CptS 464/564
Lecture 8

6th February 2003

Time and Clocks (Sec. 2.5)

Today we talk about physical and virtual clocks, which relate time in a computer system to global time standards such as UTC and TAI which are based on physical phenomena. That is today we talk about *real time.* Next time (Sec. 2.7) we will talk about *logical time* which is concerned only with the order in which events happen in a system, and *not* with the their relationship to physical time. Read Sec. 2.7 for next time.

(Aside: debate rages about the reality of real time. Philosophers and physicists argue about whether time exists independent of the way we experience it. You can find an introduction to the debate in the Sept. 2002 issue of *Scientific American.)*

Real time: an abstract, monotonically increasing, continuous function that marks the passage of time.

To *timestamp* an event is to associate it with a particular point on a real time line. Once timestamp, the stamp can be used to relate the event to other events as to order, and duration of intervals between events.

Interaction between programs and time is provided by *timers* and *clocks.* I find it interesting that *computation*, understood as anything that can done by a Turing machine, and *time* seem to be completely separate ideas: a TM can't keep time, and a clock can't compute. Yet from the earliest days of digital computing, computers have been intimately associated with clocks.

Time examples in DS

File timestamps: we expect that if file B is updated after file A, then modtime(B)>modtime(A). If different computers perform the updates, we still expect this to hold. Otherwise things break, e.g. *make.*

How long does it take for a message to go from computer A to B and back to A? Easy to answer with considerable accuracy? With such a *close time chain* questions of duration can all be related to a single clock.

How long does it take for a message to go from computer A to computer B? Hard to answer accuately. *Distributed durations* that involve multiple clocks are hard. We'll see later today how to estimate distributed durations and how good an estimate we can make.

Global time, such as provided by the Global Positioning System, can improve both the ease and accuracy of such measurements. As more of our systems come to depend on GPS it may become yet another *critical infrastructure:* one such as electricity, energy, transportation, that continued functioning of the economy relies upon. Thus we have to worry when deploying systems using GPS about the reliability and availability of the service, and ask ourselves "what if suddenly it weren't there for a few hours, days, weeks or months?"

Clock basics - local physical clocks

Model: hardware implementing a monotonically increasing discrete function $pc_k(t)$mapping real time into clock time at node k. A physical clock is characterized by its granularity, g, and its drift rate, ρ. I find the notation of the book in Tables 2.1 and 2.2 to be insufficiently precise for understanding, so I will use an alternate notation to express the concepts related there.

Granularity is the notion that the clock ticks in discrete steps, unlike real time which is continuous. Each discrete step is *intended* to represented the passage of a certain amount of time, which we'll call $\overline{g}$. Thus for any two real times $t_0$and t_1,

$$pc_k(t_1) - pc_k(t_0) = n\overline{g}$$

for some integer n.

Drift rate is the notion that computer clocks do not run at exactly the same rate as the atomic standard clocks that define real time. To express this notion let $t(pc_k^i)$ be the real time at which pc_k first reads i. The physical drift rate ρ_k^p is a number such that

$$0 \leq 1 - \rho_k^p \leq \frac{t(pc_k^{i+\overline{g}}) - t(pc_k^i)}{\overline{g}} \leq 1 + \rho_k^p$$

for all physical clock readings, i.

For *good* clocks typically found in computers, ρ^p is typically on the order of one part in 10^{-5}, a few parts per million, a few tens of milliseconds per hour or a few seconds per month. Some computers have *much worse* clocks, varying from real-time by even minutes per day.

Local clocks are good for timestamping local events, measuring local durations, and measuring round-trip durations, but are not good for measuring distributed durations or providing globally valid timestamps.

Global virtual clocks

One good thing about distributed computing is that well-behaved systems may be able to keep less-well-behaved systems in line. Clock synchronization algorithms can keep clocks synchronized system-wide.

We're not going to study how to build globally synchronized virtual clocks, but it is important that you have some sense of how they are characterized. Let vc_k be the virtual clock mapping real time to virtual time at site k.

We will distinguish between internally synchronized clocks and externally synchronized clocks. *Internal synchronization* refers to how well the clocks in a system agree with each other. *External synchronization* refers to how well they agree with an external time reference (such as GPS).

The key attributes for internal synchronization are the *granularity*, *convergence* δ^v, the *precision* π^v, and the *drift* ρ^v. Drift is defined analogously to the case for a physical clock and refers to how fast or slow the virtual clock runs.

Convergence is a property of the algorithm that synchronizes the clocks as well as the physical clocks on which it is based and the network on which it runs. It tells us how well the algorithm is able to synchronize the clocks by telling us how close they are immediately after synchronization is performed at time 0.

$$\left|t(vc_k^0) - t(vc_l^0)\right| \leq \delta^v$$

for any two sites k and l.

Precision describes how closely virtual clocks remain synchronized while the system runs. It depends on how fast physical clocks drift as well as how often the clock synchronization algorithm is run and how good the convergence is. The global virtual clock exhibits precision π^vif

$$\left|t(vc_k^i) - t(vc_l^i)\right| \leq \pi^v$$

for any clock reading i, and any two sites k and l.

Externally synchronized system of clocks are characterized by an additional parameter, their *accuracy*, α^v. The global virtual clock exhibits accuracy α^v if

$$\left|t(vc_k^i) - i\right| \leq \alpha^v$$

at all sites k and at all times i.

Illustrate each of these concepts with fig. 2.7.

Measuring distributed durations with global clocks

Use Figure 2.8(a).

We want to figure out how long message m_1 took to go from p to q. Call that delivery time $T_D(m_1)$. If p puts the timestamp T_{p_1} into the message, q can estimate the desired time as $T_D(m_1) \cong T_{q_1} - T_{p_1}$. Which parameters of the global clock tell us how good this estimate is? Most important are granularity and precision (assuming the delivery time is short). Based only on hte precision, ignoring granularity and drift) how accurate is the estimate?

Measuring distributed durations with only local clocks

Use Figure 2.8(a) again.

Notice how in this instance a prior message from q to p is required to "close the timing loop". In the absence of such a message, q has no reference point for T_{p_1}. With a prior message, observe that $T_D(m_0) + (T_{p_1} - T_{p_0}) + T_D(m_1) = (T_{q_1} - T_{q_0})$, in which we subtract only local clock values. Rearranging we get

$$T_D(m_1) = (T_{q_1} - T_{q_0}) - (T_{p_1} - T_{p_0}) - T_D(m_0)$$

where $T_D(m_0)$ is unknown. However, if we know a minimum delivery time for messages, call it $T_{D_{min}}$, we will know that $T_{D_{min}} \leq T_D(m_1) \leq (T_{q_1} - T_{q_0}) - (T_{p_1} - T_{p_0}) - T_{D_{min}}$.

Thus, if p includes $(T_{p_1} - T_{p_0})$ with message m_1, q will be able to estimate $T_D(m_1)$.

This is example is illustrative of the kinds of techniques that are used in clock synchronization algorithms.

Logical clocks

In addition to the local physical clocks and global virtual clocks that we have discussed, there is another kind of "clock" commonly discussed in the DS literature, namely *logical clocks.* We will take up logical clocks in the section on ordering.

Synchrony (2.6)

First thing to be aware of: the terms *synchronous* and *asynchronous* are used with several different meanings in computer science and even within distributed systems research.

- Message passing: if the send and recv primitives are blocking we call it synchronous message passing; if they are non-blocking we call it asynchronous.

- Hardware digital systems: clock-driven systems are called synchronous; sometimes refers to whether the clocks in two interconnected digital systems are synchronized. E.g. Asynchronous serial communication.
- Groupware systems: whether participants interact at the same time (synchronous – chat) or at different times (asynchronous – email)
- Distributed systems theory: time delays are bounded (synchronous) or unbounded (asynchronous). This last sense of the term is what we cover now.

Definition: a *synchronous system* is one where it is possible to bound the delays in communication and processing. An *asynchronous* does not have such bounds, and therefore actions and communication may take an arbitrarily long time. For example, in a synchronous communication system we will have some value $T_{D_{max}}$ that bounds the time taken for a message to be delivered after it is sent.

Consequence: in a synchronous system if a message is expected by a certain time and it doesn't arrive, then one can conclude that it wasn't sent (or that the assumption of synchronicity was not warranted.)

Consequence: in an asynchronous system, *nothing* can be inferred from the non-arrival of a message by a certain time. You can't conclude it was lost, or that it wasn't sent. It might arrive a millisecond from now, a century from now, or never. This property leads to some rather startling impossibility results in asynchronous systems.

If synchrony is so useful, why not always build synchronous systems? You have to pay for it, and what you have to pay for it you might not be willing to give up: e.g., sharing, openness, scalability, interactivity. Exercise: why does each of these have to be given up, at least partially, to achieve synchrony?

We skipped 2.6.2 and 2.6.4. You are not responsible for that material.