Introduction to Distributed Systems

Material adapted from Distributed Systems: Concepts & Design, George Coulouris, et al. and Engineering Distributed Objects, Wolfgang Emmerich

Outline

- What is a Distributed System?
- Examples of Distributed Systems
- Distributed System Requirements
- Transparency in Distributed System
What is a Distributed System?

- A system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. (Coulouris)
- A distributed system is a collection of autonomous hosts that are connected through a computer network. Each host executes components and operates a distribution middleware, which enables the components to coordinate their activities in such a way that users perceive the system as a single, integrated computing facility. (Emmerich)
What is a Distributed System?

Centralized System Characteristics

- One component with non-autonomous parts
- Component shared by users all the time
- All resources accessible
- Software runs in a single process
- Single Point of control
- Single Point of failure
Distributed System Characteristics

- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple Points of control
- Multiple Points of failure

Key Terms

- **Resources** – things shared in a distributed system
  - hardware (disks, printers)
  - software (files, databases, data objects)
- **Server** – program or process that performs services in response to requests from other processes.
- **Client** – process that makes requests of a server by invoking an operation.
- **Remote Invocation** – complete send and response sequence
- Servers & Clients are **software processes**
Examples of Distributed Systems

Boeing 777 Configuration Management

SE442 - Principles of Distributed Software Systems
Problems to be solved

- Scale
  - 3,000,000 parts per aircraft
  - Configuration of every aircraft is different
  - CAA regulations demand that records are kept for every single part of aircraft
  - Aircraft evolve during maintenance
  - Boeing produce 500 aircraft per year
  - Configuration database grows by 1.5 billion parts each year
  - Projected life of each aircraft 30 years
  - 45,000 engineers need on-line access to engineering data

Problems to be solved (cont’d)

- COTS Integration
  - Existing IT infrastructure was no longer appropriate
  - Boeing could not afford to build required IT infrastructure from scratch
  - Components were purchased from several different specialized vendors
    - relational database technology
    - enterprise resource planning
    - computer aided project planning
  - Components needed to be integrated
Problems to be solved (cont’d)

Heterogeneity

20 Sequent database machines as servers for the engineering data
200 UNIX application servers
NT and UNIX workstations for engineers

Why distributed object technology

- Object wrapping of COTS
- Resolution of distribution at high level of abstraction
- Resolution of heterogeneity
- Scalability
Distributed System Requirements

Requirements

- Integration of new, legacy and components off-the-shelf
  - Legacy components might not need to be re-engineered
  - COTS cannot be modified
- Heterogeneity of
  - hardware platforms
  - operating systems
  - networks
  - programming languages
- Construction of distributed systems
Common Requirements/Challenges

- What are we trying to achieve when we construct a distributed system?
- Certain requirements are common to many distributed systems
  - Heterogeneity
  - Resource Sharing
  - Openness
  - Security
  - Concurrency
  - Scalability
  - Fault Tolerance
  - Transparency

Resource Sharing

- Ability to use any hardware, software or data anywhere in the system.
- Resource manager controls access, provides naming scheme and controls concurrency.
- Resource sharing model (e.g. client/server or object-based) describing how
  - resources are provided,
  - they are used and
  - provider and user interact with each other.
Openness

- Openness is concerned with extensions and improvements of distributed systems.
- Detailed interfaces of components need to be published.
- New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.

Concurrency

- Components in distributed systems are executed in concurrent processes.
- Components access and update shared resources (e.g. variables, databases, device drivers).
- Integrity of the system may be violated if concurrent updates are not coordinated.
  - Lost updates
  - Inconsistent analysis
Fault Tolerance

- Hardware, software and networks fail!
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by
  - recovery
  - redundancy

Scalability

- Adoption of distributed systems to
  - accommodate more users
  - respond faster (this is the hard one)
- Usually done by adding more and/or faster processors.
- Components should not need to be changed when scale of a system increases.
- Design components to be scalable!
Transparency in Distributed Systems

Transparency

- Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.
- Transparency has different dimensions that represent various properties distributed systems should have.
Access Transparency

- Enables local and remote information objects to be accessed using identical operations.
- Example: File system operations in NFS.
- Example: Navigation in the Web.
- Example: SQL Queries
Location Transparency

- Enables information objects to be accessed without knowledge of their location.
- Example: File system operations in NFS
- Example: Pages in the Web
- Example: Tables in distributed databases

Concurrency Transparency

- Enables several processes to operate concurrently using shared information objects without interference between them.
- Example: NFS
- Example: Automatic teller machine network
- Example: Database management system
Replication Transparency

- Enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs
- Example: Distributed DBMS
- Example: Mirroring Web Pages.

Failure Transparency

- Enables the concealment of faults
- Allows users and applications to complete their tasks despite the failure of other components.
- Example: Database Management System
Migration Transparency

- Allows the movement of information objects within a system without affecting the operations of users or application programs
- Example: NFS
- Example: Web Pages

Performance Transparency

- Allows the system to be reconfigured to improve performance as loads vary.
- Example: Distributed make.
Scaling Transparency

- Allows the system and applications to expand in scale without change to the system structure or the application algorithms.
- Example: World-Wide-Web
- Example: Distributed Database
Two Views of Transparency

- The system should hide its distributed nature, programs running on a multiple-computer system appear no different from a single-computer system.
- The system should *not* hide its distributed nature. The programs *are* aware of the multiple computers in the system.
- When designing distributed applications we need to favor the second view.
  (see: “A Note on Distributed Computing”, Jim Waldo, et al.)

Key Points

- What is a Distributed System
- Adoption of Distributed Systems is driven by Non-Functional Requirements
- Distribution needs to be transparent to users and application designers
- Transparency has several dimensions
- Transparency dimensions depend on each other