Operating System Design

Dr. Jerry Shiao, Silicon Valley University

Distributed File Systems

- Overview
- Examine a Distributed Computer System infrastructure in the Distributed File System (DFS) model.
  - Distributed File System: File system where Multiple Users share Files and Storage Resources when the files are physically dispersed among sites of a Distributed System.
- DFS Infrastructure: Using Network construct and Low-Level Protocols to transfer Messages between Systems.
- Naming and Transparency
  - Mapping between Logical and Physical Objects.
  - Location Transparency and Location Independence.
- Remote File Access
  - Remote Service using Remote Procedure Call (RPC).
- Stateful versus Stateless Service
  - Server-side Storage: Stateful or Stateless File Service
- File Replication
  - Using Redundancy for Improving Availability.

Distributed File Systems (DFS)

- Distributed System is Collection of Loosely Coupled Computers – Interconnected by Communication Network.
  - Share physically dispersed files using DFS.
- A DFS manages set of dispersed storage devices.
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces.
  - These constituent storage spaces corresponds to Set of Files.
  - Component Unit: Smallest set of files that can be stored on a system.
- Structure of a DFS:
  - Service: Software entity running on one or more systems providing particular type of function to clients.
  - Server: Service software running on a computer system.
  - Client: Process invoke service using a set of operations (i.e. 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.

Distributed File Systems (DFS)

- Structure of a DFS (Cont)
- Service carried out across the network.
- Instead of a centralized data repository, the system has multiple and independent storage devices.
- Multiplicity and dispersion of its servers and storage devices should be made invisible: Client Interface does not distinguish between local and remote files.
- Performance Measurement:
  - Most important: Amount of time to satisfy Service Request.
    - Time to deliver request to a server, processing time at the server, time to get response back to the client.
  - DFS Transparency: Ideal DFS would be comparable to a conventional file system.

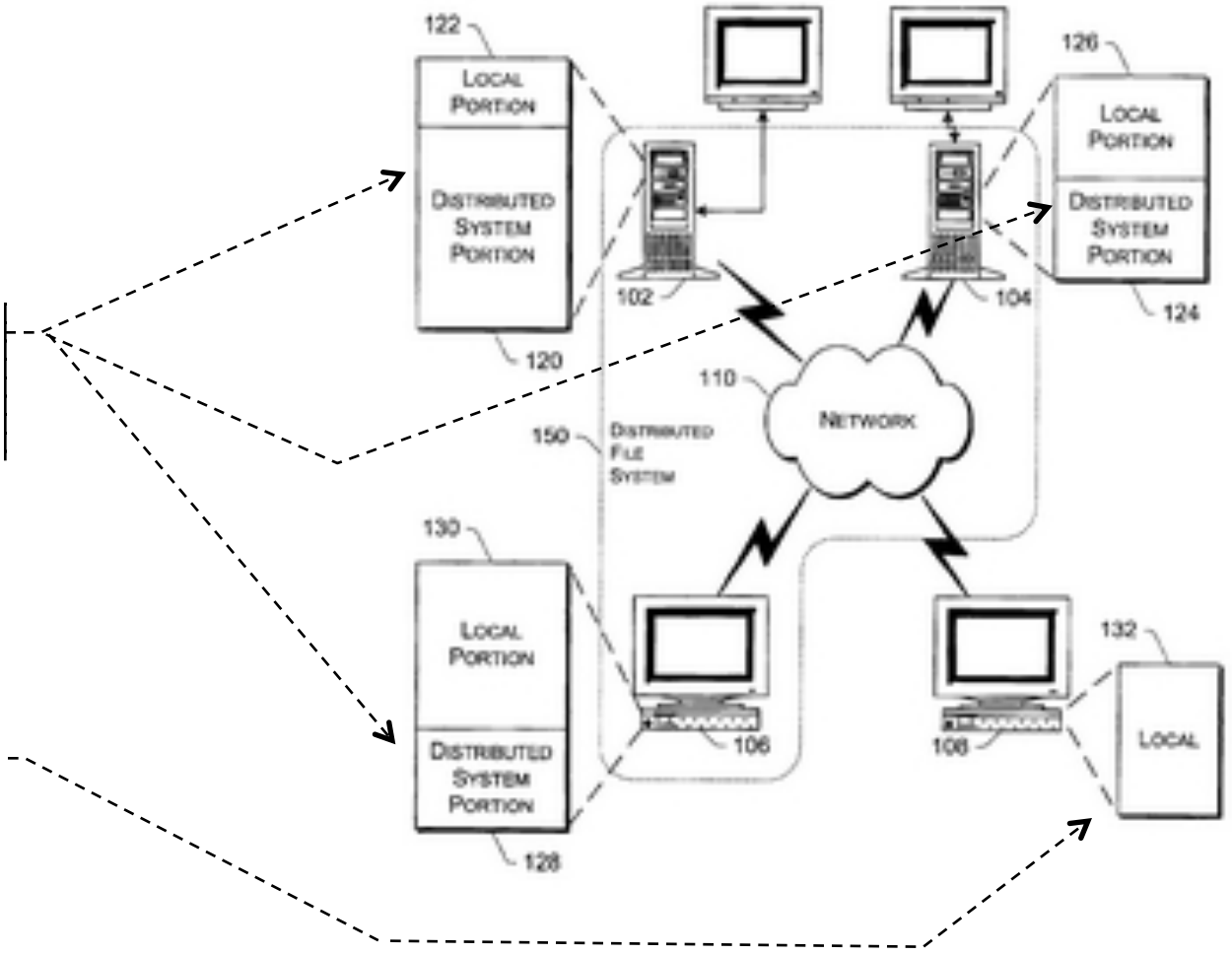
Distributed File Systems (DFS)

TQN: false statement about DFS?
The client needs to know about the protocol & structure & which server to connect to on the server.

- Structure of a DFS

Server: Service software running on a system.

Client: Process invoke service using a set of operations



Distributed File Systems (DFS)

- Naming and Transparency
- **Naming** – Mapping between logical and physical objects.
  - Computer System: Manipulates physical blocks of data on disk tracks.
  - Users: Uses logical data objects represented by File Names.
    - Mapped to a lower-level Numerical Identifier that is further mapped to physical blocks of data on disk tracks.
- **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.
  - In DFS, the range of the name mapping is expanded to include specific remote system where the file is stored.
  - DFS treating files as abstraction levels leads to possibility of File Replication.
  - 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. ↑ closest physical location

Distributed File Systems (DFS)

- Naming Structures
- **Location Transparency** – File Name does not reveal the file's physical storage location.
  - Usually static, does not support file migration.
  - Files associated permanently with a specific set of disk blocks.
  - Movement of File Names and disks are manual.
- **Location Independence** – File Name does not need to be changed when the file's physical storage location changes.
  - Dynamic Mapping: Map the same File Name to different locations.
  - Stronger property than Location Transparency.
  - Andrew File System supports File Mobility.
    - Protocol between Client and Server provides migration, without changing User-Level Names or Low-Level names of the corresponding files.

Distributed File Systems (DFS)

- Naming Structures (Cont)
- Difference between Location Transparency and Location Independence.
  - Location Transparency:
    - Provides users with convenient way to share remote files, by simply naming files in location-transparent manner, as though files were local.
    - Sharing somewhat cumbersome, logical names statically attached to physical storage device.
    - Overall system-wide storage space looks like a single virtual resource.
    - Benefit is the ability to balance the utilization of disks across the system.

Distributed File Systems (DFS)

- Naming Structures (Cont)
- Difference between Location Transparency and Location Independence.
  - Location Independence:
    - Provides better abstraction file files (divorce of data from location).
    - Logical data container not attached to specific storage location.
    - Overall, system-wide storage space is a single virtual resource.
    - Naming Hierarchy separated from Storage-Devices Hierarchy and from the inter-computer structure.
  - Diskless Computer Systems: Servers provide all files, including the Operating System Kernel.
    - Special boot code in flash enables networking and retrieves the kernel from File Server and loaded.
    - Kernel uses DFS to make ALL the Operating System files available.
    - Client cost low (no disk required): Operating System upgrade simple.
  - Current trend uses combination of local disks and remote file servers.
    - Operating System and Networking Software stored locally.
    - File system for User Data and System (utilities) files on remote File System.
    - Possible store common System Utilities (Word Processor, Web Browser) also locally: Limit network access and improve system throughput.

Distributed File Systems (DFS)

- Naming Schemes
- Scheme 1: Files named by combination of their host name and local name; guarantees a unique systemwide name.
  - host:local_name (local_name is UNIX-like path).
  - Neither Location Transparent or Location Independent.
- Scheme 2: Sun's Open Network Computing (ONC) NFS File System - Attach remote directories to local directories, giving the appearance of a coherent directory tree.
  - Automount Feature: Mounts are done on demand, based on Table of Mount Points and File-Structure names.
  - NFS structure Administratively Complex:
    - Highly unstructured: Remote directory attached anywhere on local directory.
    - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.
    - Accreditation mechanism control which remote system allowed to attach.
- Scheme 3: Total integration of the Component File Systems
  - A single global name structure spans all the files in the system.
  - Difficult to maintain: UNIX Device Files and machine-specific binary directories.

Distributed File Systems (DFS)

- Implementation Techniques
- UNIX Systems Hierarchical Directory Tree:
  - Mapping of File Name to associated location. Mount Table:
  - Aggregate sets of files into Component Units and provide mapping on a component-unit basis instead of single-file basis.
  - Administratively manageable.
- Location-Independent File Identifiers: Technique for replication and local caching.
  - File Names → Lower-Level File Identifiers that indicate the Component Unit. on remote system
  - Second Level Component Units → Locations.
  - Hierarchy under Component Unit Migration for consistent updates.
  - Using structured names: Individual parts of the name are unique only within context of the rest of the parts (i.e. extend name with timestamp).
- Andrew File System :
  - Aggregates files into Component Units.
  - Use Lower-Level Location-Independent File Identifiers.
  - File identifiers made unique by adding more bits.

Distributed File Systems (DFS)

- Remote File Access
- Remote-service mechanism: disk access for each access request.
  - Remote Procedure Call (RPC)
  - DFS needs caching to reduce Network Traffic and Disk I/O.
- Cache Scheme: ← only blocks, not entire file.
  - Copy of data brought from server to local client system.
  - Replacement algorithm (Least-Recently-Used Algorithm) keeps cache bounded.
  - 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.
    - More cached data, more overhead with keeping cached copies consistent.
    - Network Virtual Memory similar to Demand-Paged Virtual Memory.
  -

Distributed File Systems (DFS)

- Cache Scheme: (Cont)
  - DFS cached data granularity can be blocks of a file to entire file.
    - More data cached then requested (Disk Read-Ahead).
    - Large Caches:
      - Increase hit ratio, but increase miss ratio (more data transferred).
      - Increase consistency problems.
      - Large Caches, large block size useful.
    - Small Caches:
      - Large block size less useful, since less blocks can fit in the cache.
    - Large Block sizes:
      - Network Transfer Unit could require overhead in disassembly and reassembly.
    - AFS caches files in 64KByte chunks.
    - UNIX Block sizes are 4KBytes and 8KBytes.
      - 8KBytes Cache benefical for Large Caches over 1MBytes.

Most DFS uses memory-cache, not disk cache

Distributed File Systems (DFS)

- Cache Location
- 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 larger and less expensive 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.

Distributed File Systems (DFS)

Q: What is wrong for memory cache?
A: Memory cache is more reliable than disk cache.

- Cache Location
- NFS: Hybrid of Caching and Remote Service.
  - NFS implementation based on Remote Service (RPC).
  - Augmented with Client-Side and Server-Side memory caching for performance.
  - NFS and most implementations does not provide Disk Caching.
  - Solaris 2.6+ NFS:
    - Client-Side Disk-Caching Option (cachefs File System).
    - NFS Client reads blocks of a file from Server:
      - Cache in Memory.
      - Cache in Disk.
      - Disk Cache referenced when memory is flushed or system reboots.
      - When Cache-Miss, RPC sent to Server to retrieve Data Blocks and written to Memory Cache and Disk Cache.

Distributed File Systems (DFS)

- Cache-Update Policy
- Policy to write modified Data Blocks back to Server's Master Copy has critical effect on system performance and reliability.
- **Write-through** – Write data through to disk as soon as they are placed on any cache.
  - Reliable, but poor performance.
  - Write must wait until information sent to the Server, causing poor write performance.
  - Equivalent to Remote Service for Write Access and Caching for Read Access.
- **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, ONLY last update is written.
  - Poor reliability; unwritten data will be lost whenever a user machine crashes.

great performance

minimize network congestion.

Distributed File Systems (DFS)

- Cache-Update Policy
- **Delayed-write** – (Cont)
  - Variations differ in WHEN modified Data Blocks are flushed to the Server.
    - Variation – Flush Data Block when it is removed from the Clients Cache.
      - Good Performance.
      - Client Cache might not be flushed for a long time.
    - Variation – Scan cache at regular intervals and flush blocks that have been modified since the last scan (UNIX).
      - NFS and Metadata (Directory Data and File-Attribute Data).
        - Metadata changes are issued synchronously to the Server.
        - Avoids File-Structure and Directory-Structure corruption during Client or Server system crashed.
      - NFS and cachefs: Writes written to local Disk Cache area when written to the Server

Distributed File Systems (DFS)

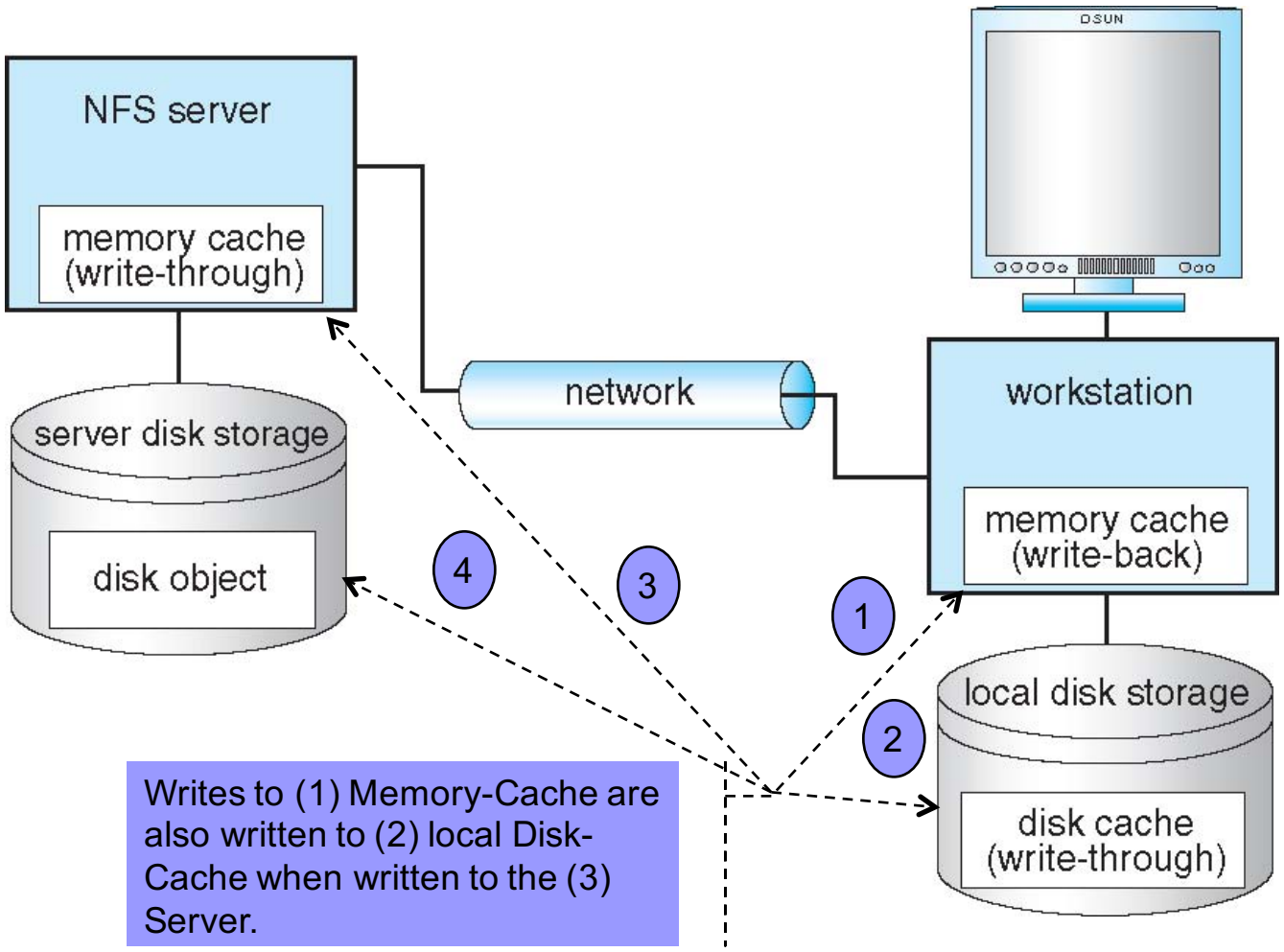
- Cache-Update Policy
- **Delayed-write** – (Cont)
  - Variation – **Write-on-close:** Writes data back to the server when the file is closed.
    - Best for files that are open for long periods and frequently modified.
    - AFS uses Write-on-close:
      - Files open for short periods or rarely modified, no improvement of network traffic.
      - Closing process MUST delay until file is written (reduce performance of Delay-Write Policy).
      - Performance advantage over Delay-Write Policy when files open for long periods and modified frequently with more frequent flushing.

Distributed File Systems (DFS)

- Cache-Update Policy

Tan : which one is false about Write Thru. Cache update policy?

Ans: It has a poor performance in Read.



Distributed File Systems (DFS)

changed?

- Consistency
- Is locally cached copy of the Data consistent with the Master Copy?
- Client determines its cached Data is out-of-date, an up-to-date copy of Data needs to be cached.
- **Client-initiated approach** ← stateful, has a session with server
  - Client initiates a validity check.
    - Contacts Server and checks local Data are consistent with the Master Copy.
    - Frequency of Validity checking:
      - Check before every access.
      - Check on first access (File Open).
      - Checks at fixed time intervals.
- **Server-initiated approach** ← stateful as well!
  - Server records, for each client, the (parts of) files it caches.
  - When server detects a potential inconsistency, it must react.
    - Two clients conflicting modes (Read or Write) caches a File.
    - Caching can be disabled for that File and switched to Remote Server mode.

Distributed File Systems (DFS)

- Comparison of Caching and Remote Service
- Tradeoff between increased performance with decreased simplicity.
- Caching: ← use cache if need stateful.
  - 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.
  - Total network overhead lowered transmitting large Data Blocks (Caching) as opposed to transmitting series of Requests/Responses (Remote Service).
  - Disk access on Server optimized for contiguous segments of Data, as opposed to random Disk blocks.

if it is stateless, does not need cache

Distributed File Systems (DFS)

- Comparison of Caching and Remote Service
- Tradeoff between increased performance with decreased simplicity.
- Caching: (Cont)
  - Cache Consistency is problem: Access patterns has frequent Writes, Caching usually is superior.
    - When Writes are frequent, procedure to overcome consistency problems with multiple Clients will incur substantial overhead in performance, network traffic, and Server load.
  - Complicated: Lower-Level Interface is different from the Upper-Level User Interface: Data transferred between Server and Client, rather than in response to the specific needs of a File Operation.
  - Caching Beneficial: Systems have local Disks or large main memory.

Distributed File Systems (DFS)

- Comparison of Caching and Remote Service
- Tradeoff between increased performance with decreased simplicity.
- Remote Server:
  - Handles every remote access across the network; penalty in network traffic, server load, and performance.
  - Remote-Service Method used on diskless, small-memory-capacity systems.
  - Simplicity: Remote-Service Lower-Level Interface is an extension of the local File-System Interface across the network.

every time the client makes a request, server has to open the file, return, closes → overhead for each command

Distributed File Systems (DFS)

- Stateful Versus Stateless Service
- Server-Side Information when a Client access Remote Files.
- Stateful Service: When Client access Remote File, the Server tracks each File being accessed by each Client.
- Stateless Service: When Client access Remote files, the Server provides Data Blocks without knowledge on how file are used.
- Stateful Service Mechanism:
  - Client performs open() operation on a file.
  - Server fetches information about the file (File Information) from its disk, stores it in its memory, and gives the client a connection identifier unique (File Descriptor) 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 not active.
- Stateful Service Increased performance:
  - Fewer disk accesses: File information cached in main memory.
  - Stateful Server knows if a file was opened for sequential access and can thus read ahead the next blocks.

Distributed File Systems (DFS)

Q. which one is TRUE for stateless?
Ans. each request identifies the file & position in full.

- Stateless Service: (Cont)
- Stateless Service Mechanism:
  - Avoids state information by making each request self-contained.
  - Each request identifies the file and position in the file (for Read and Write Access) in full.
  - Server not required to keep Table of Open Files in Main Memory.
  - No need to establish and terminate a connection by open() and close () operations (each File Operation stands on its own).
    - Operations read() and write() results in sending remote messages.
    - Operations open() and close() are local operations (no remote messages).
  - Penalty for robust Stateless Service:
    - Request Messages are longer (each Request fully identifies the file).
    - Slower processing (no cached information in Server memory).
  - DFS Constraints:
    - Each Request identifies the Target File, therefore system-wide, low-level naming scheme must be used.
    - Duplicate File operations could be received, therefore the same operation MUST return same result (read(), write(), delete()).
  - NFS is Stateless File Service.

Distributed File Systems (DFS)

- Stateless Service: (Cont)
- Distinction Between Stateful and Stateless Service
- Failure Recovery
  - Stateful Server: Loses all its volatile state in a crash.
    - Restore state by recovery protocol based on a dialog with Clients (Clients on network waiting for response), 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).
  - Stateless Server: The effects of server failure sand recovery are almost unnoticeable. ← easier for recovery.
    - Clients on network cannot distinguish between slow Server and recovering Server (Client repeats request).
    - A newly reincarnated server can respond to a self-contained request without any difficulty.

Distributed File Systems (DFS)

- 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; Servers must maintain tables to map the file descriptors to inodes, and store the current offset within a file.
    - Reason NFS, a Stateless Service, does not use File Descriptors and includes full File operation parameters (i.e. including explicit offset in every Access).

Distributed File Systems (DFS)

can delay the replica sync/refresh but eventually has to update replicas.

- File Replication
- Replication of files on different systems in a DFS is a useful redundancy for improving availability.
  - Replicas of the same file reside on failure-independent machines.
- Multi-machine replication can benefit performance: 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.
  - At Lower-Level, replicas must be distinguished from one another by different Lower-Level names.
- Consistency of replicas can be sacrificed for availability and performance.
  - Preserving consistency creates potential for undefined blocking time.
  - Locus system (designed at UCLA): One node in the cluster designated as CSS (Current Synchronization Site) and all accesses to files in the DFS would be coordinated through CSS.
- Demand replication – reading a nonlocal replica causes it to be cached locally, thereby generating a new nonprimary replica.

Distributed File Systems (DFS)

- Andrew File System
- A distributed computing environment (Andrew) under development since 1983 at Carnegie-Mellon University, purchased 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

Distributed File Systems (DFS)

- Andrew File System
- Clients are presented with a partitioned space of file names: a **local name space** and a **shared name space**.
- The **shared name space** is presented by the Servers, called Vice, 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.
  - Distinct for each workstation: contains system programs for autonomous operation and performance, temporary files, user private files.
- 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**.

Distributed File Systems (DFS)

- Andrew File System
- 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.
  - Clusters are used to address the problem of scaling.
  - Workstations should use the Server on their own Cluster.
- A key mechanism selected for remote file operations is whole file caching.
  - Key mechanism for Remote File operation is to use 64KByte Caches: Allows Reads and Writes to be directed to the cached copy without Server involvement.
  - Opening a file causes it to be cached, if possible, in its entirety, on the local disk.

large locality.
↓
largest cache size.

client ⟷ Vice Servers ⟷ primary server

Distributed File Systems (DFS)

- Andrew File System
- Client Mobility
  - Clients can access any file in the shared name space from any workstation.
- Security
  - Vice interface considered the boundary of trustworthiness, because no Client programs are executed on Vice System.
  - Authentication and secure-transmission functions are provided as part of connection-based communication package based on RPC paradigm.
  - Vice Server and Client communicate through encrypted messages.
    - Client information and groups are stored in a database at each Server.
- Protection
  - AFS provides Access Lists protecting directories and UNIX type file protection.
    - Access List information on user allowed or not allowed to access a directory.
    - Access Types: Read, Write, Lookup, Insert, Administer, Lock, and Delete.
- Diversity
  - Clear interface to Vice required for integration of diverse workstations and OS.
  - The local /bin directory has symbolic links to machine-specific executables residing in Vice.

Distributed File Systems (DFS)

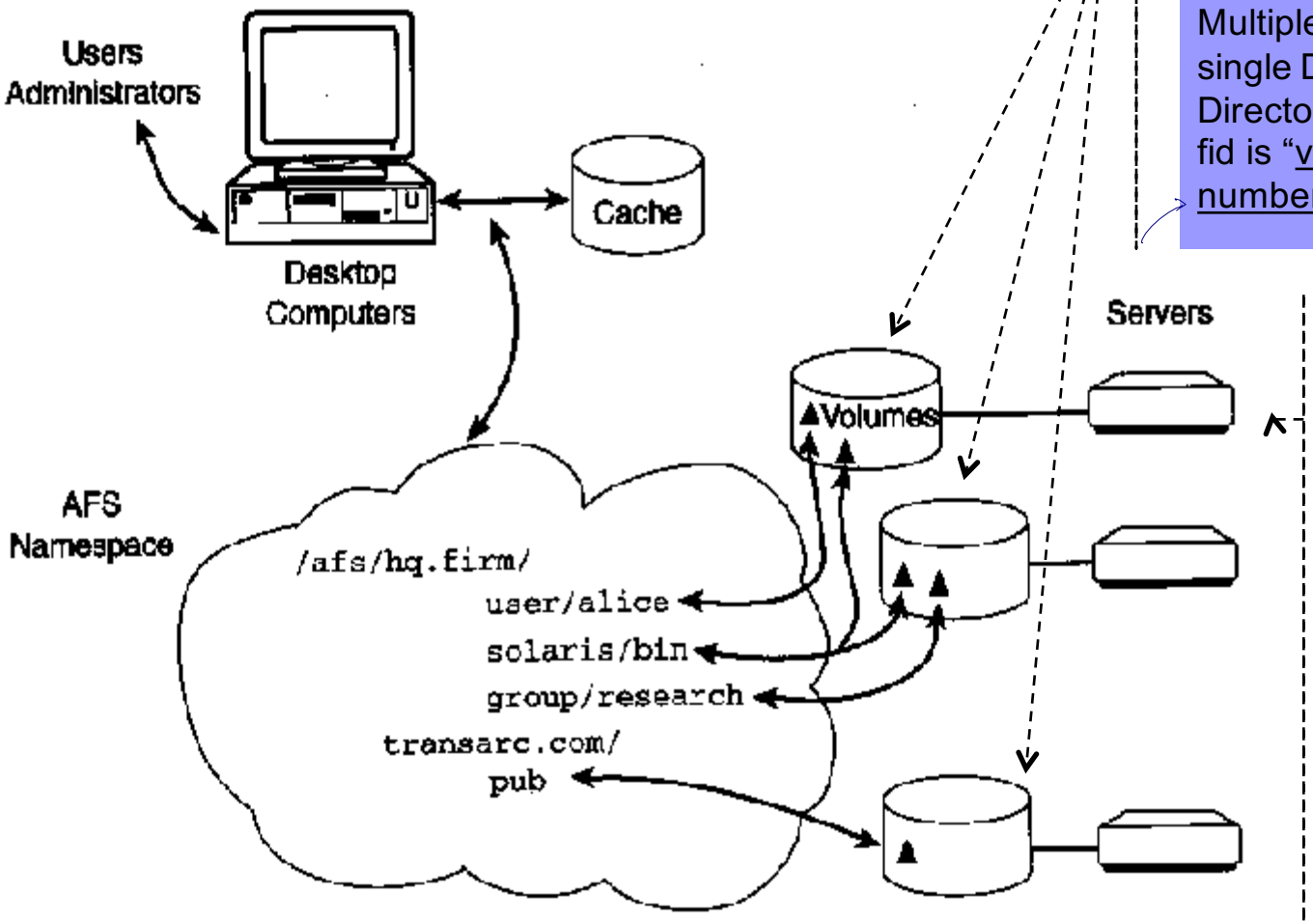
- Andrew File System
- The Shared Name Space
- AFS Shared Name Space made up of component units (volumes), with the files of a single Client.
  - Volumes reside within Disk Partition and may grow/shrink in size.
  - Volumes are similar to UNIX mount mechanism, but does not need an entire Disk Partition (containing a File System) to be mounted.
  - Volume is key administrative unit and identifies and locates individual Files.
- A **fid** identifies a Vice file or directory - A fid is 96 bits long and has three equal-length components:
  - Volume Number: Index into Volume-Location Database on each Server.
  - 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.

Distributed File Systems (DFS)

- Andrew File System
- The Shared Name Space (Cont)
- Fids are location transparent: File movements from Server to Server do not invalidate cached directory contents.
  - Location information kept on a volume basis in a Volume-Location Database replicated on each Server.
  - Volume-Location Database contains location of every volume in the system.
  - Aggregation of files into volumes keeps the location database at a manageable size.
  - Volume-Location Database specifies Servers with Read-Write copy and Read copy replication sites.
    - Read-Only replication for system-executable files and seldom-updated files.
- Volumes are migrated among Disk Partitions and Servers (balance available disk space and utilization of Servers).
  - Volume moved, original Server has temporary forwarding information.
  - During Volume movement, original Server can still handle updates.
  - During switch-over, Volume briefly disabled for switch-over modifications.
  - Volume movement is atomic, operation aborted if either Server fails.

Distributed File Systems (DFS)

- Andrew File System
- The Shared Name Space



Shared Name Space made up of component units called volumes.
Multiple volumes reside in single Disk Partition.
Directory is identified by fid. A fid is "volume number, vnode number, uniquifier".

At the end of a path-name traversal, all the intermediate directories and the target file are in the cache with callbacks on them.
Future open calls to this file will involve no network communication, unless a callback is broken on a component of the pathname.

Distributed File Systems (DFS)

- Andrew File System
- File Operations and Consistency Semantics
- Fundamental architectural principle of AFS: Caches entire files from server.
  - A client workstation interacts with Vice servers only during opening and closing of files.
  - Reading and Writing files do not cause remote interaction.
  - Key distinction for performance.
- Venus caches files from Vice when they are opened, and stores modified copies of files back to Server when they are closed.
  - Reading and writing bytes of a file are done by the kernel without Venus intervention on the cached copy.
  - Writes at some sites are NOT visible immediately at other sites.
- Venus assumes cached entries (files or directories) are valid unless notified by Server: callback policy reduces cache-validation requests received by Servers.

consistency of files is managed by the server, using the callback provided by the client to the server for notification if the file got change on the server.

Distributed File Systems (DFS)

TQ: the technique used by AFS
Ans: whole file caching.

- Andrew File System
- File Operations and Consistency Semantics (Cont)
- callback policy:
  - When Client caches a file or directory, Server updates its state information with a callback to record the caching.
  - The Server notifies the Client before allowing another Client to modify the file.
  - When Client open the file later, the Client has to get new version from Server.
- Venus contacts Vice Servers on opens of files that are not in the cache or had the callback revoked or on closes of locally modified files.
- Exceptions to the caching policy are modifications to directories that are made directly on the server responsibility for that directory.

Distributed File Systems (DFS)

- Andrew File System
- 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.
  - Venus uses mapping cache that associates volumes to Server locations.
  - Volume not in the cache requires requesting the location information from a connected Server and entering location information into the mapping cache.
    - Connection to Server is made for authentication and security purposes.
  - When the target file is cached, a copy is created on the local disk.
  - The File Descriptor of the cached copy is returned to the Client process.
- 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.
- Venus and Server processes access UNIX files directly by their inodes to avoid expensive path name-to-inode translation routine.
  - Special system calls are needed for the internal inode interface.

Distributed File Systems (DFS)

- Andrew File System
- Implementation (Cont)
- Venus manages two separate caches:
  - One for status.
    - The status cache is kept in virtual memory to allow rapid servicing of `stat()` (file status returning) system calls.
  - One for data.
    - 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.
- LRU algorithm are used to keep each of them bounded in size.
  - When a file is flushed from the cache, Venus notifies the Server to remove the callback for this file.
- File Server services handled by a daemon process that creates a thread to handle each RPC call.
- Server thread handles RPC call as whole-file transfer request.
- Crash of Server thread can paralyze the Server daemon process.

Distributed File Systems (DFS)

- Network File System (NFS) V4
- NFS V4 stateful: ← mostly memory cache (but does have disk cache)
  - Server maintains state of the Client session from file open to close.
  - NFS V4 supports open() and close().
- NFS V4 does not support "mount" protocol, allowing NFS V4 to integrate with network firewalls.
- Clients can cache File Data locally.
  - File access resolved through local cache instead of Server.
- Clients request File locks from Servers.
  - Client maintains lock until released or lease expires.
- NFS V4 support mandatory File locks (Windows).
  - Server delegates responsibility for File's lock and contents to the Client that requested the lock.
  - Other Clients must request lock access from responsible Client.
- NFS V4 based on TCP.
  - Adjusts to varying traffic loads on the network.
  - Client responsibility reduces load on Server and improves cache coherency.

Distributed File Systems (DFS)

- Summary
- DFS is a File-Service System whose Clients, Servers, and Storage Devices are connected through a network as a distributed system.
- DFS appear to its Clients as a conventional centralized File System.
- Client interface does not distinguish between local and remote files.
- Naming scheme in DFS attaches remote directories to local directories, giving the appearance of a coherent directory tree.
- Request to handle remote files are through:
  - Remote Service: Request for access are delivered to the Server, the Server performs the access, and results forwarded back to the Client.
  - Caching: Request for access are performed on the cached copy. If the data needed to satisfy the access request are not already cached, then a copy of the data is brought from the Server to the Client system.
    - Repeated access to the same information can be handled locally, without additional network traffic.
    - LRU replacement policy keeps the cache size bounded.
    - Problem with keeping cached copies consistent with master copy.

Distributed File Systems (DFS)

- Summary (Cont)
- Storing Service-Side information are either:
  - Stateless: The Server provides Data blocks as requested by the Client without knowledge of how the Data blocks are used.
  - Stateful: The Server tracks each file being accessed by each Client.
- Replication used for redundancy and improving availability (selecting a nearby replica to service an access requests).
- Andrew File System (AFS) is characterized by location independence and location tranparency.
  - Caching and replication are used to improve performance.