Outline

- Chapter 15: Distributed System Structures
- Chapter 16: Distributed File Systems
- AFS paper
  - Should be familiar to you - ND uses AFS for all its file storage

Advantages of Distributed Systems

- Resource sharing
- Computation speedup
  - Load sharing
- Reliability
  - Replicated services - e.g. web services (yahoo.com)
- Network Operating Systems
  - Explicit network service access
- Distributed Systems - transparent
  - Data migration
  - Computation migration
  - Process migration

Network constraints

- Specific system design depends on the network constraints
  - LAN vs WAN (latency, reliability, available bandwidth, etc.)
  - Naming and Name resolution (Internet address)
  - Routing, data transmission, connection and other networking strategies
- Distributed File System as a Distributed "Operating" system service

Distributed File System

- Naming and transparency:
  - Location transparency: Name does not hint on the file's physical storage location
    - (/net/wizard/tmp is not location transparent)
  - Location independence: Name does not have to be changed when the physical storage location changes
    - AFS provides location independence
    - (/afs/nd.edu/user37/surendar)

Remote file access

- Caching scheme
  - Cache consistency problem
    - Blocks (NFS) to files (AFS)
  - Cache location
    - Main memory vs disk vs remote memory
  - Cache update policy
    - Write-through policy, delayed-write policy (consistency vs performance)
  - Consistency (client initiated or server initiated)
    - Depends on who maintains state

Stateful vs stateless service

- Either server tracks each file access or it provides block service (stateless)
  - AFS vs NFS
  - Server crash looks like a slow server to stateless client.
  - Server crash means that state has to be rebuilt in stateful server
  - Server needs to perform orphan detection and elimination to detech “dead” clients in stateful service
  - Stateless servers: larger requests packets, as each request carrys the complete state
  - Replication - to improve availability

AFS

- Developed in mid 80's at CMU to support about 5000 workstations on campus
- Stateful server with call backs for invalidation
- Shared global name space
- Clusters of servers implement this name space at the granularity of volumes
- All client requests are encrypted
- AFS uses ACLs for directories and UNIX protection for files

File operations and consistency semantics

- Each client provides a local disk cache
- Clients cache entire files (for the most part - AFS3 allows blocks)
  - Large files pose problems with local cache and initial latency
- Clients register call back with server & Server notifies clients on a conflict read-write conflict to invalidate cache
- On close, data is written back to the server
- Directory and symbolic links are also cached in later versions
- AFS coexists with UNIX file systems and uses UNIX calls for cached copies

Design principles for AFS and Coda

- Workstations have cycles to burn - use them
- Cache whenever possible
- Exploit file usage properties
  - Temporary files are not stored in AFS
  - Systems files use read-only replication
  - Minimize system wide knowledge and change
  - Trust the fewest possible entities
  - Batch if possible