CSC 4103 - Operating Systems
Spring 2007

LECTURE - XXIII

DISTRIBUTED SYSTEMS - II

Tevfik Koşar

Louisiana State University

April 26th, 2007

Cache Update Policy

- **Write-through** – write data through to disk as soon as they are placed on any cache
  - simple
  - reliable (little information is lost if client crashes)
  - but poor performance in writes (each write has network overhead)
- **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 - flush a block back when it is about to be ejected from client's cache
  - 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 (eg. AFS)
    - 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
  - Contacts server to check 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
    - Potential inconsistency: two clients open the same file in conflicting modes
    - When servers detects this, it disables caching for this file ==>switch to remote service mode of operation

Comparing Caching and Remote Service

- In caching, many remote accesses handled efficiently by the local cache; most remote accesses will be served as fast as local ones
- Servers are contracted only occasionally in caching (rather than for each access)
  - Reduces server load and network traffic
  - Enhances potential for scalability
- Remote server method handles every remote access across the network; penalty in network traffic, server load, and performance
- Total network overhead in transmitting big chunks of data (caching) is lower than a series of responses to specific requests (remote-service)

Caching and Remote Service (Cont.)

- Caching is superior in access patterns with infrequent writes
  - With frequent writes, substantial overhead incurred to overcome cache-consistency problem
- Benefit from caching when execution carried out on machines with either local disks or large main memories
- Remote access on diskless, small-memory-capacity machines should be done through remote-service method

Stateful vs Stateless Service

Two approaches for storing server-side info when a client accesses remote files:

- Stateful: Server tracks each file being accessed by each client
- Stateless: Server provides blocks as they are requested by each client, without knowing how those blocks are used

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
- Eg. AFS

Stateless File Server

- Avoids state information by making each request self-contained
- Each request (eg. read and write) identifies the file and position in the file in full
- No need to establish and terminate a connection by open and close operations
- No need to keep a table of open files in memory
- Eg. NFS

Stateful vs Stateless

- Advantage of Stateful over Stateless:
  - Increased performance
  - File info is cached in memory ==> fewer disk accesses
  - Stateful server knows if a file was opened for sequential access and can thus read ahead the next blocks

Distinctions Between Stateful & Stateless Service

- Failure Recovery: in case of a crash
  - 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 failures and recovery are almost unnoticeable
    - A newly reincarnated server can respond to a self-contained request without any difficulty
    - No distinction between a slow server and a recovering server from a client's point of view

Distinctions (Cont.)

- Penalties for using the robust stateless service:
  - longer request messages
  - slower request processing
- 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

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

An Example: AFS

- A distributed computing environment (Andrew) under development since 1983 at Carnegie-Mellon University, purchased by Transarc, and then by IBM and released as **Transarc DFS,** now open sourced as OpenAFS
- AFS tries to solve complex issues such as uniform name space, location-independent file sharing, client-side caching (with cache consistency), secure authentication (via Kerberos)
  - Also includes server-side caching (via replicas), high availability
  - Can span 5,000 workstations

AFS (Cont.)

- Clients are presented with a partitioned space of file names: a **local name space** and a **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 (Cont.)

- Local name space is small, distinct for each workstation, contain programs for autonomous operation, better operation, and privacy
- Servers collectively are responsible for the storage and management of the shared name space
- A key mechanism selected for remote file operations is to try to cache entire files
  - Reduces file-open latency
  - Allows read/write from/to cache without involving server

AFS Shared Name Space

- Andrew's **volumes** are small component units associated with the files of a single client
  - Volumes are mounted together similar to mounting partitions in UNIX (but with finer granularity)
- A **fid** identifies a Vice file or directory - A fid is 96 bits long and has three equal-length components (32 bit each):
  - 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 in a volume-location database, and the information is replicated on each server

AFS File Operations

- Andrew tries to cache entire files form servers
  - A client workstation interacts with Vice servers only during opening and closing of files
- *Venus* - (client process intercepting file-system calls) caches files from Vice when they are opened, and stores modified copies of files back when they are closed
- Reading and writing bytes of a file are done by the kernel without Venus intervention on the cached copy
- Venus caches contents of directories and symbolic links, for path-name translation
- Exceptions to the caching policy are modifications to directories (eg .changing permissions) that are made directly on the server

AFS Implementation

- Client processes are interfaced to a UNIX kernel with the usual set of system calls
  - Kernel modified to detect references to Vice files and forward requests to Venus process
- Venus carries out path-name translation component by component
  - Map volumes to server locations
  - If volume not in cache, contact any server & request location info
- 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

AFS 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
- A single client-level process serves all requests from that client

Any Questions?

Reading Assignment

- Read chapter 16 and 17 from Silberschatz.

Acknowledgements

- "Operating Systems Concepts" book and supplementary material by Silberschatz, Galvin and Gagne.