Distributed Systems
Delhi Technological University
Distributed File System

Instructor: Divyashikha Sethia
divyashikha@dce.edu

Introduction

- Distributed systems share data
- Distributed file systems allow multiple processes to *share data* over long periods of time in a *secure* and *reliable way*

Agenda

- Architecture
- Communication
- Synchronization
- Consistency and Replication

Architecture

- Client Server Architecture
- Cluster Based Distributed File System

Client Server Architecture

•Many distributed files systems are based on client-server architectures.

•Sun Microsystem's **Network File System (NFS)** being one of the most widely-deployed ones for UNIX-based system

- Each file server provides a standardized view of its local file system.
- Server provides communication protocol that allows clients to access files stored on a server
- Heterogeneous processes, on different operating systems and machines, can share common file system

Client Server Architecture

Two Basic models:

1. The remote access model
2. The upload/download model.

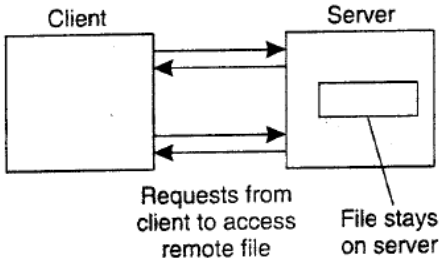
Client Server Architecture

Remote Access Model:

-Clients are normally unaware of actual location of files

-Client is offered only an interface containing various file operations but server is responsible for implementing those operations

- Example: NFS

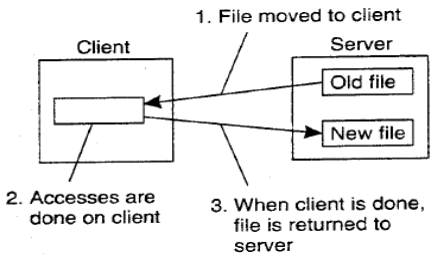


Client Server Architecture

Upload/Download Model:

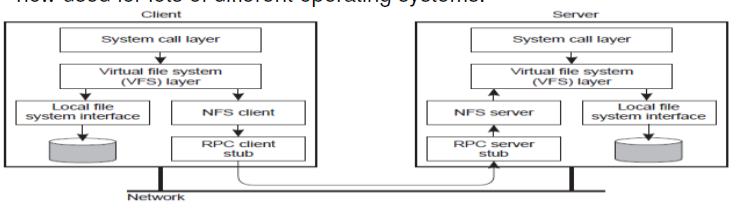
-Client accesses a file locally after downloading from server

-When done uploads file to server



Example: NFS Architecture

NFS is implemented using the Virtual File System abstraction, which is now used for lots of different operating systems.



Client Side:
-client accesses file system using system calls provided by local OS

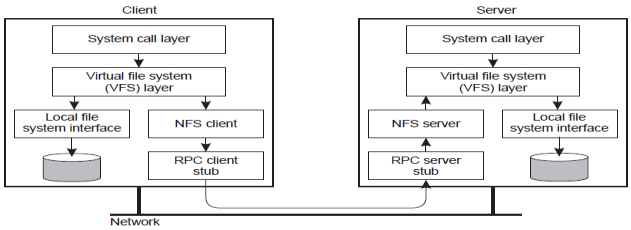
-local UNIX file system interface is replaced by an interface to Virtual File System (VFS)

- VFS either interfaces with local file system or NFS client, which takes care of handling access to files stored at a remote server

- In NFS, all client-server communication is done through RPCs. NFS client implements NFS file system operations as RPCs to server

Example: NFS Architecture

NFS is implemented using the Virtual File System abstraction, which is now used for lots of different operating systems.



Server Side:
-NFS server is responsible for handling incoming client requests

- RPC stub unmarshals requests and NFS server converts them to regular VFS file operations that are subsequently passed to VFS layer.

- VFS is responsible for implementing a local file system in which actual files are stored

NFS: File System Model

-Files are treated as uninterpreted sequences of bytes.

-Hierarchically organized into a naming graph in which nodes represent directories and files.

-Supports hard links as well as symbolic links, like any UNIX file system.
(http://linuxgazette.net/105/pitcher.html

- Files are named, but are otherwise accessed by means of a UNIX-like file handle

- To access a file, a client must first look up its name in a naming service and obtain the associated file handle.

NFS File Operations

Oper.	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Remove	Yes	Yes	Remove a file from a file system
Rmdir	Yes	No	Remove an empty subdirectory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name in a symbolic link
Getattr	Yes	Yes	Get the attribute values for a file
Setattr	Yes	Yes	Set one or more file-attribute values
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

Cluster-Based File Systems

-NFS based on traditional client-server architecture

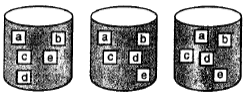
-Can be enhanced with few clusters

- With very large data collections, simple client-server approach is slow. For speeding up file accesses, distributed files across several clusters so that files can be fetched in parallel

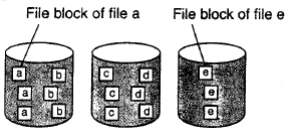
Cluster-Based File Systems

File distribution techniques:

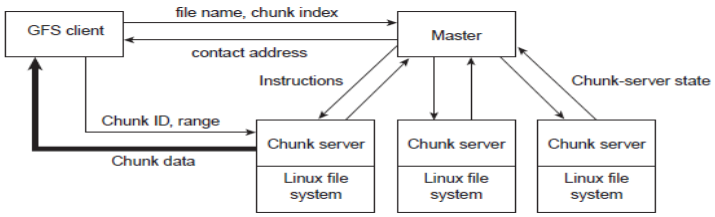
1. File stripping technique - single file is distributed across multiple servers so that it becomes possible to fetch different parts in parallel
- Useful if application organized to support parallel support and file has regular structure (dense matrix)



2. Partition whole file across different servers
- What happens in data centers of Amazon and Google – with reads and updates to massive number of files and computers might malfunction



Example: Google File System



The Google solution

Divide files in large 64 MB chunks, and distribute/replicate chunks across many servers:

- The master maintains only a (file name, chunk server) table in main memory) minimal I/O

- Files are replicated using a primary-backup scheme; the master is kept out of the loop

GFS..

-Google files tend to be very large, commonly ranging up to multiple gigabytes, where each one contains lots of smaller objects.

-Updates to files usually take place by appending data rather than overwriting parts of a file.

-Server failures are common

- GFS master is contacted only for metadata information – client passes file name and chunk index and expects contact of chunk server

- GFS master does not keep accurate account of chunks only occasionally contacts chunk servers to get information of what they store

GFS..

- Master initially allocates chunks to chunk server and is up to date

- Chunks are also replicated

- Over time if chunk server may crash and for simplicity they do not update status on master

- Master polls and gets up to date later

- Due to the chunks being replicated there is high probability that chunk will be available on one of the chunk server

GFS..

Scalability:

•Master is not a bottleneck

i) Bulk of actual work done by chunk server since client directly contacts chunk server for chunk access and updates

- Chunks are replicated using primary-backup scheme

- For update client contacts nearest chunk server holding data and pushes update which further pushes updates to the next nearest server

- Once all updates have been propagated client pushes updates to the primary chunk server

- Primary assigns sequence number to the update and passes it to all backups

GFS..

Scalability:

2) Hierarchical name space table is (hierarchical) name space for files is implemented using a simple single-level table, in which path names are mapped to metadata (such as the equivalent of inodes in traditional file systems).

- maintained in memory

-Updates to data are logged to persistent storage.

- When the log becomes too large, a checkpoint is made by which the main-memory data is stored in such a way that it can be immediately mapped back into main memory

- Leads to reduced I/O on GFS master

•

GFS..

•Single master to control a few hundred chunk servers

• Google organized into smaller services that are mapped onto collection of clusters that work together

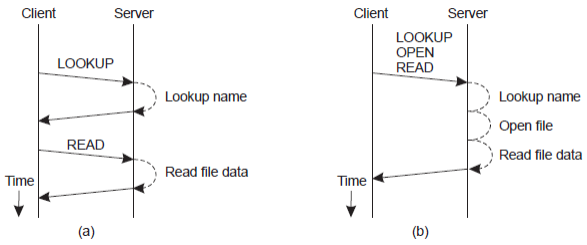
Agenda

- Architecture
- Communication
- Synchronization
- Consistency and Replication
- Fault Tolerance

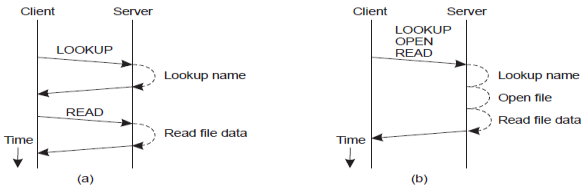
RPCs in File Systems

•Many (traditional) distributed file systems deploy remote procedure calls to access files.
•For wide-area networks alternatives to be exploited.

-Reason for RPC mechanism:
To make system independent from underlying operating systems, networks and transport protocols



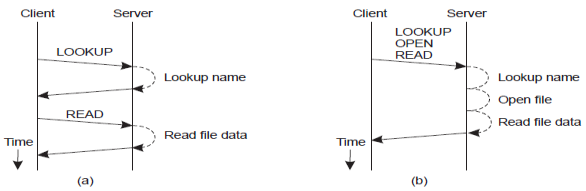
RPC is NFS



➢***Open Network Computing RPC (ONC RPC)***
•Every NFS operation can be implemented as a single RPC to file server

(a)NFS ver3 read data from a file for the first time (2 RPCs required):
- client first looks up file handle using the lookup operation
- issue a read request

RPC is NFS



(b) NFS ver 4: Compound procedures by which several RPCs can be grouped into a single request.

- client combines lookup and read request into a single RPC
- After file handle lookup, passed to open operation, after which server continues with read operation.
- Only two messages exchanged between client and server.
- Operations grouped together in compound procedure are handled in the order as requested

The RPC2 Subsystem

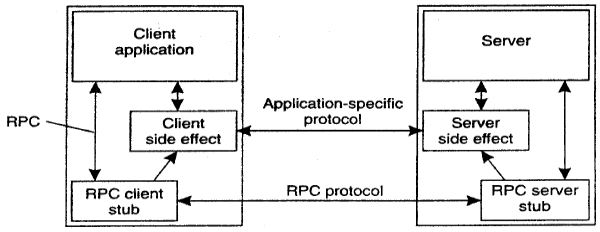
•Developed as part of the Coda file system package that offers reliable RPCs on top of the (unreliable) UDP protocol

•Each time a remote procedure is called, the RPC2 client code starts a new thread that sends an invocation request to the server and subsequently blocks until it receives an answer

•Server regularly sends back messages to the client to let it know it is still working on the request

•Client thread will notice that the messages have ceased and report back failure to the calling application if server dies

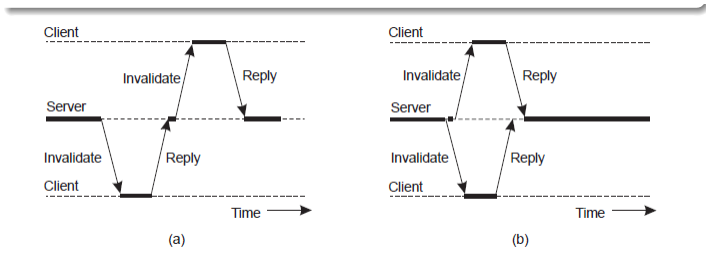
RPC2 side effects



Support for side effects:
- mechanism by which the client and server can communicate using an application specific protocol
- eg: client opens file at video server, requires setup of continuous data stream for data transfer delay to be within min and max.
- RPC2 allows client and server to set up separate connection for transferring video data to client on time.
- Connection setup is done as a side effect of an RPC call to server

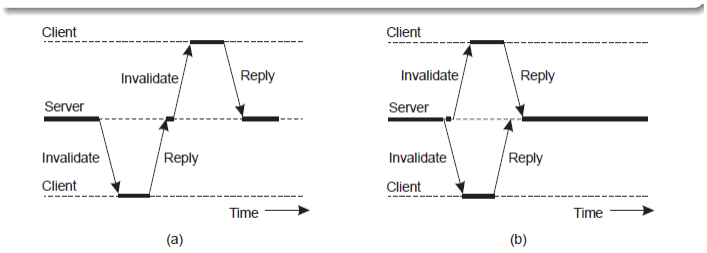
RPC2 multicasting

Support for multicasting:
-track of which clients have a local copy of a file.
-file is modified, a server invalidates local copies by notifying the appropriate clients through an RPC
(a)Invalidate one client at a time
- may be delayed considerably because the server cannot reach a possibly crashed client, but will give up on that client only after a relatively long expiration time. Meanwhile, other clients will still be reading from their local copies



RPC2 multicasting

(b)Invalidating each copy one by one
- server sends an invalidation message to all clients at the same time.
- Server notices usual expiration time that certain clients are failing to respond to the RPC, and can declare such clients as being crashed



RPC2 multicasting

Parallel RPC is implemented by means of MultiRPC system which is part of RPC2 system

- parallel invocation of RPCs is fully transparent to the callee
- At the caller's side, parallel execution is also largely transparent
- the caller invokes a number of one-way RPCs, after which it blocks until all responses have been received from the nonfailing recipients
- set up a multicast group, and send an RPC to all group members using IP multicast

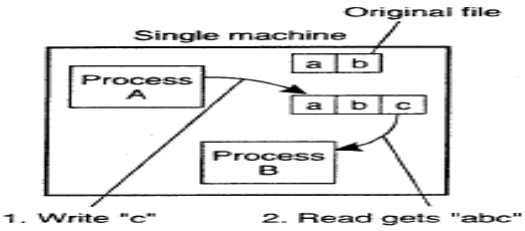
Agenda

- Architecture
- Communication
- Naming
- Synchronization
- Consistency and Replication
- Fault Tolerance

Synchronization

Important issue for Distributed File System since files are shared

Semantics for File Sharing:
-UNIX semantics
-processes sharing files - read operation follows a write operation, then read returns the value just written
- two writes happen in quick succession, followed by a read, then value read is value stored by last write with the help of system time ordering



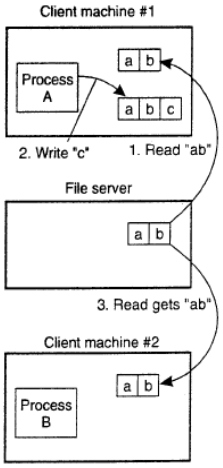
Synchronization..

UNIX semantics can be achieved easily as long as there is only one file server and clients do not cache files
- minor problem of network delays read after a write get old value

-Distributed file system requests going to a single file server is a poor design

-Hence client maintain local copies of heavily-used files in their private (local) caches

- Problem: if a client locally modifies a cached file and shortly thereafter another client reads the file from the server, the second client will get an obsolete file



Synchronization Shared Files

1. Propagate all changes to cached files back to server immediately – but is inefficient

2.Session semantics: relax the semantics of file sharing:
- Changes to an open file are initially visible only to process/ machine that modified file. Only when file is closed changes are made visible to other processes/machines. Hence read will not see immediate writes

- Problem: multiple clients updating shared file in their cache – as each file is closed in turn value is sent to server , with the final result being that of the client which most recently sent closed request to server

Synchronization Shared Files

3. Make file immutable – no writes possible only create and read possible

-create an entirely new file and enter it into the directory system under the name of a previous existing file, which now becomes inaccessible (at least under that name).

- when two processes try to replace the same file at the same time: allow one of the new files to replace the old one

- file is replaced while another process is busy reading it - reader to continue using the old file, even if it is no longer in any directory. detect that the file has changed and make subsequent attempts to read from it fail

Synchronization Shared Files

4. Use atomic transactions.

BEGIN_TRANSACTION
Read/Write system calls
END_TRANSACTION

-system guarantees that all the calls contained within the transaction will be carried out in order, without any interference from other, concurrent transactions.

- If two or more transactions start up at the same time, the system ensures that the final result is the same as if they were all run in some (undefined) sequential order.

File locking

•Central lock manager is generally deployed
•Complexity in locking comes from the need to allow concurrent access to the same file.
•Number of different locks exist

-Read and Write locks are separate

- Multiple clients can simultaneously access same part of file for read only data.

- Write lock is needed to obtain exclusive access to modify part of a file.

- Operation lock is used to request a read or write lock on a consecutive range of bytes in a file

nonblocking operation: if the lock cannot be granted due to another conflicting lock, the client gets back an error message and has to poll the server at a later time

- FIFO-ordered list maintained by the server. As soon as the conflicting lock has been removed, the server will grant the next lock to the client at the top of the list, provided it polls the server before a certain time expires.

NSFv4 File locking operations

Operation	Description
Lock	Create a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the lease on a specified lock

- lockt operation is used to test whether a conflicting lock exists. For example,
a client can test whether there are any read locks granted on a specific range of bytes in a file, before requesting a write lock for those bytes

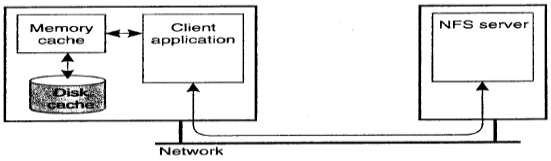
- Unless a client renews the lease on a lock it has been granted, the server will automatically remove it.

Consistency and Replication

- Client side caching
- Server side replication

Client Side Caching NFSv4

Each client can have a memory cache that contains data previously read from the server. In addition, there may also be a disk cache that is added as an extension to the memory cache, using the same consistency parameters.



-clients cache file data, attributes, file handles, and directories

Caching file data:

-Write operations can be carried out on cache and when file is closed NFS make modifications flush back to the server based on the session semantics

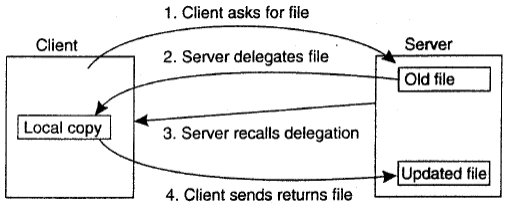
NFSv4 Open Delegation

•may delegate some of its rights to a client when a file is opened

•**Open delegation** takes place when the client machine is allowed to locally handle open and close operations from other clients on the same machine

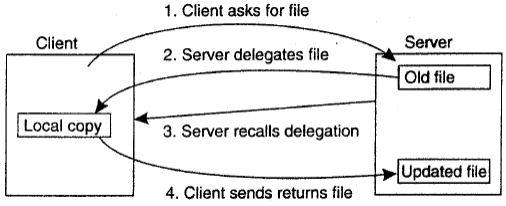
•Client machine is sometimes allowed to make such decisions, avoiding the need to contact the server

NFSv4 Open Delegation…



•If server has delegated opening of file to client that requested write permissions, file locking requests from other clients on same machine can be handled locally.
•The server will still handle locking requests from clients on other machines, by simply denying those clients access to the file.
•

NFSv4 Open Delegation…



•Server needs to be able to recall the delegation, for example, when another client on a different machine needs to obtain access rights to the file
• server must be able to callback to the client to recall delegation
  - server requires to keep track of the clients to which file is delegated
  - causes the server to be stateful instead of stateless
  - problem when the client crashes – solution form delegation based on some time of lease

Server side replication

- Less common for DFS
- File sharing primary when for reading then local client caching is better
- High degree of replication and a low read/write ratio may actually degrade performance
- file servers are generally replicated only for fault tolerance.
- Server replication of CODA – self study

THANKS