CE177

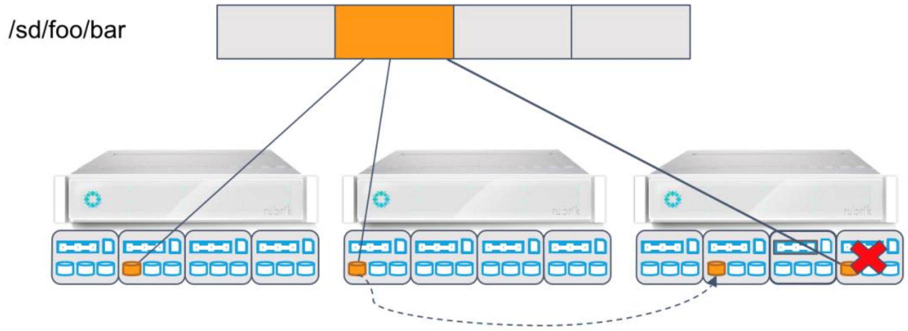
Advanced Operating Systems

Hamid Fadishei, Assistant Professor
Computer Engineerign Departemnt,
University of Bojnord
Fall 2018

Distributed File Systems

Distributed File System

- Distributed File System
  - A special case of distributed system
  - A file system that is distributed over multiple computers
- DFS benefits
  - Allows sharing files
  - Centralized administration
  - Uniform view



DFS Goals

- DFS is a special case of DS
- Goals of DFS are special case of DS goals
- DFS Goals
  - Network (Access) Transparency
    - *Users should be able to access files over a network as easily as if the files were stored locally*
  - Location Transparency
    - *File name doesn’t specify physical location*
  - Availability
    - *Files should be easily and quickly accessible*
    - *The number of users, system failures, or other consequences of distribution shouldn’t compromise the availability.*
  - Scalability

DFS Issues

- Issues that should be addressed to reach the goals
  - Communications (to address network transparency)
  - Naming (to address location transparency)
  - Consistency, replication and caching (to address availability and scalability)
  - Security
  - Fault tolerance (very much related to replication)

DFS Architectures

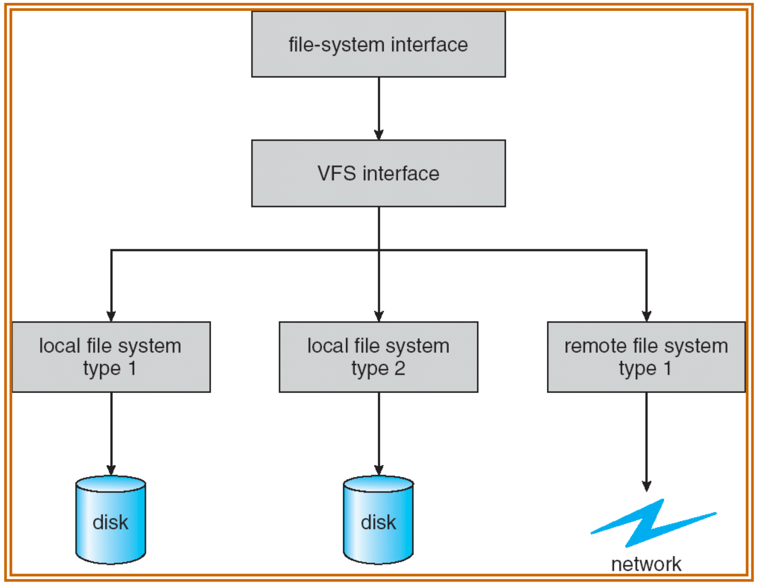
- DFS Architecture
  - Client-server
    - *Traditional: Example: NFS*
    - *Cluster-based: Example: GFS*
  - Symmetric
    - *Based on peer-to-peer technology*
    - *Example: Ivy*

Case Study: NFS

NFS stands for: Network File System
It is a client-server DFS

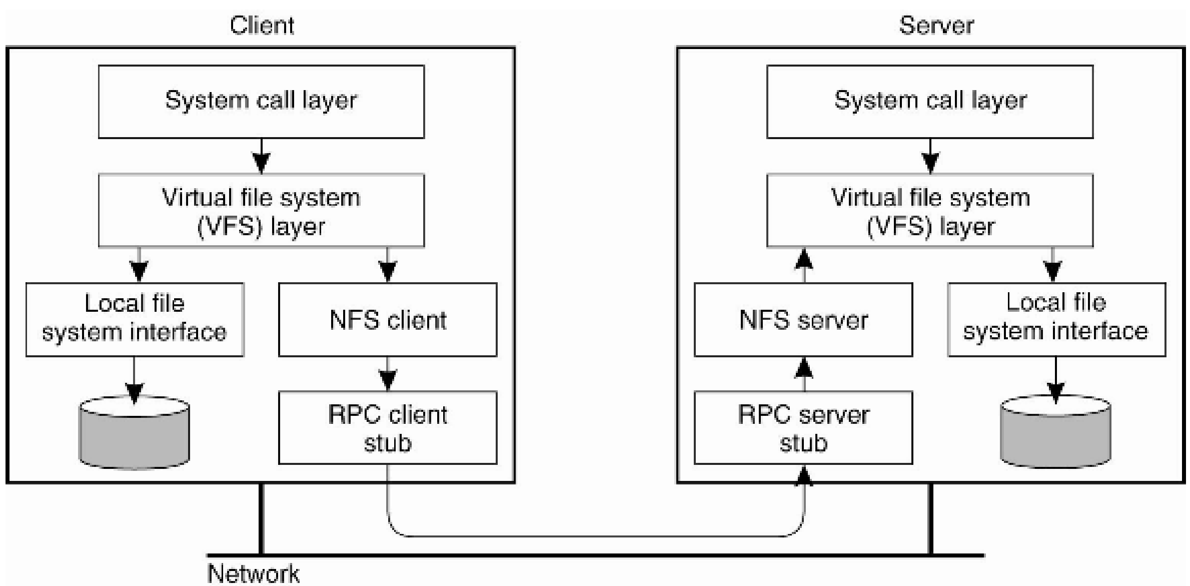
NFS

- A traditional client-server architecture
- Originally implemented in SUN Solaris
- Currently available in most Oses



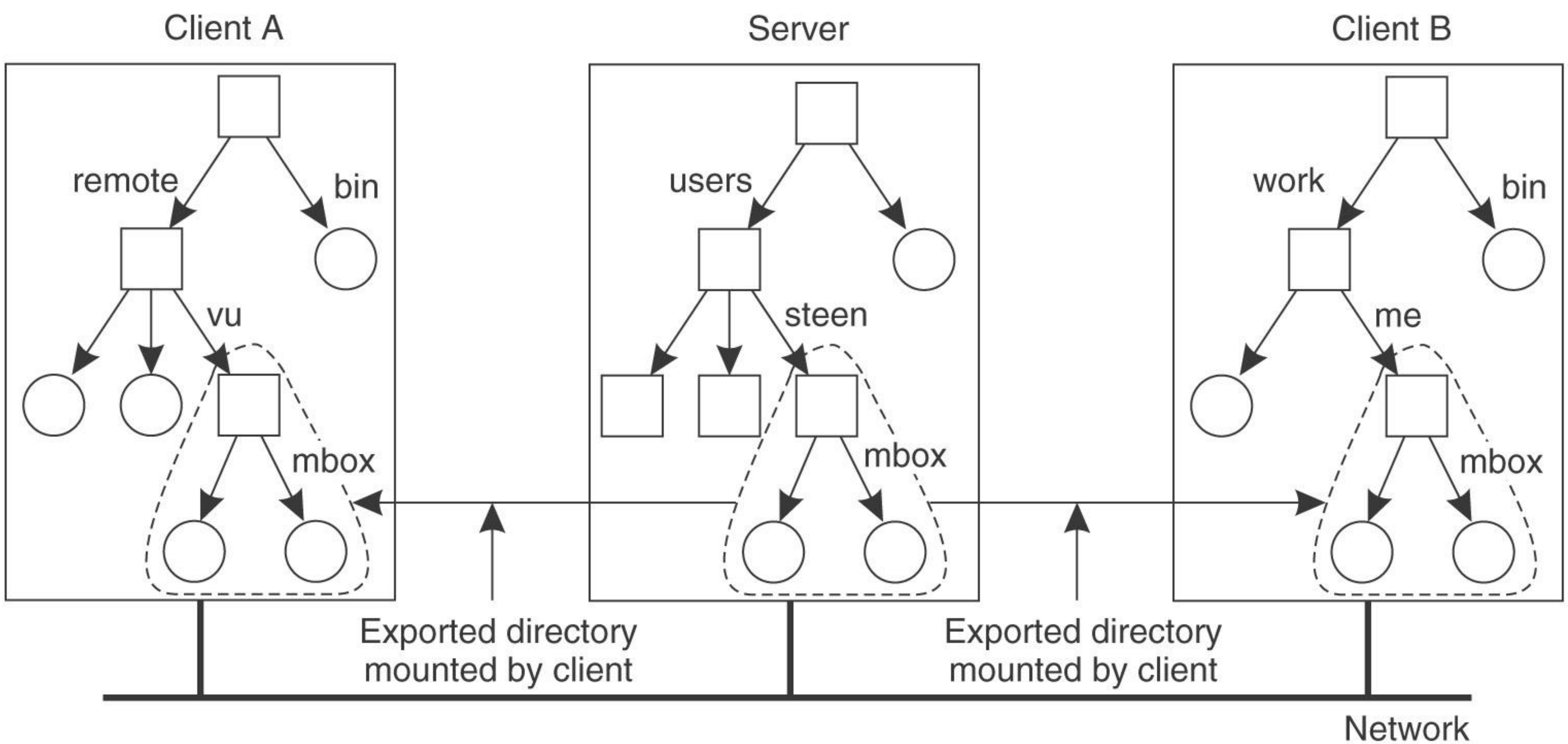
NFS Architecture

- NFS Architecture
  - Layered structure
  - Virtual File System (VFS) acts as an interface between the operating system's system call layer and all file systems on a node
  - The user interface to NFS is the same as the interface to local file systems. The calls go to the VFS layer, which passes them either to a local file system or to the NFS client



NFS Naming

- Addressed via mount points
  - Server A sees : /users/steen/mbox
  - Client A sees: /remote/vu/mbox
  - Client B sees: /work/me/mbox



Stateless vs Stateful

- Earlier NFS versions were stateless
  - No *state* information in server 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.
  - Poor support for locking or synchronization among concurrent accesses
- Stateful service
  - Client *opens* a file
  - Returns to client a *connection identifier* unique to client and open file
  - Identifier used for subsequent accesses until session ends

```
char buffer[MAX];
int fd = open("foo", O_RDONLY);
read(fd, buffer, MAX);
read(fd, buffer, MAX);
…
read(fd, buffer, MAX);
close(fd);
```

NFS API

Operation	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 in a given directory
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 from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Get the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

Case Study: GFS

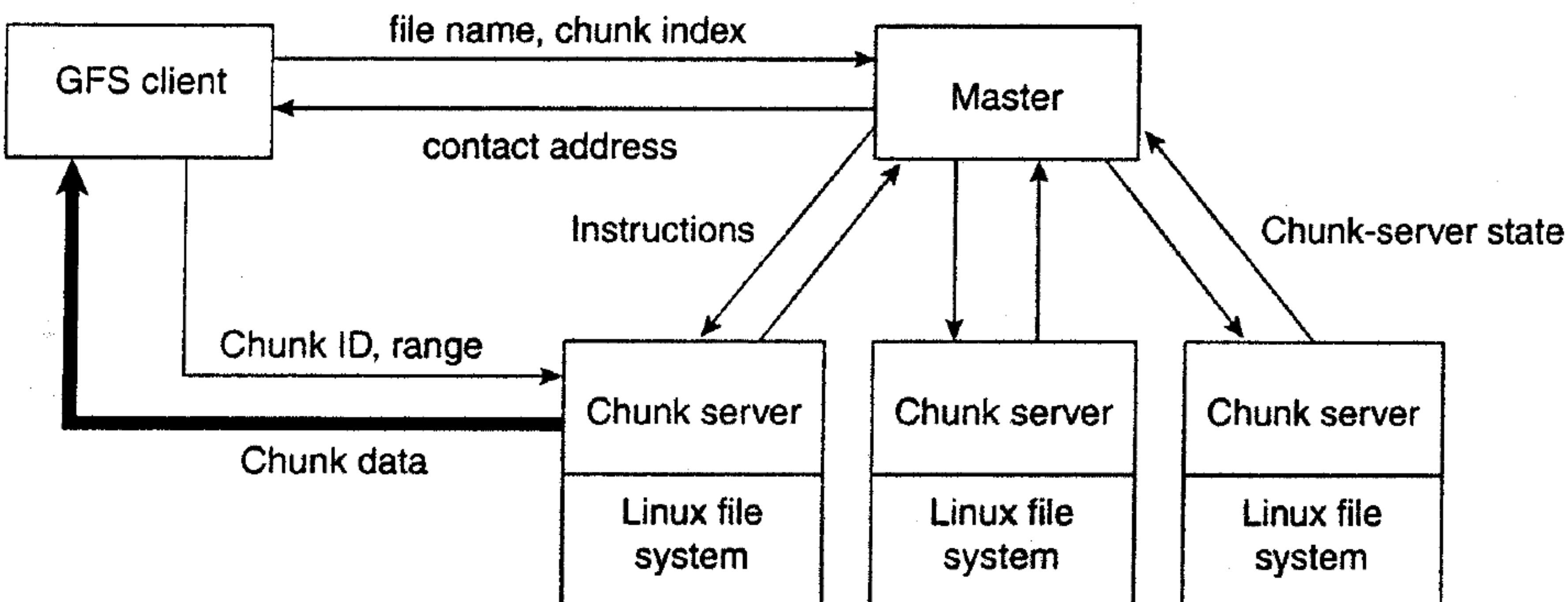
GFS stands for: the Google File System
It is a cluster-based DFS

Why cluster based DFS

- Downsides of a client-server architecture
  - Performance bottle neck
  - Single-Point-Failure

GFS Architecture

- Files are divided into 64MB chunks and placed on chunk servers
- Chunks are replicated
- Master knows (more or less) where (on which chunk server) to find the chunks of a file
- Master polls chunk server periodically to update its metadata



The philosophy behind GFS

- GFS is optimized for Google's core data storage
- They needed a DFS that can be made of commodity hardware
    - Many number of nodes which can be normal PCs (cheap and unreliable)
    - Nodes can fail at any time
    - Files are large and growing in size
    - Files are access by clients concurrently
- The files in their workload had specific characteristics
    - Very large
    - Growing in size, updated by appending
    - virtually never modified (other than by appends) nor deleted
    - Files are mostly read-only
    - Files are read mostly sequentially (streaming read, e.g. pattern mining)

Scalability of GFS

- Clients only contact the master to get metadata => no bottleneck
- Updates are performed by having a client update the nearest server which pushes the updates to one of the backups, which in turn sends it on to the next and so on.
- Master does not need to keep very up-to-date metadata
  - A Chunks server knows what exactly it stores
  - If client retrieval failed(low probability), ask Master again, master update latest info from chunk servers

Tasks and Exercises

Task 1

- All students, due next session
- Read the following paper:
  - Shvachko, Konstantin, et al. "**The Hadoop Distributed File System**." Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on. IEEE, 2010.
- And be prepared for a quiz!

Task 2

- Single student
  - Study the assigned paper
  - Present in class (due 2 weeks)
  - Find and study 3 related papers
  - Deliver a paper-formatted survey (due final exam)
- 1st student:
  - Wu, Suzhen, et al. “**PP: Popularity-Based Proactive Data Recovery for HDFS RAID systems**.” Future Generation Computer Systems 86 (2018): 1146-1153.
- 2nd student:
  - Ciritoglu, Hilmi Egemen, et al. "**Investigation of Replication Factor for Performance Enhancement in the Hadoop Distributed File System**." *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, 2018.
- 3rd student:
  - Ganesan, Aishwarya, et al. “**Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to Single Errors and Corruptions**.” FAST. 2017.