Distributed Systems

March 5, 2001

What is a Distributed System?

- Tanenbaum and van Renesse: A distributed system is one that looks to its users like an ordinary, centralized, system but runs on multiple independent CPUs
- Symptoms? Shroeder:
  - Multiple, independent processing units
  - Processors communicate via a hardware interconnect
  - Processing unit failures are independent
  - Manage resource sharing
  - State is shared among processors

Design Issues

- Scaling
- Communication
- Coordination
- Transparency
- Naming
- Load sharing
- Consistency
- Failures
- Security
- Heterogeneity
- Mobility

Naming

- What do we need to name?
  - Processes, Services, Hosts, Objects, Groups, ...
- Binding - Discovering and associating a name to a one of these
- e.g., DNS: hostname to IP address

Communication

- Messages can have many characteristics:
  - Length, priority, streams
- Communication medium properties affect communication performance
  - bandwidth, latency, multi-cast capability, message prioritization

Consistency

- Since we assume network links can fail at any time, replication is required to maintain consistency for longer computations
  - Replication of data
  - Replication of computation
- Costs associated with consistency:
  - Reduction in the amount of effective resources
  - Managing extended failures

Load Sharing

- Local vs. non-local
  - e.g., communication failure less likely in local clusters of processors
- Process migration can be expensive
- What about a system that knows the load of each machine, then assigns computation?
  - Doesn't scale well
  - Issues - Turning Completeness, Propagation Delay

Remote Procedure Call

- Introduced by Birrell and Nelson in 1985 (Xerox PARC)
- Abstracts the notion of where a computation runs
- Semantics can include:
  - notion that computation may fail (e.g., errors, faults, deadlocks)
  - blocking or non-blocking
  - synchronous vs. asynchronous
  - at least once, at most once, exactly once

Remote Procedure Call, cont.

- Implementation assumes a client/server framework
- RPC infrastructure must implement:
  - Server Registration, Binding, Marshalling, Message Send/Receive
  - Compiler/linker can implement stubs for RPC
- RPC used by NFS, AFS, but not X

Server Registration

- Use Discovery or Directory Service
- Each server registers a service and a version
  - Service - typically implemented as an integer; a set of well-known services exist
  - Version - a number used to select which server to use
- Who decides which server to return to the client?
  - Key issue: Load Balancing

Marshalling

- Assume heterogeneous computing environment
- Need to have a consistent set of data types
- What about pointers?
  - Client/Server operate in separate contexts…
- What about exceptions?
  - Exceptions can result from errors at the client site or network problems

End to End Design

- History: Design of the Internet
  - Q: Where should functionality be built into the Internet
  - e.g., should sessions & flows be a fundamental part of the network, or something implemented at higher layers?

End to End Design

- Insight - Many functions can be built at the ends of the network (e.g., RPC). Thus, building these functions in at lower layers will have significant tradeoffs
  - e.g., In the 70's, everyone used the DARPA-net to log into computers. If "sessions" had been built into the lower layers of the Internet, web-based traffic in the 90's would not have scaled well

13