# FARSITE and AFS

Alan Sussman

CMSC 818S

April 17, 2007

# Notes

- Project interim report due tomorrow
- Final dates – how about Monday-Wednesday (5/14-16)?

# FARSITE

# Overview

- Loosely coupled, insecure, unreliable machines
- Logically centralized, secure, reliable file storage service
  - encryption for file data privacy
  - one-way hashing for file data integrity
  - replication for file data durability
  - directory metadata maintained by Byzantine-replicated state machines and crytopgraphy
    - but use signed, dated certificates to avoid its full cost by caching authorizations granted with the expensive protocols
- Targets desktop machines in academic and corporate settings
  - workloads have high access locality, low persistent update rate, not much concurrent read/write sharing
  - machines have high fail-stop rate, low but non-trivial rate of malicious or opportunistic subversion
- Central administration only needed for initial configuration and to authenticate new users and machines via signing certificates

# System Design

- Namespace roots
  - multiple roots, each a single virtual file server
  - consists of a unique root name and a set of machines to manage the root
    - a Byzantine fault tolerant group
- Trust/Certification
  - *namespace certificate* to associate root of namespace with set of machines managing root metadata
  - *user certificate* associates user with his/her public key, for access control
  - *machine certificate* associates machine with its public key, to prove machine is a valid resource
  - certificates can be revoked, since they expire

# System architecture

- Machine may have 3 roles – client, member of directory group, file host
- Directory group is set of machines that collectively manages a root file system
  - each machine stores a replica of the metadata
  - uses a Byzantine fault tolerant protocol that guarantees data consistency as long as at least 2/3 of the machines behave properly
- Performance enhancements include:
  - client caching of file contents, with expiration leases
  - delay pushing updates to the directory group, since may not be necessary (since file writes often deleted or overwritten soon after)
  - file data encrypted so only authorized users can decrypt
  - use secure hash so client can validate file contents, so a file host cannot corrupt file data
  - directory group can *delegate* part of its namespace to another group, to shed load

# Reliability and Availability

- Main technique is replication
  - directory data replicated across members of a directory group, with Byzantine fault tolerance
  - file data just replicated on multiple file hosts
  - if a machine becomes unavailable, its functions migrate to one or more other machines
    - directory migration performed aggressively, to maintain Byzantine properties
    - file migration performed in background, targeting equitable distribution of file availability (equal use of low and high-available machines)

# Security

- Directory metadata includes *access control* list, assumed to be correct (Byzantine guarantee)
  - client authenticates using its private key
- File content and file/directory names encrypted for *privacy*
  - using multiple levels of encryption
- File data *integrity* maintained by cheap to compute, update and validate secure hash of the file contents

# Durability

- Updates to file metadata (create, modify, rename, delete file or directory) done on client's local disk and logged
  - log pushed back to directory group periodically and when a lease is recalled, which then applies log entries to system metadata after verification of each entry
- Also need to deal with client machine crash, via complex method that avoids client signing every update, and atomic mods of both metadata and file content

# Consistency

- Directory group has ultimate responsibility
- But use leases to clients to improve performance
  - *content leases* (read/write and read-only) say which client machines have control of a file's content – granted by directory group, and can be recalled
    - can cover a file, or a directory of files, and expires
  - *name leases* say which client machine has control over a name in the directory namespace, can be recalled
    - if the name doesn't exist, the client can create the file or directory
    - if the directory name does exist, then the client can create files or sub-directories
  - *mode leases* to support Windows file-sharing semantics
    - *read, write, delete, exclude-read, exclude-write, exclude-delete*
    - checked at file open to grant the type of access the client wants – read, write or delete
  - *access leases* to not delete a file until all clients done with it – Windows delete file semantics issues

# Scalability

- Hint-based pathname translation
  - to avoid having to search through all directory groups to find a given name, from the root (the bottleneck)
  - basically do prefix matching in client cache to find the best directory group to start
- Delayed directory-change notification
  - use Windows callbacks to allow client to find out when a change occurs to a directory – best-effort

# Efficiency

- Space
  - reclaim space from duplicated files
    - claim is that Windows helps do this
- Time
  - client caches encrypted file contents
  - lease mechanisms, hint-based pathname translation
  - delay replicating file, since creation or update often followed by deletions or other updates

# Manageability

- Local machines
  - removing a machine, or replacing a disk, is same as a failure – fix via replication
  - use major and minor version numbers to ensure interoperability between versions of software, in all messages establishing connections
  - backup for reliability not needed
- Administration
  - through distributed, Byzantine fault tolerance
  - for lazy and periodic tasks (e.g., replica relocation), use timed Byzantine operations triggered by keeping track of local times and getting agreement on global time,
    - directory group invokes the operation on one remote machine as a hint, which invokes the operation on the group – kind of strange, but it works

# Evaluation

- Small scale tests on 5 P3 machines show that performance is worse than local NTFS, but better than CIFS (remote file access for Windows), and it's not in the kernel
  - slower than CIFS on writes, faster on reads and queries

# Andrew File System

# Overview

- Location-transparent distributed file system
  - project started at CMU in 1983
  - target is 5000 to 10000 nodes – they got there
  - client-server organization
    - set of servers are trusted – *Vice*
    - clients are user-level processes , called *Venus*, that cache **whole files** from Vice, store back if needed
      - contact Vice only when file opened or closed, all reads and writes done on cached copy of file
  - goal is to maximize number of clients a server can support
  - paper concentrates on scalability issues

# The prototype

- Venus client connects to server on well known port, server creates a process to deal with future client requests
- Communication between servers via shared file system
- Vice server contains directory hierarchy mirroring structure of files it stores
  - and *Stub* directories pointing to portions of Vice name space on other servers
  - and clients cache pathname prefix info to direct file requests to the right servers
- Full pathnames used to name files and directories
  - read-only replication of top levels of name tree, with single server as owner for updates
- Cached copies of files verified by timestamp on server responsible for file
- Performance problems from cache validity checks, too many server processes, pathname traversals, and unbalanced load on servers

# Performance enhancements

- Cache management
  - Venus caches directory contents and symbolic links, and files
  - One cache for status, one for data
    - keep status cache in memory for quick metadata search (file *stat* call)
  - directory mods are made on the server, but also updated in the client cache
  - biggest change is Venus cache consistency method
    - assume valid unless notified otherwise by server – a callback
    - potential for inconsistent state between server and clients, but better performance

# Performance (cont.)

- Name resolution
  - Use two-level names, as in standard Unix filesystem (pathnames and inodes)
  - a fixed-length *Fid*, and directory entries map a component of a pathname to an fid
    - 32-bit *Volume* number (a collection of files on 1 server)
    - 32-bit *Vnode* number – index into an array with file storage info for a Volume
    - 32-bit *Uniquifier* – to prevent name collisions

# Performance (cont.)

- Communication and server process structure
  - To allow server processes to share information in memory, they basically built a thread package, calling it *Lightweight Processes (LWP)*
  - Bind an LWP for each client operation
  - Clients and servers communicate via RPC, outside kernel

# Performance (cont.)

- Low-level storage representation
  - access Vice files on server via inodes rather than pathnames – requires new system calls
  - Venus does this too, for the client cache in the local file system

# File consistency

- AFS guarantees:
  - writes to open file on client machine are visible to other processes on that machine, but not to other clients in network
  - once file closed, changes are made visible to any new opens by a client
    - already open instances of file don't see the changes
  - other file operations (metadata) visible everywhere on network once operation completes
  - multiple client processes can perform same operation on a file concurrently – no implicit locking
    - applications must synchronize correctly

# Performance measurements

- The enhancements do help scalability, and overall performance
  - lower server loads
  - faster client response times
- And performs much better than Sun NFS under heavy loads
  - with a lot less network traffic

# Changes for operability

- Volumes
  - collection of files that form a partial subtree of the Vice name space
    - glued together to form the complete name space
    - resides in a single disk partition
- Moving volumes
  - to redistribute among servers for balancing available disk space and server utilization
  - just requires update to volume location database, and move data with copy-on-write creation of a *Clone* that is serialized and shipped to new site – updates during the process work because of copy-on-write (just keep making Clones until nothing changes)
- Quotas
  - implemented with 1 volume per user, with a quota
  - problem is that it has to fit in 1 disk partition

# Operability (cont.)

- Read-Only replication
  - system programs and files in upper levels of Vice name space frequently read, rarely updated
  - so replicate at multiple servers – and no callbacks – one read/write copy, and a set of read-only replication servers
  - at granularity of a volume
- Backup
  - unit is a volume
  - make a read-only clone, then asynchronous transfer of clone to machine where it will be dumped to tape
  - volume can be restored to any server
  - to handle accidental deletions, a read-only clone of user's files is made available in subtree of user's home directory
    - still uses copy-on-write for performance

# Summary

- AFS is a success, 20 years later
- Commercialized, and used at many sites