
- Send has a sequential priority over receive.
- Transitivity has to maintained.

Thus if there is a set of events $h_i = \{e_i^1, e_i^2, \ldots, e_i^n\}$, then an order H_i can be defined on it such that it gives a binary relation $\rightarrow$ on event h_i such that $e_i^k \rightarrow e_i^j$ if and only if e_i^k occurs before e_i^j.

This was ordering in a process. Ordering between processes is also done on send and receive messages, where as defined earlier, send has the sequential priority over receive. Thus if e_i sends to e_j, then $e_i \rightarrow_{msg} e_j$. Thus using these two ordering methods, one can order all the events. Thus ordering of two events is done as follows :

$$e_i^x \rightarrow e_j^y \text{ iff } \begin{cases} x < y & \text{if } i = j \\ e_i^x \rightarrow_{msg} e_j^y \\ \exists\ e_k^z \text{ st } e_i^x \rightarrow e_k^z \ \&\ e_k^z \rightarrow e_j^y \end{cases}$$

One can then define H such that $H = \bigcup H_i$.

Logical Concurrency

Logically concurrent events $(e_i || e_j)$ are such that $e_i \not\rightarrow e_j$ and $e_j \not\rightarrow e_i$. Logically concurrent events may not happen at physically the same time. One of the properties of logical concurrency is that it is not associative.

Consistency

Logical clock as a function can be implemented as :

$$C : H \to T \text{ st for any two } e_i \to e_j$$
$$C(e_i) < C(e_j)$$

If a system satisfies this function for all e_i, e_j, then the system is known as consistent. If an additional constraint $e_i \to e_j \iff C(e_i) < C(e_j)$, then the system is strongly consistent.

There are three implementations of providing a system with a C, which focus on two things, local and global movements. The methodologies are :

- Scalar Time
- Vector Time
- Matrix Time

Scalar Time

There are three rules of implementing Scalar time :

Rule 1 : Before executing an event, process p_i executes the following :

$$C_i := C_i + d(d > 0)$$

Rule 2 : Each message piggybacks the clock value of its sender at sending time.

Rule 3 : Then the node is updated as follows :

$$C_i := \max(C_i, C_{msg})$$
Execute Rule 1
Deliver the message

These are the rules of the scalar time algorithm. One of the property of this algorithm is that at least n/d events occur before time n.

Interesting Discussion

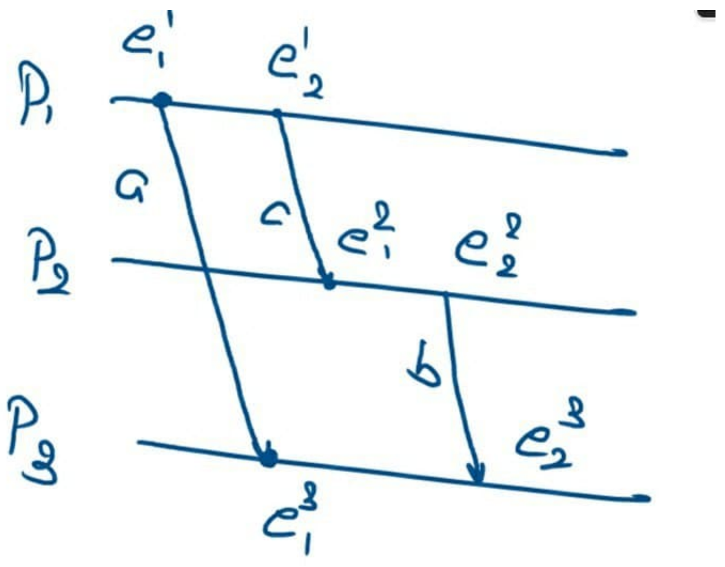
One discussion of particular interest was of construction of a DAG to find out if the edges are causally dependent on each other. The DAG is constructed in the manner that the first event is taken as the root and if there is a $\to$ or a $\to_{msg}$ between the events, there is an edge between them.

This was proven to be correct, that the DAG constructed would indeed give the causal dependency. Another point noted was that a logical time implementation can be done using this DAG by running a BFS on it. This method was proven to give consistent but not strongly consistent function C.

Question

The end of the lecture question was to prove that scalar time does not provide strongly consistent functions C.

The other questions were based on out of order messages, where $P_1 \to aP_3$, $P_1 \to cP_2$ and $P_2 \to bP_3$.



There are two problems with out of order messages :

1. In absence of strong consistency, one cannot know if one timestamp being smaller than the other means that one happened before the other
2. One would not know if a message of lower timestamp is on the way and yet to arrive

The questions are :

- Can P_3 identify which event a or b was sent first, with the help of scalar time.
- Will strong consistency solve problem 1.
- Will strong consistent broadcast messages resolve problem 2.