Chapter 5

Limitations of a Distr. System

- Lack of global clock
    - common clock ? Synchronized clocks ?
- Absence of shared memory
    - cannot obtain a “coherent” view of “global” state
    - coherence ==> state observations made at the same time.

Temporal fundamentals

- *Happened before* relation (-->)
  - a --> b iff
    - a occurred before b in the same process
    - a is the event of sending a message in a process and b is the event of receiving the same message by another process
  - --> is transitive
  - a can causally affect b if a --> b
  - if ! ((a --> b) and (b --> a)) then a || b (concurrent). a and b do not have a causal relationship.

Lamport's Logical Clocks

- Conside a "clock" Ci associated with process Pi. It is simply a process which assigns a number Ci(a) to any event a in the process such that C(a) < C(b) if a --> b
  - Ci(a) < Ci(b) if a and b in the same process and a --> b
  - Ci(a) < Cj(b) if a is send(m) in Pi and b is recv(m) in Pj
- To make the above true
  - Ci should monotonically increase between successive events within a process (Ci = Ci + d)
  - every message sent is stamped with the Ci of the sending process. On receipt, the receiver sets its Cj to the greater of its present value or the received timestamp (max (Cj, tstamp+d)
- This can be thought of as "virtual" time, but it moves only in response to events.

Limitations

– Since each clock can “independently” advance, we cannot in general infer happened before, and hence causality from clock value relations

Vector Clocks

- Each process maintains a vector C of size n, where n is the number of processes in the system.
- For process i, the i^{th} entry of the vector is the local clock. The other entries represent its best guess of the clock at other processes.
  - When an event occurs at a process i, Ci[i] is incremented.
  - When a message is sent, it is time-stamped (with the vector clock). Upon receipt by process j, Cj is updated as
    - forall k, Cj[k] = max (Cj[k], tmstamp[k])
- Every process has the most up to date knowledge of its clock (forall i,j, Ci[i] >= Cj[i])

- Two vector timestamps are equal iff all their components are equal, unequal if even one component differs.
- Less than or equal to iff each component is less than or equal to, not LTE if even one component is greater.
- Less than iff (LTE AND not EQ) => if at least one component is smaller
- Not less than iff not(LTE and NEQ)
- Concurrent iff ((a NLT b) AND (b NLT a))
- LT E specifies a partial order (but concurrency does not)
- Note that now, --> iff (a LT b)

Causal Ordering of Messages

- If M1 is sent before M2, then every recepient of both messages must get M1 before M2
  - underlying network will not necessarily give this guarantee.
- Consider a replicated database system. Updates to the entries should be received in order!
- Basic idea -- buffer a later message

Birman-Schiper-Stephenson Protocol

- Assumes that communication is via broadcasts
- Pi stamps outgoing messages with a vector time
- Pj, upon receiving a message from Pi VTm buffers it till
  - VTpj[i] = VTm[i] - 1 AND forall k, k != i, VTpj[k] >= VTm[k]
- When Pj receives a message, it updates VTpj

Schipper-Eggli-Sandoz Protocol

- Each process maintains a vector VP of size N-1. The elements are tuples (Pj,t), where Pj is the destination of a message, and t the time the message was sent.
  - Send:
    - Send message with timestamp tm and VP to Pk
    - insert (Pk, tm) into VP
  - RECV:
    - If VM does not contain any tuple with Pk, OR tm <= tlocal then receive else buffer
    - Upon Receipt
      - » Merge VM with VPk
      - » Update P2's logical Clock
      - » Check for buffered messages that can be delivered.

Global State

- Due to absence of global clock, states are recorded at different times
- For global consistency, state of the communication channel should be the sequence of messages sent before the sender’s state was recorded minus the messages received before the receivers state was recorded.
- Local states are defined in context of an application
  - a send is a part of the local state if it happened before the state was recorded. Ditto for a recv.

- A message causes an inconsistency if it was received, but not sent
- A collection of local states forms a global state
- This global state is consistent iff there are no pairwise inconsistency between local states.
- A message is in transit when it has been sent, but not received.
- The global state is transitless iff there are no local state pairs with messages in transit.
- Transitless + Consistent ➔ Strongly Consistent State

Chandy Lamport Algorithm

- The “initiating” process sets up a marker and records its state. It then sends the marker out on each outgoig channel BEFORE it sends any message.
- When a marker is received
  - if your state has not been recorded, record channel state as empty, record your state, forward marker
  - otherwise, record the state of the channel as all messages recd after recording of state but before receiving marker.
  - Assumes FIFO channels.
  - The recorded state may not be identical to any of the actual states of the system !

Cut of a Distr. Computation

- A set of cut events at individual sites
- Is consistent iff every message that was received before a cut event was sent before the corresponding cut event at the sender
- ==> cut events are not causally related
- VTc = sup(VTc1, VTc2, … VTcn)
- If cut events are not causally related, then we can show that
  VTc = (VTc1[1], VTc2 [2], … VTcn[n])

Termination Detection

- When has a distributed computation terminated
  - Instance of getting a consistent global state
- System mode -- process is either active or idle, and can delegate computation tasks
- Huang’s algorithm uses currency distribution notions. The initiator has a fixed amount of currency. When it delegates tasks, it distributes currency. When the delegated task is done, currency is returned. When originator has all currency back then computation is terminated.