Logical time and Vector Clocks

Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems.

Announcements

- Please set a *zoom photo* of yourself
- Make sure you're on Piazza
- Project details coming soon, but good to start thinking about what you may want to work on
  - *Project speed-dating in class (bring 2 ideas)*
  - *Long project proposal stage (lots of writing)*

Intro

- vClocks used to address causality problem
- **is a causally related to b?** i.e., can a influence b or vice versa
- Why do we care about time?
- To determine **global state** of the system, we need time. Without it, we don't know the ordering of events that shape the global state of the system.
- Dist systems are hard to program. Trad. reality (?) expects certain order or invariants. Dist. systems don’t have this. So manipulating time helps to generate abstractions that match this context.
- If you don't know time, you can override recent changes made by another node (tracking changes)
- Distinguishing old and new … tracking **events**!
- Dist. System = CODE + ENVIRONMENT. ENV is typically non-deterministic => ordering of events is unreliable/cannot be predicted ahead of time.

Intro

- vClocks used to address causality problem. Determine global order across all processes.

- vClock is a timestamp

- *vClock is a record of a processor's point of view of what it could have been influenced by*

- vClock is local knowledge of other processes execution "time" (# steps)

Intro

- *vClock is a record of a processor's point of view of what it could have been influenced by*
- Upsetting issues:
  - *Scalability*: vClocks require linear space in nodes in the system
  - More nodes => bigger vClock
  - bigger vClock => more **bandwidth**
  - Can't offload the bandwidth cost off of the node

Logical clocks

- What are they, and why do we need them?

Logical clocks

- Why logical and not physical?
- Logical clock requires less space (?)
- *Msg latency* is a problem: phys clocks measure real time intervals. Msgs “use” real time, but an indeterminate amount => if you broadcast a phys timestamp then (1) it will be wrong on receipt (late), and (2) unpredictably late.
- Phys clocks could order local events. If I only have one local phys clock, I can use it to order local non-deterministic events (multi threaded/multi core).
- You can synchronize the p. clocks! *But then you have to decide on accuracy (have to decide what is good enough)*

Logical clocks

- If my activity has a faster resolution than my p. clock, then I'm in trouble (won’t get an exact timestamp)
-

Application of v. clocks

- Causal bcast, message stability, message pattern detection
- Writing consistency: Perfect examples to illustrate why v clocks useful
- What are these white and black events? Black events are ones that you care about (e.g., you only care about database operations). I only want to reason about the ordering between DB ops. Choosing the right logical time abstraction for the problem.
- Why is this pattern thing a debugging scenario? *Pattern ~ mental model* of what should happen. Pattern detection ~ testing your mental model against what happened. Imagine facebook with person logs in -> DB logs -> analytics system (relating triples of events)
- v clocks reduce the complexity of the system to an abstraction && also reduce complexity to *comparison* between v clocks.
- MC is the immediate past of events for an event
- Q: what if the middle event is / is not a communication event, would you use the same alg? (Ivan: this doesn't matter).

Application of v. clocks

- Causal bcast, message stability
- Causal b cast ~ TCP multicast
- Do you implement bcast at application layer or not?
    - TCP cast : transport abstraction
    - Causal b cast : application-level abstraction
- Do we need causal bcast? (when would you use it, would you ever use it, what would use it for?)
- msgs from different nodes arrive in same order
- Example: Google docs editing — edits should be ordered the same across all of our Gdoc sessions (Google uses operational transform: https://en.wikipedia.org/wiki/Operational_transformation)
- More fault tolerance systems: all processes need to know state of other processes (alive or dead)

Application of v. clocks

- Msg is *stable* if every node has received it
  - This is an essential part of (reliable msging) protocols like TCP
- Use v clocks that every node has received a msg from some processor
- Why? Their motivation: determine when to run *discard(m) : frees up m from the msg buffer*
- Keep msg until you know for sure that no one else needs the msg. If some node needs a msg, make sure that the msg exists in the system (and can be delivered to that node).
- Don’t I hold onto it forever? There is a separate protocol that re-sends msgs to nodes that have missed them. It’s not in the paper. Nice **exercise**: add this protocol to msg stability description.

Closing thoughts

- Theoretical concept? NO! But.. yes, sort of.
- Libraries that implement it: https://github.com/DistributedClocks -> Use to inspect/visualize the resulting log https://bestchai.bitbucket.io/shiviz/
- Used to associate timestamps with data objects and operations on data
- *Key takeaway: Partial ordering view of distributed systems.*