

### **Chapter 16 Distributed-File Systems**

- Background
- \*Naming and Transparency
- Remote File Access
- Stateful versus Stateless Service
- File Replication
- Example Systems





#### 🏅 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)
- Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files





### **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
- 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







### **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
- 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 form the storage-devices







# Naming Schemes — Three Main Approaches

- Files named by combination of their host name and local ·name; guarantees a unique systemwide name.
- Attach remote directories to local directories, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently.
- Total integration of the component file systems.
  - $\buildrel {\buildrel {\uildrel {\uildrel {\uildrel {\uildrel {\uildrel {\uildrel {\uildrel {\uildrel {\uildr$
  - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.





## Remote File Access

- 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.





- 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.



## 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 graphes.
  - 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.



## Comparing Caching and Remote Service

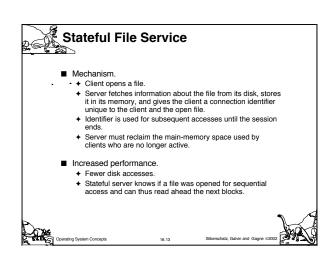
- In caching, many remote accesses handled efficiently by the local cache; most remote accesses will be served as fast as local ones.
- Servers are contracted only occasionally in caching (rather than for each access).
  - ◆ Reduces server load and network traffic.
  - ◆ Enhances potential for scalability.
- 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).

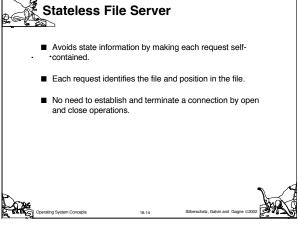


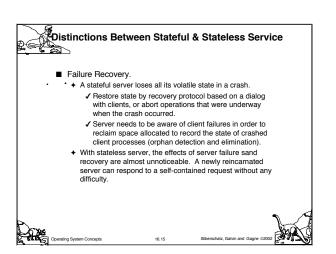
## Caching and Remote Service (Cont.)

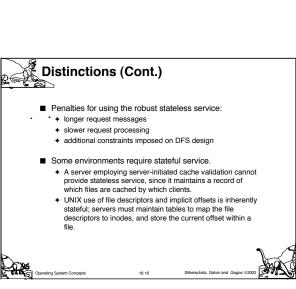
- Caching is superior in access patterns with infrequent writes. With frequent writes, substantial overhead incurred to overcome cache-consistency problem.
- Benefit from caching when execution carried out on machines with either local disks or large main memories.
- Remote access on diskless, small-memory-capacity machines should be done through remote-service method.
- In caching, the lower intermachine interface is different form the upper user interface.
- In remote-service, the intermachine interface mirrors the local user-file-system interface.

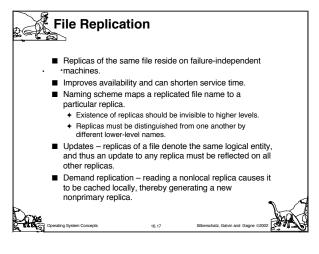


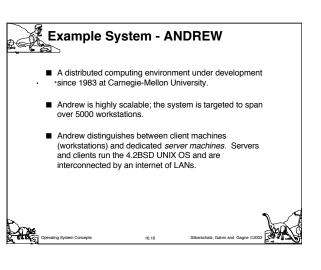












## ANDREW (Cont.)

- 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.
- Servers collectively are responsible for the storage and management of the shared name space.



### ANDREW (Cont.)

- Clients and servers are structured in clusters interconnected by a backbone LAN.
- A cluster consists of a collection of workstations and a cluster server and is connected to the backbone by a router.
- 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.



## ANDREW 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.





- Andrew caches entire files form servers. A client ·workstation interacts with Vice servers only during opening and closing of files.
- Venus 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 that are made directly on the server responsibility for that directory.



## 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 nameto-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.

