Distributed Systems

Kai Shen

Parallel Computing vs. Distributed Systems

- Parallel computing – compute sub-tasks simultaneously so that work can be completed faster.
- Distributed systems – a set of autonomous computers working together to appear as a single coherent system (and to achieve unified goals). Examples:
  - client-server web transactions
  - networked file systems
  - a data center with hundreds of thousands of machines
  - crowdsourcing among many smartphones
  - … …
- Parallel computing more identified with what it does
- Distributed systems more identified with what they are
- Not mutually exclusive
  - MPI and MapReduce

Distributed Systems Model

- Study of general approaches requires a system model
- Autonomous computers
- Network connecting all computers (all-to-all reachable)
- Communications
  - Messages
  - Streams
- Failures
  - Messages may be lost
  - Nodes may stop working
  - Or worse!

Distributed Systems Overview

- Time and order
  - Challenging but useful in distributed systems
- Fault tolerance
  - Make consistent decisions when nodes may fail
- Replication and consistency
  - Consistent replication with updates: Does each replica get all the updates? Are updates applied in the same order?
- Scalable Internet systems
  - Data centers, cloud computing, distributed data store (Bigtable etc.)

Time and Clocks

- Why is it important?
  - Determine the order of events occurred on different computers
- Examples:
  - In “make”, source files who have not been changed since last compile don’t have to be recompiled
    - Compilation and editing on different computers
  - Two distributed updates to the same location need to be ordered to know the final state
  - More examples?

Physical Clocks

- How do we get time?
- Fixed-frequency events
  - Quartz crystal oscillates at fixed frequency to triggered timer interrupts
  - Atom vibrates (makes state transitions) at fixed frequency
- Consistent for a single timing device
- But drifts/skews common over multiple devices in a distributed system

Physical Clock Synchronization

- Clock synchronization
  - Identify clock skewness and correct it by adjusting clock
  - No turning back in time
- Cristian’s algorithm
  - A client requests time from a server, the server responds with current time
  - Delay in response – estimate as half of request/response time
- Averaging of everyone’s time
  - A central server polls everybody and does the averaging

Distributed Clock Synchronization

- Basic mechanisms: broadcast or poll
  - Everyone (or some clock source) broadcasts its time periodically
  - A node can poll clock sources for the time
- Approach
  - Calculate clock drifts from multiple sources and averages them
  - Marzullo's algorithm: the best estimate is taken to be the interval consistent with the largest number of sources
    http://en.wikipedia.org/wiki/Marzullo%27s_algorithm
    Also foundation of the Network Time Protocol

Precise Timing

- Atom clock
  - Atomic Clock FOCS-1 (Switzerland) started operating in 2004 at an uncertainty of one second in 30 million years.
  - http://en.wikipedia.org/wiki/File:FOCS-1.jpg
- GPS computes precise time
  - 10s of nanoseconds accuracy, but affected by Selective Availability

Logical Ordering

- Agreement on ordering of events (rather than the absolute time) is what matters
- Lamport ordering of distributed events
  - Inherent ordering: a → b, or "a happens-before b" holds regardless of runtime conditions (such as processor speed, machine overload, and message delays)
  - Specifically, a happens-before b
    - if a occurs before b on the same computer or a/b are send/receive events of the same message on two computers
    - transitive relation
- Total ordering vs. partial ordering

Lamport Logical Clock

- Assign timestamp to each event C(e) that follows "happens-before" partial ordering
  - Every computer maintains a local incrementing timestamp
  - Each message carries the sending time of the sender
  - Upon receipt of a message, the receiver clock is set to the greater of its own or the message timestamp + 1
- Lamport clock
  - follows "happens-before" but adds more ordering
  - is totally ordered?
    - Use machine ID as a secondary criterion to break ties

Utilization in Totally Ordered Broadcasts

- Totally ordered broadcasts
  - Update management in replicated databases
- Assumptions
  - FIFO messages between same sender/receiver pair
  - Messages are not lost
- Solution
  - Each message carries sender's timestamp
  - Received broadcast messages are buffered, acknowledged (in broadcast)
  - Deliver a message if
    - it has the earliest timestamp in buffer
    - and already acknowledged by everyone
  - Guarantee: will not receive a new message with earlier timestamp

Vector Timestamps

- Lamport logical clock follows but may go beyond the "happens-before" ordering
- Vector timestamp: n-element vector for n-computer system
  - At computer i:
    - VT[i] indicates the number of local events occurred so far (incremented after each local event)
    - VT[j≠i] = t means that computer i knows that t events have occurred at computer j (updated after each received message)
  - Each message carries the full timestamp, updates receiver
  - http://en.wikipedia.org/wiki/File:Vector_Clock.svg
  - Event with VT_x happens before event with VT_y if and only if every element in VT_x is less than or equal to corresponding element in VT_y

Utilization on Race Detection

- A distributed race condition may occur if two conflicting updates are not causally ordered (unordered by the vector clock)
  - Unordered updates may occur in either order depending on the runtime condition