CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

TIME AND ORDERING

Lecture A

INTRODUCTION AND BASICS

Why Synchronization?

- **You want to catch a bus at 6:05 pm, but your watch is off by 15 minutes.**
  - What if your watch is late by 15 minutes?
    - You'll miss the bus!
  - What if your watch is fast by 15 minutes?
    - You'll end up unfairly waiting for a longer time than you intended.
- **Time synchronization is required for both**
  - Correctness
  - Fairness

Synchronization In The Cloud

- Cloud airline reservation system
- Server A receives a client request to purchase last ticket on flight ABC 123.
- Server A timestamps purchase using local clock 9h:15m:32.45s, and logs it. Replies ok to client.
- That was the last seat. Server A sends message to Server B saying “flight full.”
- B enters “Flight ABC 123 full” + its own local clock value (which reads 9h:10m:10.11s) into its log.
- Server C queries A’s and B’s logs. Is confused that a client purchased a ticket at A after the flight became full at B.
- This may lead to further incorrect actions by C

Why is it Challenging?

- **End hosts in Internet-based systems (like clouds)**
  - Each have their own clocks
  - Unlike processors (CPUs) within one server or workstation which share a system clock
- **Processes in Internet-based systems follow an *asynchronous* system model**
  - No bounds on
    - Message delays
    - Processing delays
  - Unlike multi-processor (or parallel) systems which follow a *synchronous* system model

Some Definitions

- An asynchronous distributed system consists of a number of processes.
- Each process has a state (values of variables).
- Each process takes actions to change its state, which may be an instruction or a communication action (send, receive).
- An event is the occurrence of an action.
- Each process has a local clock – events *within* a process can be assigned timestamps, and thus ordered linearly.
- But – in a distributed system, we also need to know the time order of events *across* different processes.

Clock Skew vs. Clock Drift

- **Each process (running at some end host) has its own clock.**
- **When comparing two clocks at two processes**:
  - Clock Skew = Relative Difference in clock *values* of two processes
    - Like distance between two vehicles on a road
  - Clock Drift = Relative Difference in clock *frequencies (rates)* of two processes
    - Like difference in speeds of two vehicles on the road
- **A non-zero clock skew implies clocks are not synchronized.**
- **A non-zero clock drift causes skew to increase (eventually).**
  - If faster vehicle is ahead, it will drift away
  - If faster vehicle is behind, it will catch up and then drift away

How often to Synchronize?

- Maximum Drift Rate (MDR) of a clock
- Absolute MDR is defined relative to Coordinated Universal Time (UTC). UTC is the “correct” time at any point of time.
  - MDR of a process depends on the environment.
- Max drift rate between two clocks with similar MDR is 2 * MDR
- Given a maximum acceptable skew M between any pair of clocks, need to synchronize at least once every: M / (2 * MDR) time units
  - Since time = distance/speed

External vs Internal Synchronization

- **Consider a group of processes**
- **External Synchronization**
  - Each process C(i)'s clock is within a bound D of a well-known clock S external to the group
  - $|C(i) – S| < D$ at all times
  - External clock may be connected to UTC (Universal Coordinated Time) or an atomic clock
  - E.g., Cristian's algorithm, NTP
- **Internal Synchronization**
  - Every pair of processes in group have clocks within bound D
  - $|C(i) – C(j)| < D$ at all times and for all processes i, j
  - E.g., Berkeley algorithm

External vs Internal Synchronization (2)

- **External Synchronization with D => Internal Synchronization with 2*D**

- **Internal synchronization does not imply external synchronization**
  - In fact, the entire system may drift away from the external clock S!

Next

- Algorithms for clock synchronization