CS 455: Introduction To Distributed Systems [Nfs/Afs]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- If you move a file, does it get a new inode?
- Does it take longer to locate inodes of files deep in a directory?
- How do inodes work on a machine with multiple hard drives?
- RAID systems and the inodes?

Topics covered in this lecture

- Distributed file systems
  - NFS
  - AFS
- Presentation Schedule
- CS455 Final Exam

Semantics of file sharing

When two users share the same file

- Define **semantics** of reading & writing precisely
- Avoid problems

Semantics of file sharing on single processor systems

- When READ follows WRITE
  - READ returns the value that was just written
- When READ follows two successive WRITES
  - READ returns value that was written last
- Absolute time-ordering on all operations
  - Returns most recent value
  - **UNIX semantics**

Achieving UNIX semantics in a distributed system

- Possible if there is only **one server** AND
  - Clients **do not cache** files
  - READs and WRITEs processed strictly sequentially
- Performance is quite **poor**, so …
  - Allow clients to maintain local copies of files
    - In their private caches

But what if a client locally modifies the cache, and another reads same file from server?

- **Session semantics**
  - Changes to an open file are visible only to process that modified it
  - Only when file is ***closed*** is the change made visible to other processes

Make all files immutable

- No way to open a file for writing
- Only operations on a file are
  - CREATE and READ
- What happens if two processes replace the same file at the same time?
  - **Solution:** Allow one of the new files to replace the old one

The transactional approach

- Access to a file (or a group of files) done in a **transactional** fashion
- System guarantees that calls within transaction
  - Will be carried out **in order**
  - **Without interference** from other concurrent transactions

Contrasting approaches to dealing with shared files in distributed systems

Method	Comment
UNIX semantics	EVERY operation on a file is **instantly visible** to all processes
Session semantics	No changes are visible to other processes **until the file is closed**
Immutable files	**No updates** are possible Simplifies sharing and replication
Transactions	All changes have the **all-or-nothing** property

File Replication

File Replication: Rationale

- Redundancy improves availability
  - Reliability too
- Improves performance
  - Choose a **closer replica** for communications
    - Better network pipe
    - Faster interactions

File replication: Basic requirements [1/2]

- File replicas reside on **failure-independent** machines
- Availability of one replica should not affect another
- Existence of replica **invisible** to higher-levels
- Hide details of the replication scheme
  - Mapping handled by the naming schemes

File replication: Basic requirements [2/2]

- Updates to one replica must be **reflected on all** the other replicas
- **Consistency semantics** must be preserved

Replica placement metrics besides failure independence

- Client load
- Expected minimization of access latencies
- Expected improvements in communication bandwidths

Network File System

NFS is a spec and implementation for accessing remote files across LANs/WANs

- Views workstations as **independent** machines
  - With independent file systems
- Allow **sharing** among file systems transparently
  - Based on the *client-server* model
  - Every machine is both a client and a server

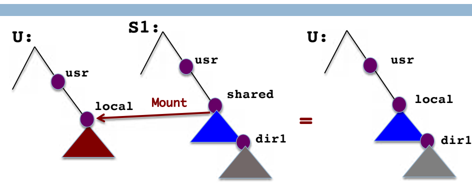
Design goals of NFS

- Operate in an **heterogeneous** environment
- Independence
  - Through RPC and External Data Representation (**XDR**)
    - XDR is an IETF standard (1995)

Accessing a remote directory from a machine A

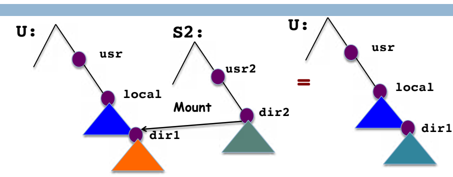
- Client on machine **A** must perform the **mount** operation
- Mounted directory
  - **Looks like** a subtree of the local file system
  - **Replaces** subtree descending from local directory
  - Local directory name **becomes the name** of the newly mounted directory

The mount operation



File in **dir1** directory accessed with prefix **/usr/local/dir1**

Cascading mounts



File in **dir2** directory accessed with prefix **/usr/local/dir1**

The mount protocol

- Establishes initial connection between a server and a client
- Server maintains an **export list**
  - List of its local files that can be mounted
- Server also maintains list of client machines and mounted directories
  - Administrative purposes

NFS Protocol provides a set of RPCs for remote file operations

- Search for file in directory
- Read a set of directory entries
- Access file attributes
- Read and write files

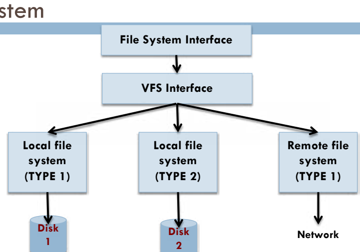
NFS protocol is stateless

- Servers **do not** maintain info about clients from one access to another
- No `open()` or `close()` operations are provided
- Each request must **specify all arguments**
  - File *identifier*
  - Absolute *offset* inside the file

More about the NFS protocol

- All NFS requests have **sequence numbers**
  - For detection of duplicates and missed messages
- **No lock management**
  - This is inherently stateful
  - Use separate service to coordinate access to files

NFS is integrated into the file system via the virtual file system



OS maps call to a VFS operation onto the appropriate **vnode**

Caching in NFS: There are two caches that are maintained in NFS

- **Attributes** cache
  - inode information
  - Kernel checks for validity during access
  - Discarded after 60 seconds
- **File blocks** cache
  - Uses both *read-ahead* and *delayed-write*

Consistency semantics of NFS

- New files may not be visible for up to 30 seconds
- New opens of a file observe only the changes that have been **flushed** to the server
  - Writes to a file at site **A** may not be visible to another site **B** that is reading the same file
- NFS provides **neither UNIX semantics nor session semantics**

The Andrew File System

Andrew File System (AFS)

- Designed at CMU
  - Derives its name from its founders
- Transarc took over development of AFS
- Transarc purchased by IBM
  - Several commercial implementations of AFS
  - In 2000, IBM announced that AFS would be open-source

Some features of AFS

- Uniform namespace
- Location-independent file sharing
- Client-side caching
  - With cache consistency
- Secure authentication
  - Via **Kerberos**
- Replication

Clients and Servers in AFS are structured in clusters interconnected by a WAN

- Servers
  - Collectively called **Vice**
- Clients
  - Run the **Virtue** protocol to communicate with Vice

The organization of clients and servers into clusters addresses the issue of scale

- Each cluster is a collection of
  - Workstations on a LAN
  - Cluster Server
    - Representative of Vice
- Each cluster connected to a WAN
- Workstations should use their *own* cluster server
  - Most of the time

Name spaces in AFS: Clients have a partitioned space of file names

- **Local** namespace
  - Root of the file system on client workstation
  - Small and distinct for each workstation
  - Contains system programs
- **Shared** namespace

Name space in AFS: The Shared name space

- Servers **collectively responsible** for the shared name space
  - Storage
  - Management
- Present shared name space as
  - Homogenous
  - Identical
  - Location transparent

Design issues and considerations in AFS

- **Scale** is the most important issue
- **Offload work** from the server to the clients
  - Cache files in chunks (64 KB)
  - Reads/writes directed to a cached copy
- Clients access any file in the shared name space
  - **From anywhere**

Design issues and considerations in AFS

- Security
  - No client programs executed on VICE machines
  - Mutual authentication between client/server
  - Communication via encrypted messages
- Protection
  - Access lists for protecting directories
    - Who **is** allowed
    - Who **is not** allowed
    - Types: read, write, lookup, insert, administer, lock and delete
  - UNIX bits for file protection

AFS shared space: Volume

- Key administrative unit
  - Identification and location of **individual files**
- Small component unit
  - Files of a single client
- Few volumes reside on a single disk partition

The AFS shared name space

- **Shared name space** made up of volumes
- Volumes glued together
  - Mechanism similar to UNIX `mount` operation
  - Granularity is different
    - In UNIX **only entire disk partitions** can be mounted

A VICE file or directory is identified by a fid that is 96 bits long

- Has **three** 32-bit elements
  - **Volume** number
  - **Vnode** number
    - Index into array of inodes for a volume
  - Uniquifier
    - Allows reuse of vnode numbers
- `Fids` are location-independent
  - File movements do not invalidate cached directory entries

Location information in AFS is kept on a volume basis

- Stored in a **volume-location database**
  - Replicated on each server
- Database size is **manageable**
  - Files are aggregated into volumes
- Client identification of volume location
  - **Query**ing this database

Volumes often migrate among disk partitions and servers

- **Balance** available space and utilizations
- When a volume is shipped
  - *Forwarding info* stored at the old location
- During the shipping of a volume
  - Old server can continue to handle updates
    - Forwarded later on to new location
  - Volume is disabled at some point to sync things up

File Operations and Consistency

AFS Caching

- In AFS **entire files are cached** from the servers
- Client workstation has **limited interactions** (remote) with the Vice servers
  - Only during opening and closing of files

How caching is done at each workstation

- OS **intercepts** file-system calls
  - Forwards to a client-level process: **Venus**
- Venus
  - Caches files received from Vice servers
  - Stores modified copies of files back on server
    - When file is closed
    - **Session semantics**

Caching is also exploited during future opens of the file

- Venus assumes cached entries are valid
  - Unless notified that they are invalid
    - **Callback**
  - No need to contact Vice otherwise

Caching and the server

- When a client caches file/directory
  - Server updates state information to record this caching
  - **Client has a callback** on the file
- Server notifies client before allowing another client to modify the file
  - **Server removes callback** for the old client

Subsequent reading and writing of cached files

- Done **without intervention** from VENUS
- VENUS **regains** control
  - When file is closed
  - Updates are pushed to server from where it came
- Session semantics for read/write
  - Other operations are *immediately visible*
    - E.g. Protection changes

Other implementation Issues: Caches and Storage

- **Mapping cache** associates volume to location
  - Avoid redundant queries for known volumes
- UNIX file system used for low-level storage
- VENUS manages two bounded caches
  - Status
  - Data

Other implementation issues: Server side

- Services file requests from all clients
- Uses lightweight-process package
  - **Non-preemptive** scheduling
  - Service many requests concurrently
  - RPC package integrated into lightweight package

HOW TO POSITION YOUR ASSIGNMENTS

How to position your assignments

- In your CV
- During interviews

PRESENTATIONS WEEK

Presentations Week [1/3]

- We will have presentations on
  - Tuesday: 12:30-1:45 pm
  - Thursday: 12:30-1:45 pm
  - Friday: 4:00-5:00 pm
- Per-session deduction of 1 point for arriving late (or leaving early)
  - 10 minute window

Presentations Week [2/3]

- Submit your slides as a PowerPoint file
  - The slides must follow the exact format specified in the guidelines
  - Please no plain BW slides … use some color!
  - Slides will be launched from the computer in the room
    - You will be provided with a remote
- Please rehearse a few times (as a team)
  - If you try to wing it while you are up here, it will take much longer than 11 minutes (10 minutes for presenting and 1 for answering questions)
    - Also, if you haven't rehearsed everyone else can tell
  - All team members must take turns talking
    - Distance students are exempt from this constraint

Presentations Week [3/3]

- No laptops are allowed in class
- 2-3 students will be randomly asked to ask questions for each talk
  - Ask legitimate questions
    - A bad/terrible question: Could you talk a little more about that?
  - Each student will be asking at least 2 questions across the 3 sessions
    - 3 chances to ask 2 good questions
    - 1 point deduction if you pose 2 bad questions in 3 attempts

CS455 Final exam

CS455 Finals

- Will account for 10% of your course grade
  - Exam will be for 100 points
  - Comprehensive exam
- **Wednesday, May 9th, from 9:40-11:40 am**
- The exam will be a **Canvas Exam in CSB-110**.
  - If you want a paper exam, please let us know by 4/27 (email cs455@cs.colostate.edu) and we will administer a paper-based exam in CSB-130 at the same time.

CS455 Finals breakdown

Elements	Points distribution
Threads: Safety, concurrency, etc	25
Basic MapReduce	10
Hadoop and HDFS	10
Spark	15
Distributed Mutual Exclusion	10
Election Algorithms	10
Architecture & Topologies	10
File Systems: UFS/inodes, NFS, and AFS	10

FINALLY … THANK YOU FOR A WONDERFUL SEMESTER

The contents of this slide set are based on the following references

- Avi Silberschatz, Peter Galvin, Greg Gagne. *Operating Systems Concepts, 8th edition. Publisher - John Wiley & Sons, Inc. ISBN-13: 978-0-470-12872-5.* [Chapter 10]
- *Andrew S Tanenbaum. Modern Operating Systems. (3rd Edition, 2007). Publisher - Prentice Hall. ISBN 0136006639/ 978-0136006633.* [Chapter 4]