CS 537
Lecture 18
Distributed File Systems

Michael Swift

Distributed File Systems

- One of the most common uses of distribution is to provide distributed file access through a distributed file system
- Basic idea: support sharing of files and sharing of devices (disks) network wide.
- Generally provides a "timesharing system" type view of a centralized file system, but with distr. implementation.

Basic Issues

- File naming
  - how are files named?
  - are those names location transparent (is the file location visible to the user)?
  - are those names location independent?
    - do the names change if the file moves?
    - do the names change if the user moves?

Basic Issues

- Caching
  - caching exists for performance reasons
  - where are file blocks cached?
    - On the file server?
    - On the client machine?
- Coherency
  - what happens when a cached block/file is modified
  - how does a node know when its cached blocks are out of date?

Issues

- Replication
  - replication can exist for performance of availability
  - can there be multiple copies of a file in the network?
  - if multiple copies, how are updates handled?
  - what if there's a network partition and clients work on separate copies?
  - at what level is replication visible?

Issues

- Performance
  - what is the cost of remote operation?
  - what is the cost of file sharing?
  - how does the system scale as the number of clients grows?
  - what are the performance limitations: network, CPU, disks, protocols, data copying?

Example Systems: NFS

- The Sun Network File System (NFS) has become a common standard for distributed UNIX file access.
- NFS runs over LANS (even over WANs -- slowly).
- Basic idea: allow a remote directory to be "mounted" (spliced) onto a local directory, giving access to that remote directory and all its descendents as if they were part of the local hierarchy.
- Ex: I mount /usr/swift on Node1 onto /students/foo on Node2. Users on Node2 can then access my files as /students/foo. If I had a file /usr/swift/myfile, users on Node2 see it as /students/foo/myfile.

NFS

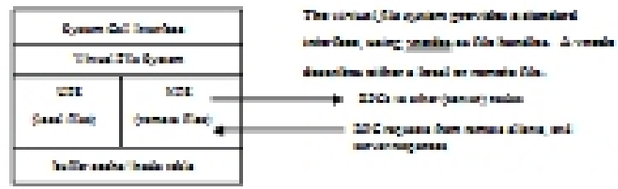
- NFS defines a set of RPC operations for remote file access:
  - searching a directory
  - reading directory entries
  - manipulating links and directories
  - reading/writing files
- Every node may be both a client and server.

Remote Procedure Call

- Basic problem when dealing with machine across a network: how do you write the code to communicate?
- Option 1: messages
  - Programmer copies message into an array of bytes, "sends" to other computer, "receives" an array of bytes in response at some point
- Option 2: RPC
  - Make a procedure call that executes on the other side
  - Tool generates code to copy arguments into a message, send data, unpack data, call server code, copy result into a message, send back, receive reply, and return to caller

NFS Implementation

- NFS defines new layers in the Unix file system



- Buffer cache caches remote file blocks and attributes

NFS Caching and Consistency

- NFS clients cache blocks of files in memory
  - Part of standard buffer cache
- On an open, the client asks the server whether its cached blocks are up to date.
  - If not, must refetch file
- Once a file is open, multiple clients can write it
  - What is the result of multiple writes?
- Modified data is flushed back to the server every 30 seconds.
  - What does a reader see?

The Andrew File System

- Developed at CMU to support all of its student computing.
- Consists of workstation clients and dedicated file server machines.
- Workstations have local disks, used to cache files being used locally (originally whole files, now 64K file chunks).
- Andrew has a single name space -- your files have the same names everywhere in the world.
- Andrew is good for distant operation because of its local disk caching: after a slow startup, most accesses are to local disk.