Quick Intro into Distributed Systems

12/11/2017

Distributed Systems

- Collection of independent computer systems that work together as a seamless entity to the end user
- Resources are distributed among all of the independent systems
- Each system is responsible for some part of processing
- Data is shared!

Centralized

- Mainframes/PCs
- Server + Client model (dumb terminals)

- Single point of failure
- All resources available
- Single Process *
- Single point of control

Distributed Systems

- The Internet/Modern Web Apps (also server + client)
- NFS
- Peer to Peer

- Autonomous components
- Multiple points of failure (fault tolerance!)
- Some resources unavailable at different times
- Concurrent processing
- Multiple points of control

Why Distribute?

- Distributed system vs. mainframe
  - Microprocessors offer better price/performance
  - More scalable => more computing power
  - Inherent distribution, e.g. computer-supported cooperative work
  - Reliability, Incremental growth
- Distributed system vs. independent computers/devices
  - Some applications require sharing of data, e.g. airline reservations
  - Sharing of hardware, e.g. expensive devices (color laser printer)
  - Easier human-to-human communication, e.g. electronic mail
  - Spread the workload over the machines in the most effective way
  - We want access to our data and services “everywhere”, e.g. cell phones
- Disadvantages: lack of software, coordination, management, and security are harder

Communications Protocols

- “Middleware”
  - Seamless view of distributed services
- Message Passing
  - Distributed control
- Client / Server
  - Sync easier
- Peer to Peer
  - Sync harder

Message System

- Messaging system like a layer of communication @ application level
- *asynchronous* communication
- Each process maintains a queue of messages
- Messaging system’s job is to manage the passing of messages
- Point to Point vs Publish/Subscribe aka producer/consumer

Claim: Building Distributed Systems is Hard

- Building distributed systems is HARD!
  Why?

Faults

- Fault Models:
  - Failstop: processor fails by halting. The fact that a processor has failed is detectable by other processors.
  - Crash: processor fails by halting. The fact that a processor has failed may not be detectable by other processors.
  - Byzantine failures: processor fails by exhibiting arbitrary behavior

Lack of Global Knowledge

- Consensus (when everyone needs to agree)
  - Every process starts with an initial value in {0, 1}
  - A non-faulty process decides on a value in {0, 1} by entering an appropriate decision state
  - All non-faulty processes that make a decision are required to choose the same value
  - Some process must eventually make a decision
- How do we know that processes have made a decision?

Coordinated Attack Problem (Gray, 1987)

- Two divisions of an army are camped on two hilltops overlooking a common valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously, they will win the battle; whereas if only one division attacks, it will be defeated. The divisions do not initially have plans for launching an attack on the enemy, and the commanding general of the first division wishes to coordinate a simultaneous attack. The generals can only communicate by means of a messenger. Normally, it takes the messenger one hour to get from one encampment to another. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?