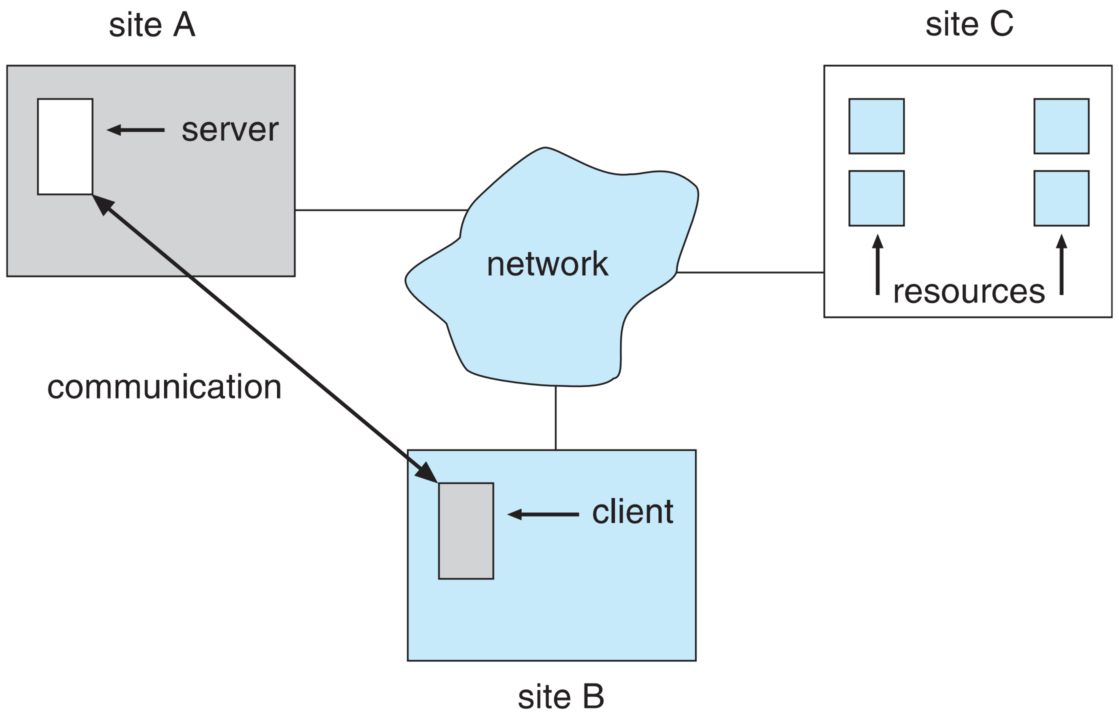
Distributed file systems

Last modified: 22.06.2017

Overview

- **Distributed system** is collection of loosely coupled processors interconnected by a communications network



Distributed File System

- **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. The **component unit** – the smallest set of files, which are stored in one machine, independently of other units.
- Challenges include:
  - Naming and Transparency
  - Remote File Access

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. should 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 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

Two important notions regarding name mappings in a DFS:

- **Location transparency** – file name does not reveal the file's physical storage location
- **Location independence** – file name does not need to be changed when the file's physical storage location changes

Naming Schemes — Three Main Approaches

1. Files named by combination of their host name and local name; guarantees a unique system-wide 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

Remote File Access

- **Remote-service mechanism** is one transfer approach
- 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
    - Could be called **network virtual memory**

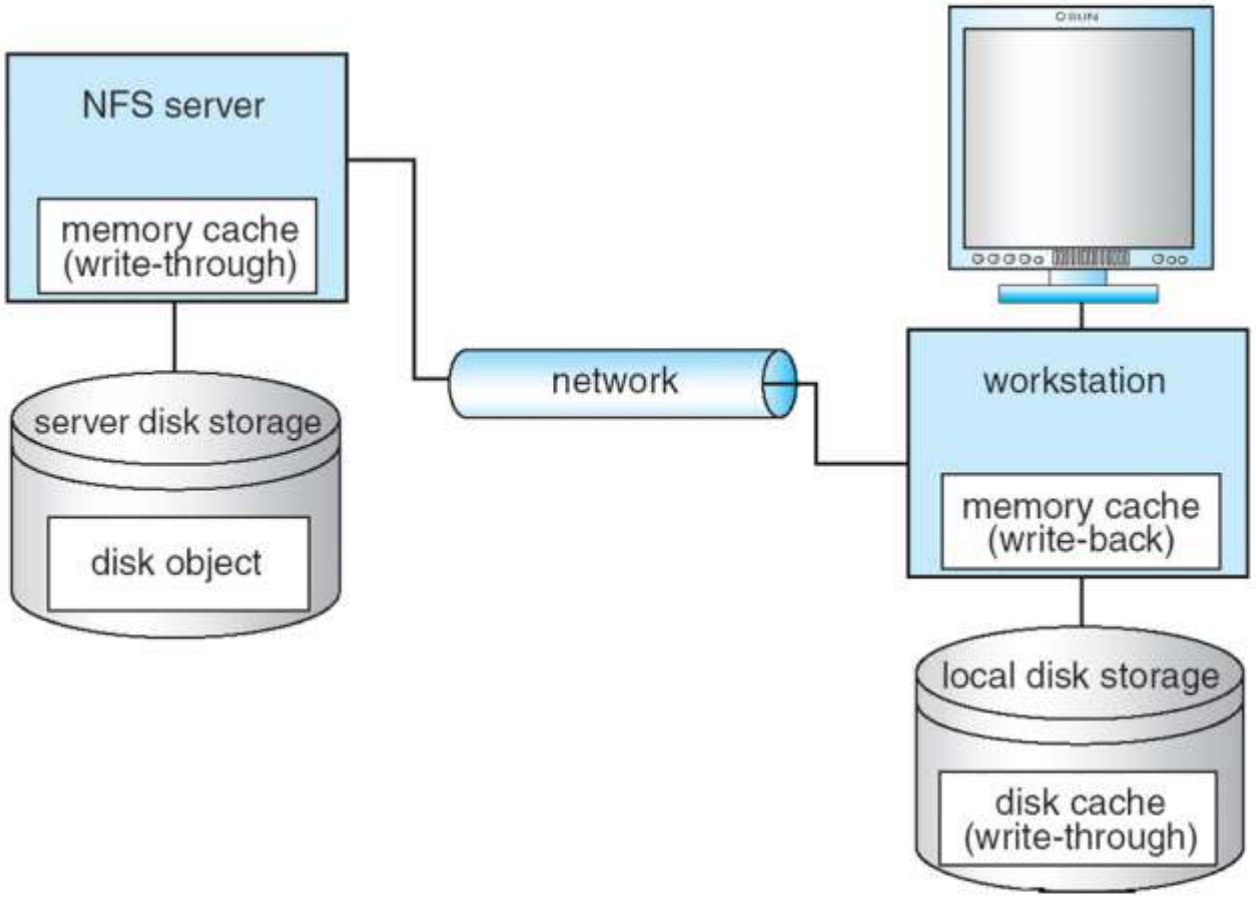
Cache Location – Disk vs. Main Memory

- 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

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** (**write-back**) – 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 (e.g. AFS)
    - Best for files that are open for long periods and frequently modified

Cachefs and its Use of Caching


NFS server
memory cache (write-through)
server disk storage
disk object
network
workstation
memory cache (write-back)
local disk storage
disk cache (write-through)

Consistency

- Is locally cached copy of the data consistent with the master copy?
- Two approaches to data consistency verification:
  - **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 by disabling caching for that particular file (switching to a 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 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 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
- 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; servers must maintain tables to map the file descriptors to inodes, and store the current offset within a file

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 non-primary replica

Data deduplication

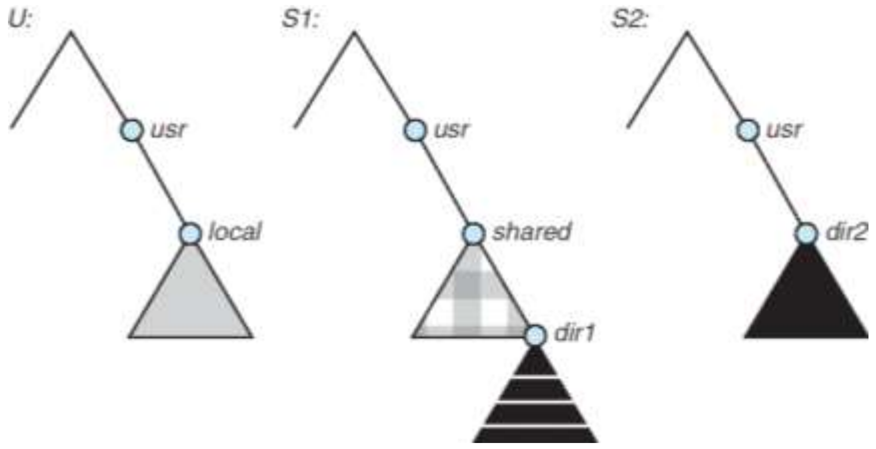
- Data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data. The deduplication process is intended to be transparent to end users and applications.
- This technique is used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent. In the deduplication process, unique chunks of data, or byte patterns, are identified and stored during a process of analysis. As the analysis continues, other chunks are compared to the stored copy and whenever a match occurs, the redundant chunk is replaced with a small reference that points to the stored chunk. Given that the same byte pattern may occur dozens, hundreds, or even thousands of times (the match frequency is dependent on the chunk size), the amount of data that must be stored or transferred can be greatly reduced.
- Methods used:
  - Chunking
  - Client backup deduplication. The deduplication hash calculations are initially created on the source (client) machines. Files that have identical hashes to files already in the target device are not sent
  - Primary storage and secondary storage. Primary storage systems are designed for optimal performance, rather than lowest possible cost. Secondary storage systems contain primarily duplicate, or secondary copies of data.

The Network File System (NFS)

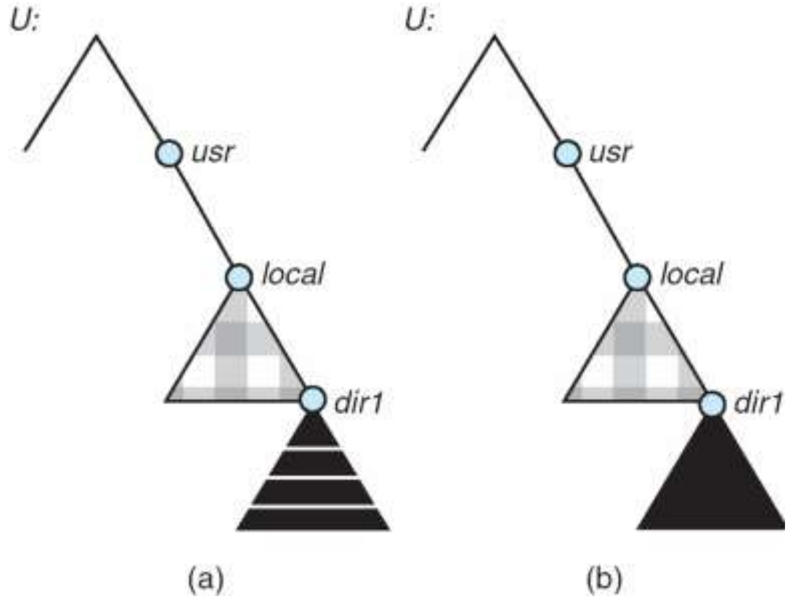
- An implementation and a specification of a software system for accessing remote files
- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
  - A remote directory can be mounted over a local file system directory
    - The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
  - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed transparently
- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures. This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

Example – NFS mounting

Three independent file systems
at servers: U, S1, S2

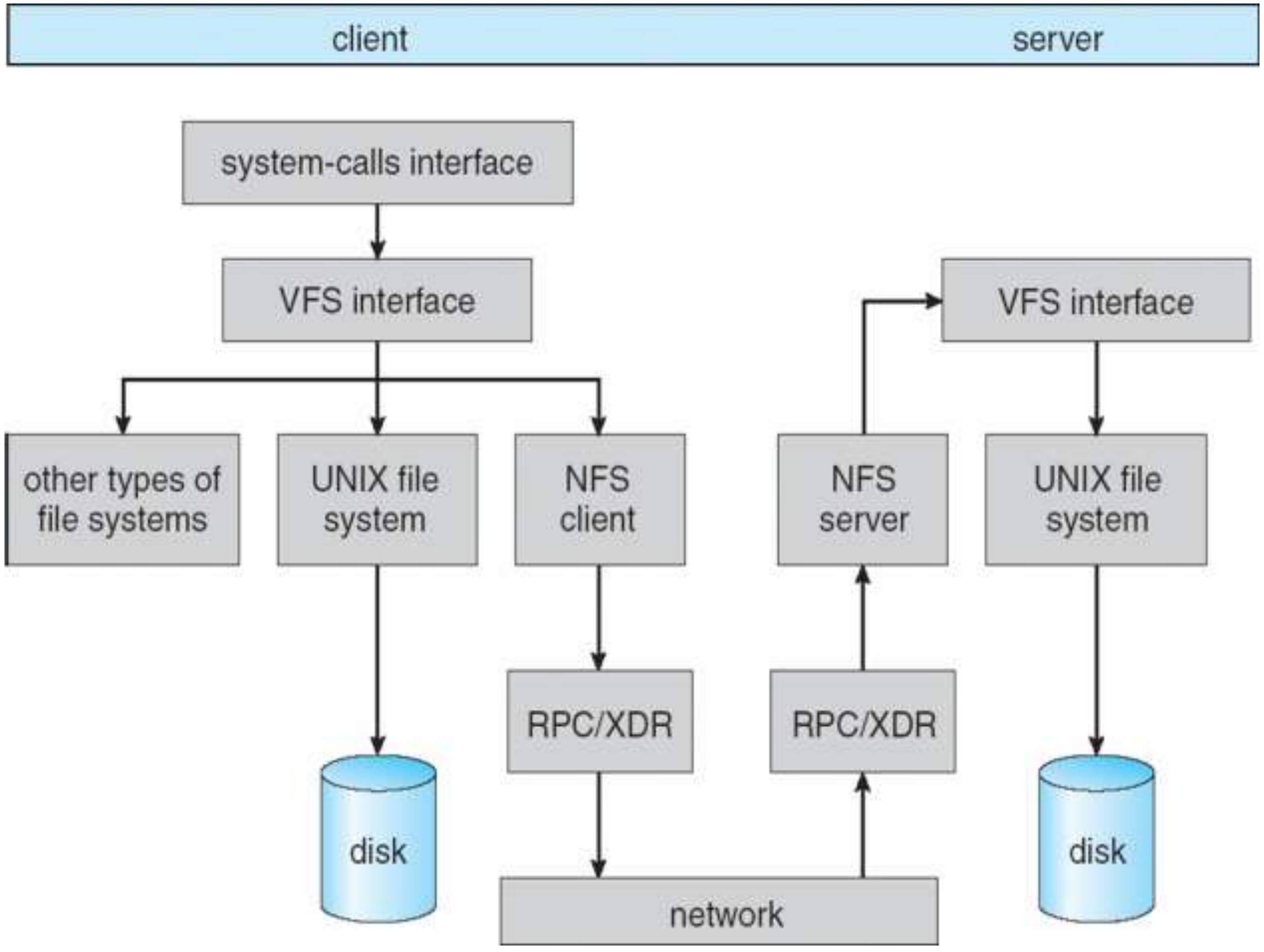


a) b)



a) Mounting directory
S1:/usr/shared/dir1 at U:/usr/local
b) cascaded mounting of directory
S2:/usr/shared/dir2

Schematic View of NFS Architecture



NFS Remote Operations

- NFS pathname translation:
  - Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
  - To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names
- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to he remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
  - Cached file blocks are used only if the corresponding cached attributes are up to date
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk