20: Distributed File Systems

Last Modified:
7/3/2004 1:51:15 PM

Background

- Distributed file system (DFS) – a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources.
- A DFS manages set of dispersed storage devices
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces.
- There is usually a correspondence between constituent storage spaces and sets of files.

DFS Structure

- **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients.
- **Server** – service software running on a single machine.
- **Client** – process that can invoke a service using a set of operations that forms its *client interface.*
  - A client interface for a file service is formed by a set of primitive *file operations* (create, delete, read, write).

Naming and Transparency

- *Naming* – mapping between logical and physical objects.
- Multilevel mapping – abstraction of a file that hides the details of how and where on the disk the file is actually stored.
- A *transparent* DFS hides the location where in the network the file is stored.
- For a file being replicated in several sites, the mapping returns a set of the locations of this file's replicas; both the existence of multiple copies and their location are hidden.

Naming Structures

- **Location transparency** – file name does not reveal the file's physical storage location.
  - File name still denotes a specific, although hidden, set of physical disk blocks.
  - Convenient way to share data.
  - Can expose correspondence between component units and machines.
- **Location independence** – file name does not need to be changed when the file's physical storage location changes.
  - Better file abstraction.
  - Promotes sharing the storage space itself.
  - Separates the naming hierarchy from the storage-devices hierarchy.

Naming Schemes — Three Main Approaches

- 1) Files named by combination of their host name and local name; guarantees a unique systemwide name.
- 2) Attach remote directories to local directories, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently.
- 3)Total integration of the component file systems.
  - A single global name structure spans all the files in the system.
  - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.

Client Caching

- Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally.
  - If needed data not already cached, a copy of data is brought from the server to the user.
  - Accesses are performed on the cached copy.
  - Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches.
  - *Cache-consistency* problem – keeping the cached copies consistent with the master file
  - Where is cache? Client disk, Client memory, Both?

Benefits of Caching

- Fast Access
  - many remote accesses handled efficiently by the local cache; most remote accesses will be served as fast as local ones.
- Decrease Load on Servers
  - Servers are contracted only occasionally in caching (rather than for each access).
  - Enhances potential for scalability.
- Efficient Use of Network
  - Reduces network traffic.
  - Total network overhead in transmitting big chunks of data (caching) is lower than a series of responses to specific requests (remote-service).

Problems of Caching

- Cache Consistency
  - With frequent writes, substantial overhead incurred to overcome cache-consistency problem
  - More later..
- New Interface
  - In caching, the lower intermachine interface is different form the upper user interface.
  - In remote-service (no caching), the intermachine interface mirrors the local user-file-system interface.

Cache Update Policy

- **Write-through** – write data through to disk as soon as they are placed on any cache. Reliable, but poor performance.

- **Delayed-write** – modifications written to the cache and then written through to the server later. Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.
  - Poor reliability; unwritten data will be lost whenever a user machine crashes.
  - Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan.
  - Variation – *write-on-close*, writes data back to the server when the file is closed. Best for files that are open for long periods and frequently modified.

Consistency

- Is locally cached copy of the data consistent with the master copy?

- Client-initiated approach
  - Client initiates a validity check.
  - Server checks whether the local data are consistent with the master copy.

- Server-initiated approach
  - Server records, for each client, the (parts of) files it caches.
  - When server detects a potential inconsistency, it must react.

Stateful File Service

- Mechanism.
  - Client opens a file.
  - Server fetches information about the file from its disk, stores it in its memory, and gives the client a connection identifier unique to the client and the open file.
  - Identifier is used for subsequent accesses until the session ends.
  - Server must reclaim the main-memory space used by clients who are no longer active.

- Increased performance.
  - Fewer disk accesses.
  - Stateful server knows if a file was opened for sequential access and can thus read ahead the next blocks.

Stateless File Server

- ❒ Avoids state information by making each request self-contained.
- ❒ Each request identifies the file and position in the file.
- ❒ No need to establish and terminate a connection by open and close operations.

Distinctions Between Stateful & Stateless Service

- ❒ Failure Recovery.
  - ❍ A stateful server loses all its volatile state in a crash.
    - • Restore state by recovery protocol based on a dialog with clients, or abort operations that were underway when the crash occurred.
    - • Server needs to be aware of client failures in order to reclaim space allocated to record the state of crashed client processes (orphan detection and elimination).
  - ❍ With stateless server, the effects of server failure sand recovery are almost unnoticeable. A newly reincarnated server can respond to a self-contained request without any difficulty.

Distinctions (Cont.)

- ❒ Penalties for using the robust stateless service:
  - ❍ longer request messages
  - ❍ slower request processing
  - ❍ additional constraints imposed on DFS design
- ❒ Some environments require stateful service.
  - ❍ A server employing server-initiated cache validation cannot provide stateless service, since it maintains a record of which files are cached by which clients.
  - ❍ UNIX use of file descriptors and implicit offsets is inherently stateful; must maintain tables to map the file descriptors to inodes, and store the current offset within a file. (Can do on the client side though)

File Replication

- ❒ Replicas of the same file reside on failure-independent machines.
- ❒ Improves availability and can shorten service time.
- ❒ Naming scheme maps a replicated file name to a particular replica.
  - ❍ Existence of replicas should be invisible to higher levels.
  - ❍ Replicas must be distinguished from one another by different lower-level names.
- ❒ Updates – replicas of a file denote the same logical entity, and thus an update to any replica must be reflected on all other replicas.
- ❒ Demand replication – reading a nonlocal replica causes it to be cached locally, thereby generating a new nonprimary replica.

AFS – Andrew File System

- ❒ A distributed computing environment under development since 1983 at Carnegie-Mellon University.
- ❒ Andrew is highly scalable; the system is targeted to span over 5000 workstations.
- ❒ Andrew distinguishes between client machines (workstations) and dedicated *server machines*. Servers and clients run the 4.2BSD UNIX OS and are interconnected by an internet of LANs.

AFS (Cont.)

- ❒ Clients are presented with a partitioned space of file names: a *local name space* and a *shared name space*.
- ❒ Servers collectively are responsible for the storage and management of the shared name space.
  - ❍ Dedicated servers, called Vice, present the shared name space to the clients as an homogeneous, identical, and location transparent file hierarchy.
- ❒ The local name space is the root file system of a workstation, from which the shared name space descends.
  - ❍ Workstations run the Virtue protocol to communicate with Vice, and are required to have local disks where they store their local name space.

AFS Shared Name Space

- Andrew's volumes are small component units associated with the files of a single client.
- A fid identifies a Vice file or directory. A fid is 96 bits long and has three equal-length components:
  - volume number
  - vnode number – index into an array containing the inodes of files in a single volume.
  - uniquifier – allows reuse of vnode numbers, thereby keeping certain data structures, compact.
- Fids are location transparent; therefore, file movements from server to server do not invalidate cached directory contents.
- Location information is kept on a volume basis, and the information is replicated on each server.

AFS File Operations

- A key mechanism selected for remote file operations is *whole file caching*.
  - Opening a file causes it to be cached, in its entirety, on the local disk.
  - Reading and writing bytes of a file are done by the kernel without Venus intervention on the cached copy.
  - Changes written back to server on close
- Exceptions to the caching policy are modifications to directories that are made directly on the server responsibility for that directory
- Servers are stateful
  - Servers hold callbacks on files clients are caching so can notify client if another one writes back the file with changes

NFS – Network File System

- Completely different design point from AFS
- Implemented by Sun in early 80s
- Write-through caching
  - Servers involved on every write
  - If two clients actively writing, writes will be interleaved semi-randomly as they reach the server
- Stateless protocol
  - Each operation requires complete information about the request
  - Operations must be idempotent (ok to repeat as many times as you want)
    - Ok for read/write; remove? May remove wrong version of a file if removed then recreated then remove is retried
  - Failed server just looks really slow to clients

NFS (con't)

- Reads allowed to proceed out of client cache
- Client periodically polls server to see if file has changed (been written by someone else while they were reading it)
- Hard to reason about who will see a change and when

Outtakes

Cache Location – Disk vs. Main Memory

- Advantages of disk caches
  - More reliable.
  - Cached data kept on disk are still there during recovery and don't need to be fetched again.
- Advantages of main-memory caches:
  - Permit workstations to be diskless.
  - Data can be accessed more quickly.
  - Performance speedup in bigger memories.
  - Server caches (used to speed up disk I/O) are in main memory regardless of where user caches are located; using main-memory caches on the user machine permits a single caching mechanism for servers and users.

ANDREW Implementation

- Client processes are interfaced to a UNIX kernel with the usual set of system calls.
- Venus carries out path-name translation component by component.
- The UNIX file system is used as a low-level storage system for both servers and clients. The client cache is a local directory on the workstation's disk.
- Both Venus and server processes access UNIX files directly by their inodes to avoid the expensive path name-to-inode translation routine.

ANDREW Implementation (Cont.)

- Venus manages two separate caches:
  - one for status
  - one for data
- LRU algorithm used to keep each of them bounded in size.
- The status cache is kept in virtual memory to allow rapid servicing of *stat* (file status returning) system calls.
- The data cache is resident on the local disk, but the UNIX I/O buffering mechanism does some caching of the disk blocks in memory that are transparent to Venus.