CSE535: Asynchronous Systems
Fall 2017

Introduction

Scott D. Stoller

Stony Brook University

Definition

- Distributed system: a collection of computers linked by a network and equipped with software that supports sharing of resources.
- “A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

  –Leslie Lamport, 2013 ACM Turing Award winner

Motivation: Resource Sharing, Communication

- Resource Sharing
  - Resources include hardware, software, and information
  - Resource sharing increases utilization of resources hence is cost effective.
  - Resource sharing allows higher total resources (e.g., RAM, CPU) than available in a single machine.
- Communication / Cooperation
  - Examples: Skype, Facebook, SharePoint, medical information systems, factory automation systems, electronic stock exchange

Motivation: Reliability

- Distributed systems may have higher or lower availability than centralized systems, depending on design.
- Lower availability arises when resource sharing introduces a single point of failure for multiple resources.
  - Example: In Cornell CS Department, all printers are networked. One day, the disk controller for the disk with the printer information database died. No one could print, from any computer to any printer, until it was replaced.
- Higher availability is generally achieved through replication (redundancy).

Potential Disadvantages

- Lower speed, if network is slow
- Lower availability, if not designed for reliability
- Lower security (more points of vulnerability)
- Cost of network hardware and network administration
    - Savings from resource sharing usually outweighs this

Design Goals: Transparency

- Transparency: the distributed system behaves like a (fast, reliable) centralized system.
  - The distribution is transparent (invisible) to users.
- Transparency makes distributed systems easier to use.
  - Centralized systems are simpler and more familiar.
- Transparency varies by software layer.
  - Higher layers (applications) generally have higher transparency.
  - Middle layers, i.e., middleware, provide transparency.

Design Goals: Transparency

- Different aspects of behavior give rise to different aspects of transparency.
- Location Transparency: users do not need to know location of resources to access them.
  - Example: Print to any networked printer without knowing which print server it is connected to.
  - Example: Access my home directory without knowing which disk or file server it is stored on.
- Host Transparency: resources can be accessed the same way from all hosts.
  - /home/facfs1/stoller accesses my home directory on all Linux and Solaris machines in the CS Dept.
  - /usr/bin does not access the same directory on all machines. It shouldn't!

Design Goals: Transparency

- Concurrency Transparency: concurrent access to a resource has the same outcome as sequential access, i.e., the same outcome as performing the operations sequentially in some total order (the "serialization order").
- Exact meaning of concurrency transparency depends on
  - The set of operations, primarily their granularity
    - Smaller granularity provides weaker guarantees but typically allows more efficient implementation.
  - The requirements on acceptable serialization order
    - Looser requirements provide weaker guarantees (greater deviation from centralized behavior) but typically allow more efficient (e.g., lower latency) implementation.

Design Goals: Transparency: Granularity

- Example: Processes P1 and P2 concurrently access a file F in a distributed filesystem (DFS).
- Assume sector size = 1KB and block size = 2KB.
- P1: write(0 1 2) || P2: write(A B C)
  - Each character above represents 1KB of data.
- What does sequential consistency imply about number of possible outcomes (POs)? Depends on what operations are considered. Sequential consistency for:
  - Sector operations (provided by disk controller): 2^3 POs
  - Block operations (provided by typical filesystem): 2^2 POs
  - Calls to write: 2 POs
- Example: NFS v3 provides block-level atomicity, with 4KB write blocks and 32 KB read blocks.

Design Goals: Transparency: Order

- Typical requirements on serialization order, from strongest (most desirable) to weakest:
  - R1: Non-overlapping (in real time) operations must be serialized in real-time order.
  - R2: Same as above, except allow bounded violations, e.g., operations less than 1 sec apart can be serialized in either order.
  - R3: No requirements.
- Example: A file F in a DFS initially contains 0 1 2. Process P1 writes A B C, and then P2 reads from F. Can P2 read 0 1 2?
  - With R1, no. With R2, yes, if the interval between the operations is small enough. Example: NFS provides R2, because a process can read stale cached data.

Design Goals: Reliability

- Common reliability metrics:
  - Availability: the fraction of requests that are processed in a timely manner
  - MTTF: Mean Time To Failure
- Reliability is typically achieved by replication.
- Example: File availability A_{file} in terms of disk availability.
  - Without replication: $A1_{file} = A_{disk}$
  - With replication of files on 2 disks:

    $A2_{file} = 1 – \text{Prob(both disks down)} = 1 – (1 – A_{disk})^2$

    - If $A_{disk} = 0.99$, $A1_{file} = 0.99$ and $A2_{file} = 0.9999$.

Design Goals: Security, Scalability

- Security
  - Includes secrecy, integrity, availability, …
  - Distributed systems often encompass systems in multiple administrative domains.
    - Example: medical information system involving hospital, doctor’s office, insurance company, government agency (e.g., Medicare), etc.
  - Policies for access control and delegation must express complex trust relationships between different administrative domains.
- Scalability
  - Ability of the system to support more concurrent users or more tasks as more hardware resources are added.