CS514: Intermediate Course in Operating Systems

Professor Ken Birman
Vivek Vishnumurthy: TA

Recap

- We've started a process of isolating questions that arise in big systems
  - Tease out an abstract issue
  - Treat it separate from the original messy context
  - Try and understand what can and cannot be done, and how to solve when something can be done

This week

- We'll focus on real time
- Basic issue: How can time be be "used" in systems
  - How can we synchronize clocks?
  - How can we use time in protocols?
  - In these kinds of systems, time has many kinds of limitations. What implications do they have for real-world applications?

What time is it?

- In distributed system we need practical ways to deal with time
  - E.g. we may need to agree that update A occurred before update B
  - Or offer a "lease" on a resource that expires at time 10:10.0150
  - Or *guarantee* that a time critical event will reach all interested parties within 100ms

But what does time "mean"?

- Time on a global clock?
  - E.g. with GPS receiver
- ... or on a machine's local clock
  - But was it set accurately?
  - And could it drift, e.g. run fast or slow?
  - What about faults, like stuck bits?
- ... or could try to agree on time

Reminder: Lamport's approach

- Leslie Lamport suggested that we should reduce time to its basics
- He defined the happens before relation and introduced a concept of logical clocks:
  - If $a \rightarrow b$, then $LT(a) < LT(b)$
- Schmuck: Extended to vector clock:
  - $a \rightarrow b$ *if and only if* $VT(a) < VT(b)$

Rules for comparison of VTs

- We'll say that $VT_A = VT_B$ if
  - $\forall_I$, $VT_A[i] = VT_B[i]$
- And we'll say that $VT_A < VT_B$ if
  - $VT_A = VT_B$ but VT_A ? VT_B
  - That is, for some i, $VT_A[i] < VT_B[i]$
- Examples?
  - [2,4] = [2,4]
  - [1,3] < [7,3]
  - [1,3] is "incomparable" to [3,1]

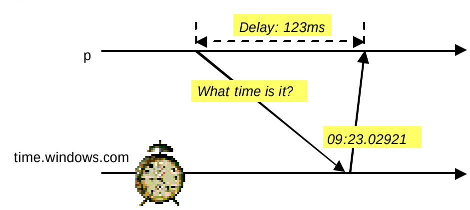
Introducing "wall clock time"

- There are several options
  - "Extend" a logical clock or vector clock with the clock time and use it to break ties
    - Makes meaningful statements like "B and D were concurrent, although B occurred first"
    - But unless clocks are closely synchronized such statements could be erroneous!
  - We use a clock synchronization algorithm to reconcile differences between clocks on various computers in the network

Synchronizing clocks

- Without help, clocks will often differ by many milliseconds
  - Problem is that when a machine downloads time from a network clock it can't be sure what the delay was
  - This is because the "uplink" and "downlink" delays are often very different in a network
- Outright failures of clocks are rare...

Synchronizing clocks

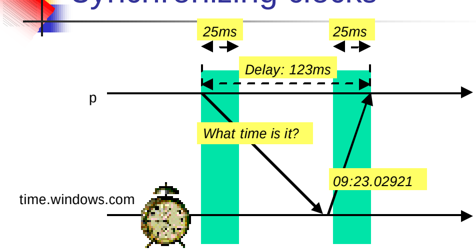


- *Suppose p synchronizes with time.windows.com and notes that 123 ms elapsed while the protocol was running... what time is it now?*

Synchronizing clocks

- Options?
  - P could guess that the delay was evenly split, but this is rarely the case in WAN settings (downlink speeds are higher)
  - P could ignore the delay
  - P could factor in only "known" delay
    - For example, suppose the link takes at least 25ms in each direction...

Synchronizing clocks



- *Suppose p synchronizes with time.windows.com and notes that 123 ms elapsed while the protocol was running... what time is it now?*

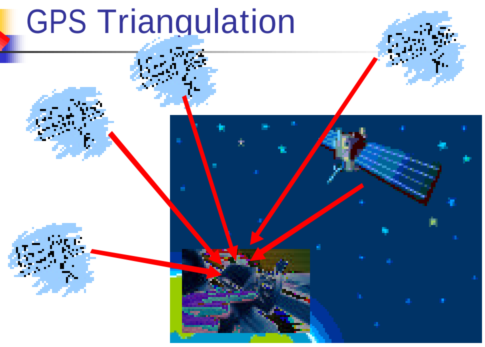
Synchronizing clocks

- In general can't do better than uncertainty in the link delay from the time source down to p
  - Take the measured delay
  - Subtract the "certain" component
  - We are left with the uncertainty
- Actual time can't get more accurate than this uncertainty!

What about GPS?

- GPS has a network of satellites that send out the time, with microsecond precision
- Each radio receiver captures several signals and compares the time of arrival
- This allows them to triangulate to determine position

GPS Triangulation



Issues in GPS triangulation

- Depends on very accurate model of satellite position
  - In practice, variations in gravity cause satellite to move while in orbit
- Assumes signal was received "directly"
  - Urban "canyons" with reflection an issue
- DOD encrypts low-order bits

GPS as a time source

- Need to estimate time for signals to transit through the atmosphere
  - This isn't hard because the orbit of the satellites is well known
  - Must correct for issues such as those just mentioned
- Accurate to +/- 25ms without corrections
- Can achieve +/1 1us accuracy with correction algorithm, if enough satellites are visible

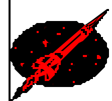
Consequences?

- With a cheap GPS receiver, 25ms accuracy, which is large compared to time for exchanging messages
  - 10,000 msgs/second on modern platforms
  - ... hence .1ms "data rates"
  - Moreover, clocks on cheap machines have 10ms accuracy
- But with expensive GPS, we could timestamp as many as 100,000 msgs/second

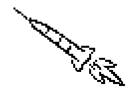
Accuracy and Precision

- *Accuracy* is a measure of how close a clock is to “true” time
- *Precision* is a measure of how close a set of clocks are to one-another
  - Both are often expressed in terms of a window and a drift rate

Thought question



- We are building an anti-missile system
- Radar tells the interceptor where it should be and what time to get there
- Do we want the radar and interceptor to be as accurate as possible, or as precise as possible?



Thought question

- We want them to agree on the time but it isn’t important whether they are accurate with respect to “true” time
  - “Precision” matters more than “accuracy”
  - Although for this, a GPS time source would be the way to go
    - Might achieve higher precision than we can with an “internal” synchronization protocol!

Real systems?

- Typically, some “master clock” owner periodically broadcasts the time
- Processes then update their clocks
  - But they can drift between updates
  - Hence we generally treat time as having fairly low accuracy
  - Often precision will be poor compared to message round-trip times

Clock synchronization

- To optimize for precision we can
  - Set all clocks from a GPS source or some other time “broadcast” source
    - Limited by uncertainty in downlink times
  - Or run a protocol between the machines
    - Many have been reported in the literature
    - Precision limited by uncertainty in message delays
    - Some can even overcome arbitrary failures in a subset of the machines!

Adjusting clocks: Not easy!

- Suppose the current time is 10:00.00pm
  - Now we discover we’re wrong
  - It’s actually 9:59.57pm!
- Options:
  - Set the clock back by 3 seconds…
    - But what will this do to timers?
    - Implies a need for a “global time warp”
  - Introduce an artificial time drift
    - E.g. make clock run slowly for a little while

Real systems

- Many adjust time "abruptly"
  - Time could seem to freeze for a while, until the clock is accurate (e.g. if it was fast)
  - Or might jump backwards or forwards with no warning to applications
- This causes many real systems to use relative time: "now + XYZ"
  - But measuring relative time is hard

Some advantages of real time

- Instant common knowledge
  - "At noon, switch from warmup mode to operational mode"
  - No messages are needed
  - Action can be more accurate that would be possible (due to speed of light) with message agreement protocols!

Some advantages of real time

- The outside world cares about time
  - Aircraft attitude control is a "real time" process
  - People and cars and planes move at speeds that are measured in time
  - Physical processes often involve coordinated actions in time

Disadvantages of real time

- Weeks ago, we saw that causal time is a better way to understand event relationships in actual systems
  - Real time can be deceptive
  - Causality can be tracked... and is closer to what really mattered!
- For example, a causal snapshot is "safe" but an instantaneous one might be confusing

Internal uses of time

- Most systems use time for expiration
  - Security credentials are only valid for a limited period, then keys are updated
  - IP addresses are "leased" and must be refreshed before they time out
  - DNS entries have a TTL value
  - Many file systems use time to figure out whether one file is fresher than another

The "endless rebuild problem"

- Suppose you run Make on a system that has a clock running slow
  - File xyz is "older" than xyz.cs, so we recompile xyz...
  - ... creating a new file, which we timestamp
  - ... and store
- The new one may STILL be "older" than xyz.cs!

Implications?

- In a robust distributed system, we may need trustworthy sources of time!
  - Time services that can't be corrupted and won't run slow or fast
  - Synchronization that really works
  - Algorithms that won't malfunction if clocks are off by some limited amount

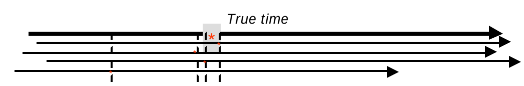
Fault-tolerant clock sync

- Assume that we have 5 machines with GPS units
- Each senses the time independently
- Challenge: how to achieve optimal precision and accuracy?

Srikanth and Toueg

- You can't achieve both at once
  - To achieve the best precision you lose some accuracy, and vice versa
- Problem is ultimately similar to Byzantine Agreement
  - We looked at this once, assuming signatures
  - Similar approach can be used for clocks

Combining "sensor" inputs



- "Shout at 10:00.00"

Combining "sensor" inputs

- Basic approach
  - Assume that no more than k out of n fail
    - Depending on assumptions, k is usually bounded to be less than n/3
  - Discard outliers
  - Take mean of resulting values
- Attacking such a clock?
  - Try and be "as far away as possible" without getting discarded

How do real clocks fail?

- Bits can stick
  - This gives clocks that "jump around"
- The whole clock can get stuck, perhaps erratically
- Clock can miscount and hence drift (backwards) rapidly

Summary

- Very appealing to use time in distributed systems
- But doing so isn't trivial
  - We need clock synchronization software or GPS... and even GPS can fail (it can break, or can have problems due to environment)
  - Fault-tolerant clock synchronization is hard
- Clocks in real systems can jump around... even on "correct" machines!

For next time

- Read the introduction to Chapter 14 to be sure you are comfortable with notions of time and with notation
- Chapter 22 looks at clock synchronization