CptS 464/564
Lecture 9

21st September 2004

Ordering (2.7)

In section 2.5 we looked at real time and the issues associated with achieving globally synchronized clocks. We now turn to a more abstract view of time that is concerned only with the order in which events occur and not with the intervals between them. In many cases what we care about is *causality* or *potential causality:* could event B have been caused or affected by event A? Answering this question has importance not only for debugging distributed systems, but also for their design. One thing to pay attention to later as we talk about distributed systems is what message ordering properties are required of an underlying system in order to argue that a particular algorithm is correct. For example, a particular distributed algorithm may only work correctly if messages are delivered in FIFO (first in first out) order.

Analyzing Causality

As noted above, we are often most interested in events where there is a causal relation between them: did event A somehow cause event B, or B cause event A? In after-the-fact analysis of a system's behavior, for example from a log, such causally related events will be the ones that give us the most insight into why the system behaved as it did. In a running system, receiving a message B before receiving message A, which caused B, *may* lead to an incorrect outcome. So causality is an (the most?) important reason for caring about order.

Without complete understanding of the code of a processs that receives a message and then sends another message, it is impossible to say whether receiving the first message caused the second to be sent. However, the following notion seems to be a basic property of our notion of causality:

- An event cannot be caused by receipt of a message after the event occurred.

Partial and total orders

Recall that the events we are concerned with are a process sending a message, a process receiving a message, and internal action by a process. The set of all possible events is an example of a *domain.*

Definition: an *irreflexive partial order*, R, on a domain, D, is a relation such that

$$\forall x,y \in D \bullet ((x,y) \in R \Rightarrow (y,x) \notin R)$$

$$\forall x,y,z \in D \bullet ((x,y) \in R \wedge (y,z) \in R) \Rightarrow (x,z) \in R$$

The first rule ensures that no element is related to itself and that no two elements are related in both directions. Notice that there is no requirement in a partial order that every two elements be related in one direction or the other. However, we have a definition that encompasses that possibility, namely a *total order,* which is a partial order in which every two distinct elements are related. That is R is a total order if R is a partial order and

$$\forall x,y \in D \bullet x = y \vee (x,y) \in R \vee (y,x) \in R$$

Examples: $<$on the domain of integers is a total order. *is-a-proper-subset-of* on sets of integers is a partial order (no set is a proper subset of itself, two sets can't be subsets of one another, and many pairs of sets are neither a subset of the other. Notice that any partial order on a domain can be extended to a total order by arbitrarily but consistently ordering the elements that are left unordered by the partial order (see Topological Sort, Knuth v. 1)

We are interested now in orderings that approximate the impractical partial order that we'll call *truly-causal. (*Impractical because we can only know it by looking inside the programs that make up the distributed system.)

Several different approximations to truly-causal ordering

We will examine in turn several different kinds of ordering and mechanisms that are used to achieve them.

- Physical order
- (Logically) happened before or *precedence*
- FIFO order

Physical order

Physical order: the order by which events occurred in a real-time timeline as seen by an omniscient observer. This seems to be a natural order because it is one that we (think that we) directly experience in everyday life. As we have seen in our earlier demonstrations, this experience may not be a good guide to the behavior of systems where message delivery is not instantaneous. Physical order is an approximation to truly-causal order because it respects the requirement that nothing can cause something that happened before it. That is if event a truly caused event b, then the time of a will be before the time of b.

Physical order isn't a very *good* approximation to truly-causal order because physical too many events are ordered by a physical order: the time of a is before the time of b does not necessarily imply that a caused b. (Give an example.) Also, note the undesirable dependence on an omniscient observer or synchronized real-time clocks.

Logically happened before or *precedence*

A better approximation to true causality comes from observing the actual flow of messages in an execution of a system. It is clear that if two events occur in the same process, one *happened before* the other. We will say that one precedes the other. It is also clear that the sending of a message must *happen before* it is received. When the relation defined by these two characteristics is *transitively closed*, the result is the *happens before* relation. We will use the symbol $\rightarrow$ to mean *happens before*, and note that $\rightarrow$ defines a partial order on events. $\rightarrow$ suggests a defintion for a new relation: if neither $a \rightarrow b$ nor $b \rightarrow a$ we will say that a and b are *concurrent events* and we write $a||b$. Observe that $||$ is *not* transitive. If $a \rightarrow b$ we will also say $b \leftarrow a$, *i.e.* b happens after a.

Precedence is a better approximation to true causality: if a truly caused b then $a \rightarrow b$, but again it is still an approximation because $a \rightarrow b$ does not necessarily imply that a truly caused b. But precedence does allow us to observe concurrency of *some* events, where physical order did not.

Other names: Causal ordering, potential causal ordering, Lamport ordering

Source: Lamport, Leslie. "Time, Clocks and the Ordering of Events in a Distributed System," Communications of the ACM, Vol. 21, July 1978, pp. 558-565. This is a classical paper in computer science and is assigned reading for students in 564.

Ordering guarantees in communication systems

As noted above, if messages are received by processes in an order that violates true causality, wrong results may be computed. Example: suppose you and I are working together on a document. You copy the text of your current version into email that you send to me. I work on the document some more and then save it to a shared file server.

You then save it to the same file. Your save overwrites mine and the final good version of the document is lost because the receipt of the content of the document at the file server didn't obey causality.

Can we implement communication systems that respects causality. Yes. Both of the discussions above concerning physical ordering and happens before suggest how such a mechanism could be implemented. But first we'll look at a simpler approach.

FIFO ordered communication systems

A FIFO ordered communication system obeys the property that for any two processes p and q and any two messages $m_1$and m_2 sent from p to q, if $m_1$is sent before m_2, $m_1$is delivered before m_2.

FIFO ordering is easy to implement: the sender puts a sequence number in each message. Messages are delivered to the recipient in the order of sequence numbers, not the order of arrival. (Exercise: Give a design for the receiver that makes this possible)

FIFO works great for pair-wise interactions. Prove to yourself that FIFO respects happens-before if there are only two participants.

When does it break down? (Fig. 2.14b) Example of FIFO delivery that does not respect happens-before.

Causally ordered delivery

Causal *not* Casual!

Perhaps a bad name, because only potential causality is captured, causal delivery respects the *happens before* relation previously defined. That is, for any two messages $m_1$and m_2 sent to the same site, r, regardless of source, if $send(m_1) \rightarrow send(m_2)$ then $deliver(m_1) \rightarrow deliver(m_2)$. To implement causal delivery, we need to track the precedence relation at each site and communicate it in the message headers. **How to do this is covered in section 2.7.6 for which graduate students are responsible.**

Example: Fig. 2.15a

Breakdown of causal delivery: Fig 2.16b. Some algorithms don't work correctly using only causally ordered delivery.

Totally ordered delivery

In some situations, such as when we want executions to be reproducible, or when we want computations to be exactly replicated at multiple sites, it is important that messages be delivered in the same order at every site. That is, for all messages $m_1$and m_2 and all sites p and q such that p and q both receive $m_1$and m_2,

$$deliver_p(m_1) \rightarrow deliver_p(m_2) \Rightarrow deliver_q(m_1) \rightarrow deliver_q(m_2)$$

That is, every site receives messages in the same order. Convince yourself that the example of 2.16b works correctly if messages are totally ordered.

Exercise: give examples of

- totally ordered delivery that is not causal
- causal delivery that is not totally ordered

Total ordering is important when multiple sites must maintained replicas of one another's states. The key observation is the states are to be the same the different sites must extend the partial happens-before relationship to a total order *in the same way at all the sites.*

Temporally ordered delivery

As we have seen, causally ordered delivery isn't sufficient if processes communicate outside the ordered delivery system. This prompts us to look back again at physical ordering as a possible basis for our ordered delivery system. Recall that the problem with physical order was that it ordered many more pairs of events than were truly causally related. We can improve on physical order with the observation that most message delivery systems have a minimum delivery time for messages, call it δ. Thus an event occuring at time t cannot be caused by a message sent after time $t - \delta$. This observation induces us to define the relation *temporally precedes*, written $\rightarrow^{\delta}$, and defined

$$m_1 \rightarrow^{\delta} m_2 \Leftrightarrow t(send(m_2) - t(send(m_1)) > \delta$$