CSE535: Asynchronous Systems
Fall 2017

Distributed System Models

Scott D. Stoller

Stony Brook University

What are (Distributed System) Models?

- A model characterizes a class of systems.
- A good model includes all relevant aspects and ignores irrelevant details.
- Different models for different purposes, e.g., functional model, performance model, real-time model
- Sequential systems: less variety in the models
  - Theoretical models: Turing machine, lambda calculus
  - More detailed models: CPU, bus, RAM, disk
- Distributed systems: wider variety of models
- Dimensions:
  - Network Structure
  - Synchrony (Timing)
  - Failures

Network Structure

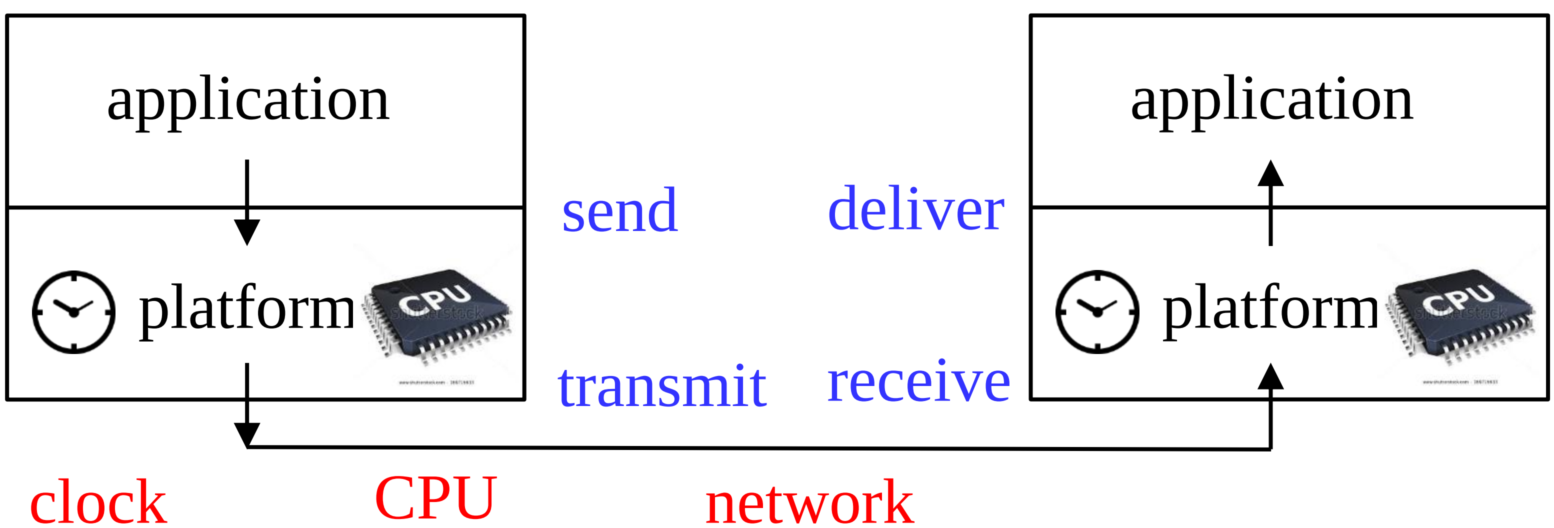
- Network structure: Pattern of interconnection of hosts
- Often represented by a graph.
- May be static or dynamic (e.g., reconfigurable).
- May be initially known to some or all hosts.
- May be connected or partitioned.
- Examples:
  - Bus, e.g., Ethernet
  - Ring, e.g., token ring, FDDI
  - Switched network, e.g., ATM, switched Ethernet
- Internetwork: set of connected networks
- Routing: finding a path through the network between two hosts.

Network Structure

- Multicast or Broadcast: send a message to multiple destinations at once.
- Hardware Broadcast: use of network-specific features to do broadcast more efficiently than sending to each destination separately.
  - Example: Bus and ring networks support efficient hardware broadcast. Some switched networks and internetworks do, too.
- When the physical network topology is irrelevant, we abstract from it, by including the routing layer in the infrastructure and modeling the network as a complete graph.
  - Complete graph: a graph with an edge between every pair of nodes.

Synchrony (Timing)

- Terminology (not standardized): see diagram below
- In contexts when I'm not specifically distinguishing between the application level and the network level, I may use the terms below interchangeably.
- The main hardware elements are labeled in red.



Synchrony

- Main timing-related parameters (in an abstract model), corresponding to main hardware elements:
  - Process execution speed
  - Message latency: time between send and deliver
  - Clock drift rate: ratio of the clock's speed to a perfect clock's speed
- Asynchronous model: provides bounds on none of these.
- Synchronous model: provides bounds on all of these.
- Aside: When referring to communication primitives (not timing) synchronous (asynchronous) means the sender does (does not) wait for a reply before proceeding.
- In a synchronous system, hosts can maintain approximately-synchronized clocks, i.e., there is a bound on the difference between simultaneous values of different clocks.

Is Synchronous or Asynchronous More Realistic?

- Real-time systems and embedded systems are synchronous.
- For general-purpose, multi-user systems, it's hard to get useful upper bounds on process execution speed and message latency.
- The upper bounds are much larger than the average/typical values, so algorithms based on waiting until the bound is reached are slow/inefficient.
- Causes of delays of process execution: cache misses, process scheduling, swapping, page faults, disk scheduling
- Causes of delays of message delivery, including message loss and retransmission: collisions in bus networks, contention or buffer overflow in switched networks, processing delays at gateways and re-routing after failures in internetworks

Timed Asynchronous Model

- There exist constants *e*, *l*, *d* such that usually
  - Process execution speed is bounded by *e*
  - Message latency is bounded by *l*
  - Clock drift is bounded by *d*

  Occasionally some of these bounds are violated.
- Are these “bounds” still useful?
- When the bounds hold, the system is said to be stable.
- Design the system to provide stronger guarantees when the system is stable and weaker guarantees when it is not.

Flaviu Cristian and Christof Fetzer. The Timed Asynchronous Distributed System Model. *IEEE Trans. On Parallel and Distributed Systems* 10(6): 642-657 (1999).

Timed Asynchronous Model

- Example: Distributed Database.
  - When the system is stable, each query is answered correctly within 1 second.
  - When it is not stable, some answers might reflect information that is stale by at most 1 minute.
  - Or: When it is not stable, some queries might not be answered. (Client can retry later if desired.)

Failures

- Benign Failure: omission failure or timing failure
- Omission failure: a component does not perform an expected action at the expected time. (Taxonomy later.)
- Timing failure
  - Process performance failure: processes execution speed violates bound
  - Channel performance failure: message latency violates bound
  - Clock failure: clock drift violates bound
- Arbitrary Failures (a.k.a. Byzantine Failures): a component behaves arbitrarily, i.e., omits actions and performs actions it should not have performed

Failure Model

- Failure model typically specifies
  - Types of failures that may occur
  - Maximum number or frequency of those failures
  - Maximum duration of each failure, if appropriate
- Transient failure: failure with finite duration.
  - Crashes are often considered permanent.
  - Communication failures are often assumed to be transient.
- A system must satisfy its requirements despite failures allowed by its failure model.

Omission Failures

- Crash failure: Faulty process halts forever.
- Crash and recover, with stable storage (i.e., storage that survives failures).
  - Crash and recover without stable storage is equivalent to crash, because a process that recovers without stable storage is equivalent to a new process.
- Fail-stop: Faulty process halts forever. Other processes can reliably detect this (no missed failures, no false alarms).

Vassos Hadzilacos and Sam Toueg. Fault-Tolerant Broadcasts and Related Problems. In Sape Mullender, editor, *Distributed Systems*, 2nd edition, chapter 5, pages 97-145. Addison-Wesley, 1993.

Omission Failures

- Send-omission failure: process calls send, but message does not make it onto the channel (i.e., is not transmitted onto the network).
- Link failure: the communication channel drops messages,
  - In switched networks, link failure may be due to buffer overflow in a switch, a hardware failure or software bug in a switch, a hardware failure in a cable, etc.
- Receive-omission failure: message is received at destination host but not delivered to application, e.g., due to buffer overflow in destination host's network stack.
- Why is it important to distinguish these, even though they have the same effect from the application's point of view?
  - Different techniques are needed to overcome them.