Storage

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 553: Cloud Computing
November 12th, 2025

Logistics

- Read chapter 2 & 3 from textbook

Storage: File Systems

- Interfaces – POSIX
  - Open, read, write, close, etc.
- Hierarchical directory structure
- Types
  - Local file systems (e.g. EXT3)
  - Network shared file system (e.g. NFS)
  - Parallel file system (e.g. Lustre)
  - Distributed file system (e.g. HDFS, Ceph)

Storage: Databases

- Key/Value Storage Systems
  - Hash tables
  - Persistent key/value systems
  - Distributed key/value storage systems (aka NoSQL)
- Relational Databases
  - Structured query language (SQL)
- Graph Databases
  - Example -- Neo4j

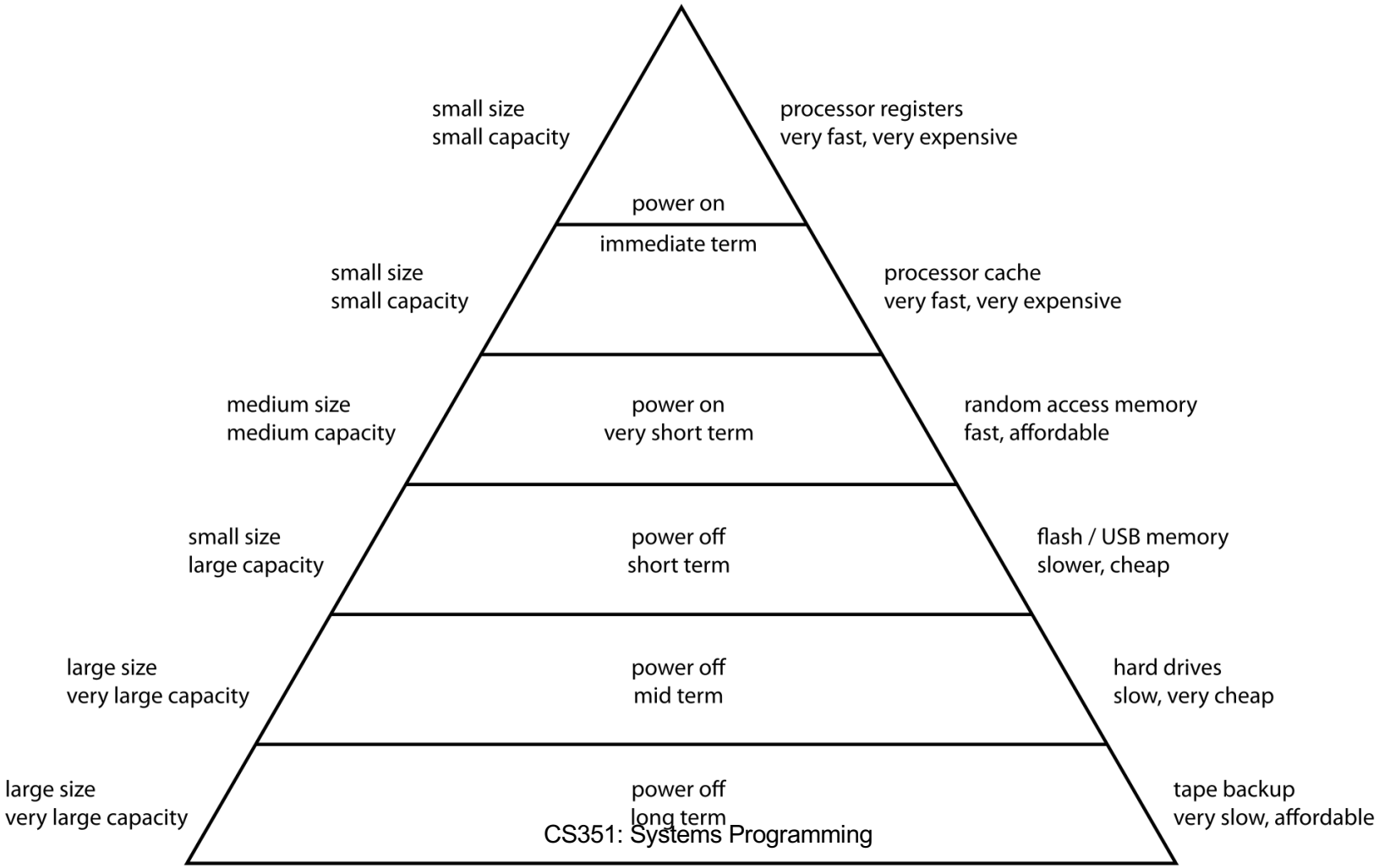
Storage: Cloud

- Amazon
  - Elastic Block Service: EBS
  - Simple Storage Service: S3
  - Glacier
  - DynamoDB
  - SQS
- Google
  - Cloud Storage
  - Nearline / Coldline
- Microsoft
  - Azure Storage / BLOB

Overview of Storage Technologies

Memory Hierarchy

Computer Memory Hierarchy



Disk Technologies

- Mechanical hard drives (HDD)
- Non-Volatile Memory (SSD)
  - SATA/SAS
  - NVMe
  - DDR4

Trends in DFS

- Wide area networking
- New hardware
  - Cheap main memory => file system in main memory with backups in videotape or optical disks
  - Extremely fast fiber optic networks => avoid client caching
- Scalability
  - From 100 to 1,000 to 10,000 nodes!
  - Use of broadcast messages should be reduced
  - Resources and algorithms should not be linear in the number of users

Trends in DFS

- Fault tolerance
  - As DSs become more widespread, provisions for higher availability have to be incorporated into the design
- Mobile users
  - Increase in disconnected operation mode
  - Files will be cached for longer periods (hours or days) at the client laptop
- Multimedia
  - New applications such as video-on-demand, audio files pose different demands on the design of a file system

Filesystems Overview

- System that permanently stores data
- Usually layered on top of a lower-level physical storage medium
- Divided into logical units called “files”
  - Addressable by a *filename* (“foo.txt”)
  - Usually supports hierarchical nesting (directories)
- A file *path* joins file & directory names into a **relative** or **absolute** address to identify a file (“/home/aaron/foo.txt”)

Shared/Parallel/Distributed Filesystems

- Support access to files on remote servers
- Must support concurrency
  - Make varying guarantees about locking, who "wins" with concurrent writes, etc...
  - Must gracefully handle dropped connections
- Can offer support for replication and local caching
- Different implementations sit in different places on complexity/feature scale

Timeline

- 1980~1990: NFS
- ~2000: PVFS
- ~2002: GPFS
- ~2003: Lustre
- ~2003: GFS
- ~2006: Sector
- ~2007: HDFS

NFS: Network File System

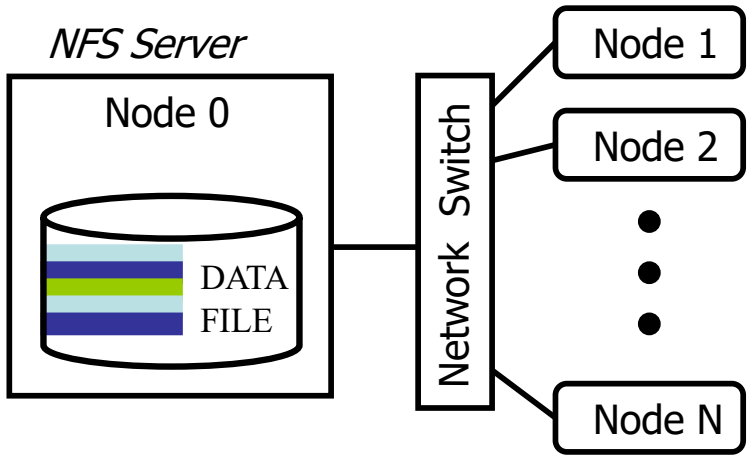
- First developed in 1980s by Sun
- Presented with standard UNIX FS interface
- Network drives are *mounted* into local directory hierarchy

NFS Protocol

- Initially completely stateless
  - Operated over UDP; did not use TCP streams
  - File locking, etc., implemented in higher-level protocols
- Modern implementations use TCP/IP & stateful protocols

NFS Architecture

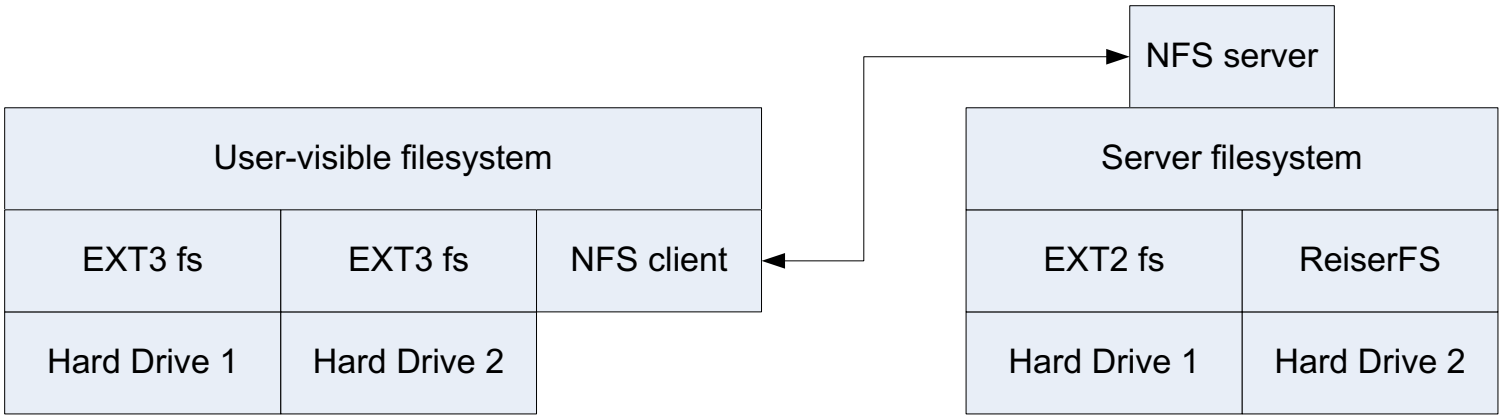
- Client/server system
- Single server for files



Each cluster node has dual-processor Pentium Linux, HD, lots of memory

NFS: Server-side Implementation

- NFS defines a *virtual file system*
  - Does not actually manage local disk layout on server
- Server instantiates NFS volume on top of local file system
  - Local hard drives managed by concrete file systems (EXT, ReiserFS, ...)
  - Other networked FS's mounted in by...?



NFS Locking

- NFS v4 supports stateful locking of files
  - Clients inform server of intent to lock
  - Server can notify clients of outstanding lock requests
  - Locking is lease-based: clients must continually renew locks before a timeout
  - Loss of contact with server abandons locks

NFS Client Caching

- NFS Clients are allowed to cache copies of remote files for subsequent accesses
- Supports *close-to-open* cache consistency
  - When client A closes a file, its contents are synchronized with the master, and timestamp is changed
  - When client B opens the file, it checks that local timestamp agrees with server timestamp. If not, it discards local copy.
  - Concurrent reader/writers must use flags to disable caching

NFS: Tradeoffs

- NFS Volume managed by single server
  - Higher load on central server
  - Simplifies coherency protocols
- Full POSIX system means it “drops in” very easily, but isn’t “great” for any specific need

PVFS Overview

- NFS not sufficient for high-performance computing workloads
- At the time, other solutions either non-existent, or did not run in Linux clusters
  - GPFS (proprietary on some IBM machines)
  - Lustre (not yet)
  - GFS (proprietary to Google)

PVFS Access

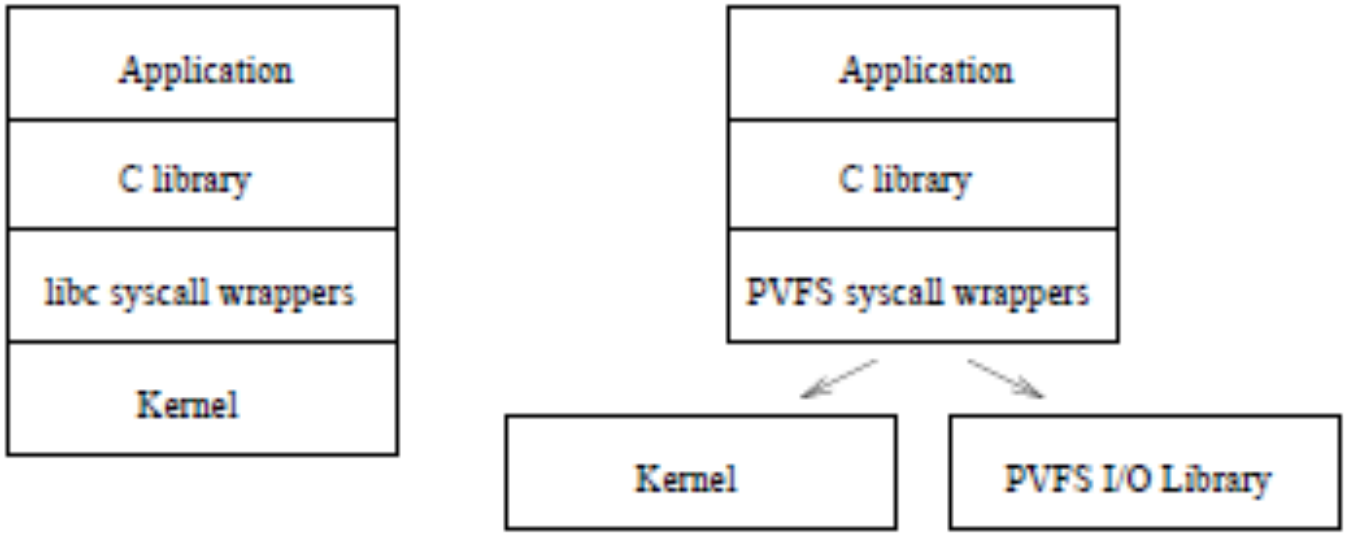
- Native PVFS Library
  - User space implementation
- Trapping I/O System calls
  - Allows applications to run without recompiling
  - Has limitations related to multi-process applications (e.g. exec causes file descriptor state to be lost)
  - Also requires high maintenance
- VFS Kernel Module
  - A module specific for PVFS, similar to NFS module

Native PVFS API example

```
#include <pvfs.h>

int main() {
int fd, bytes;
   fd=pvfs_open(fn,O_RDONLY,0,NULL,NULL);
   ...
   pvfs_lseek(fd, offset, SEEK_SET);
   ...
   bytes_read = pvfs_read(fd, buf_ptr, bytes);
   ...
   pvfs_close(fd);
}
```

Trapping System Calls



a) Standard operation

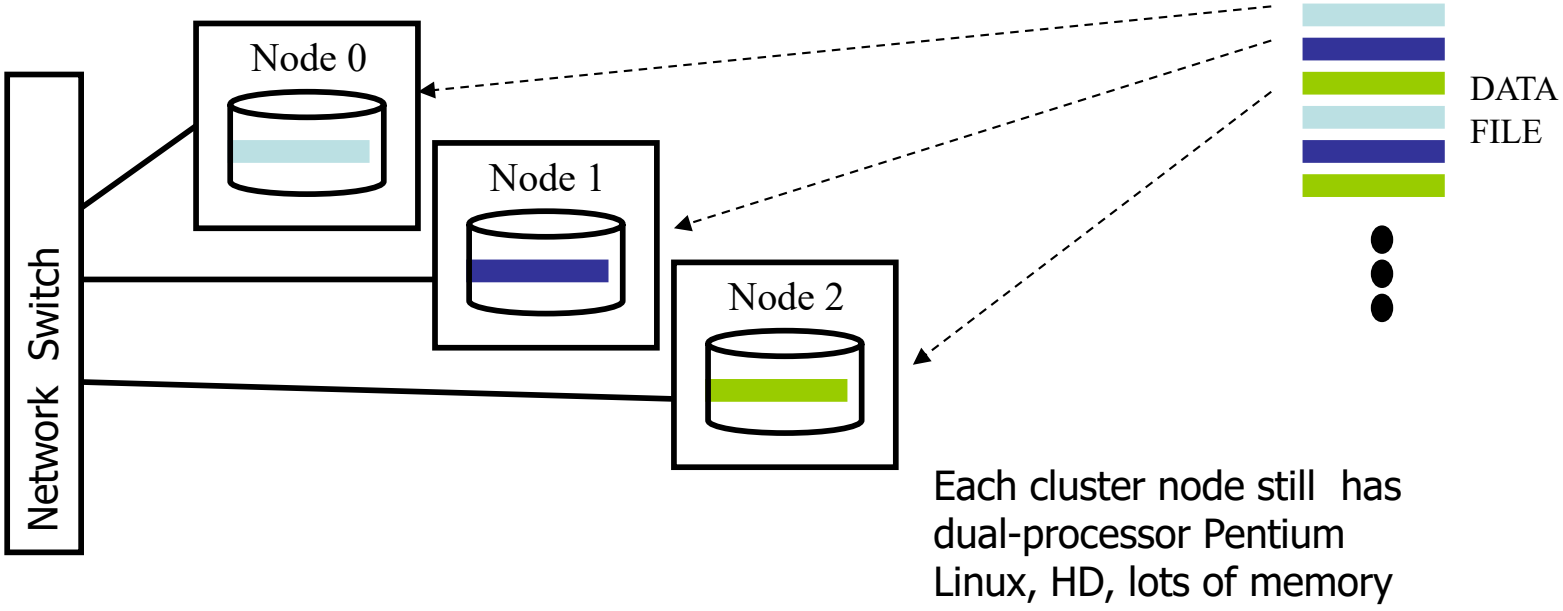
b) With PVFS library loaded

PVFS Architecture

- One node is a manager node
  - Maintains metadata information for files
- Configuration and usage options include:
  - Size of stripe
  - Number of I/O servers
  - Which nodes serve as I/O servers
  - Native PVFS API vs. UNIX/POSIX API

PVFS Architecture

- Also a client/server system
- Many servers for each file
- Fixed sized stripes in round-robin fashion



GPFS Overview

- GPFS had been used for years on IBM machines
- GPFS has been used on some of the largest supercomputers, including Linux-based ones
- GPFS aims for POSIX access semantic in a parallel file system
- All nodes have the same view
- Use distributed locking protocols

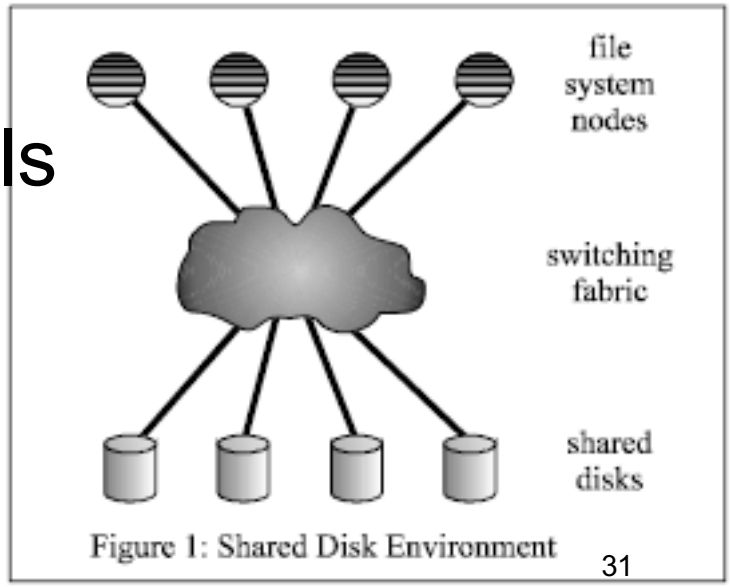


Figure 1: Shared Disk Environment

GPFS Details

- Parallel data and metadata access
- Data striping across disks
- General Large File System Issues
  - Data stripping and allocation, pre-fetch, and write-behind
  - Large directory support
  - Logging and recovery

GPFS Managing Consistency

- Locking management
  - Distributed locking
  - Centralized management
- GPFS distributed lock manager
- Parallel data access
  - Byte range locks
- Synchronizing access to file metadata
- Allocation maps
  - Managing free space
- Centralized token manager scaling

GPFS Fault Tolerance

- Node failures
  - Use recovery logs from shared disks
- Communication failures
  - Heartbeat messages
- Disk failures
  - RAID
  - Replication

Lustre Overview

- Also has a distributed lock manager
  - But more limited than that of GPFS
  - Intent locking
    - Switch between different strategies based on concurrency level
- Object-based vs. Block-based
  - Object-based protocols can help in locking and allocation of metadata
  - Lustre is backwards compatible with block-based storage
- Client caching metadata

Distributed File Systems: State of the Art

- GFS: Google File System
  - Google
  - C/C++
- HDFS: Hadoop Distributed File System
  - Yahoo
  - Java, Open Source
- Sector: Distributed Storage System
  - University of Illinois at Chicago
  - C++, Open Source

Filesystems Overview

- System that permanently stores data
- Usually layered on top of a lower-level physical storage medium
- Divided into logical units called “files”
  - Addressable by a *filename* (“foo.txt”)
  - Usually supports hierarchical nesting (directories)
- A file *path* joins file & directory names into a **relative** or **absolute** address to identify a file (“/home/aaron/foo.txt”)

Shared/Parallel/Distributed Filesystems

- Support access to files on remote servers
- Must support concurrency
  - Make varying guarantees about locking, who "wins" with concurrent writes, etc...
  - Must gracefully handle dropped connections
- Can offer support for replication and local caching
- Different implementations sit in different places on complexity/feature scale

Logistics

- HW5 on Hadoop/Spark is due tomorrow (Tuesday)
  - Extended 1 day, to Wednesday 11/26/25
- 5 assignments still worth 70% of grade
- Study guide review handout, will review Monday 12/1
- Final exam on Wednesday 12/3, same lecture location and time
  - 120 minutes
  - Closed book, closed note, individual
  - Will cover all material, lectures, reading, and programming assignments
  - 100 questions (multiple choice or true/false)
  - 30% of the final grade
- Topics left to cover: GFS, S3, EBS, and Spark

GFS: Motivation

- Google needed a good distributed file system
  - Redundant storage of massive amounts of data on cheap and unreliable computers

- Why not use an existing file system?
  - Google's problems are different from anyone else's
    - Different workload and design priorities
  - GFS is designed for Google apps and workloads
  - Google apps are designed for GFS

GFS: Assumptions

- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of HUGE files
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads
- High sustained throughput favored over low latency

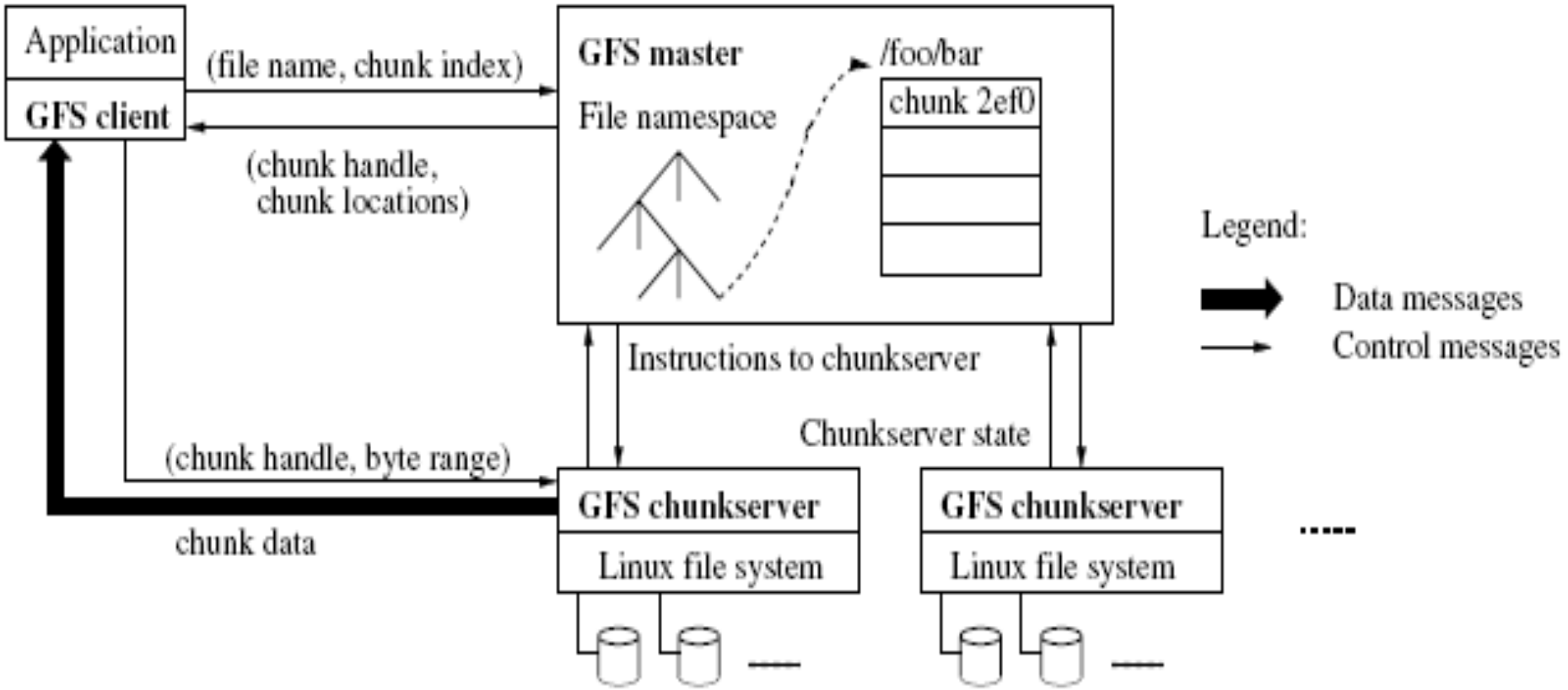
Google Workloads

- Most files are mutated by appending new data – large sequential writes
- Random writes are very uncommon
- Files are written once, then they are only read
- Reads are sequential
- Large streaming reads and small random reads
- High bandwidth is more important than low latency
- Google applications:
  - Data analysis programs that scan through data repositories
  - Data streaming applications
  - Archiving
  - Applications producing (intermediate) search results

GFS Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ *chunkservers*
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Familiar interface, but customize the API
  - Simplify the problem; focus on Google apps

GFS Architecture



GFS Architecture

- Single master
- Multiple chunk servers
- Multiple clients
- Each is a commodity Linux machine, a server is a user-level process
- Files are divided into chunks
- Each chunk has a handle (an ID assigned by the master)
- Each chunk is replicated (on three machines by default)

GFS Architecture

- Master stores metadata, manages chunks, does garbage collection, etc.
- Clients communicate with master for metadata operations, but with chunkservers for data operations
- No additional caching (besides the Linux in-memory buffer caching)

GFS Discussion

- Client/GFS Interaction
- Master
- Metadata
- Why keep metadata in memory?
- Why not keep chunk locations persistent?
- Operation log
- Data consistency
- Garbage collection
- Load balancing
- Fault tolerance

Amazon Web Services (AWS)

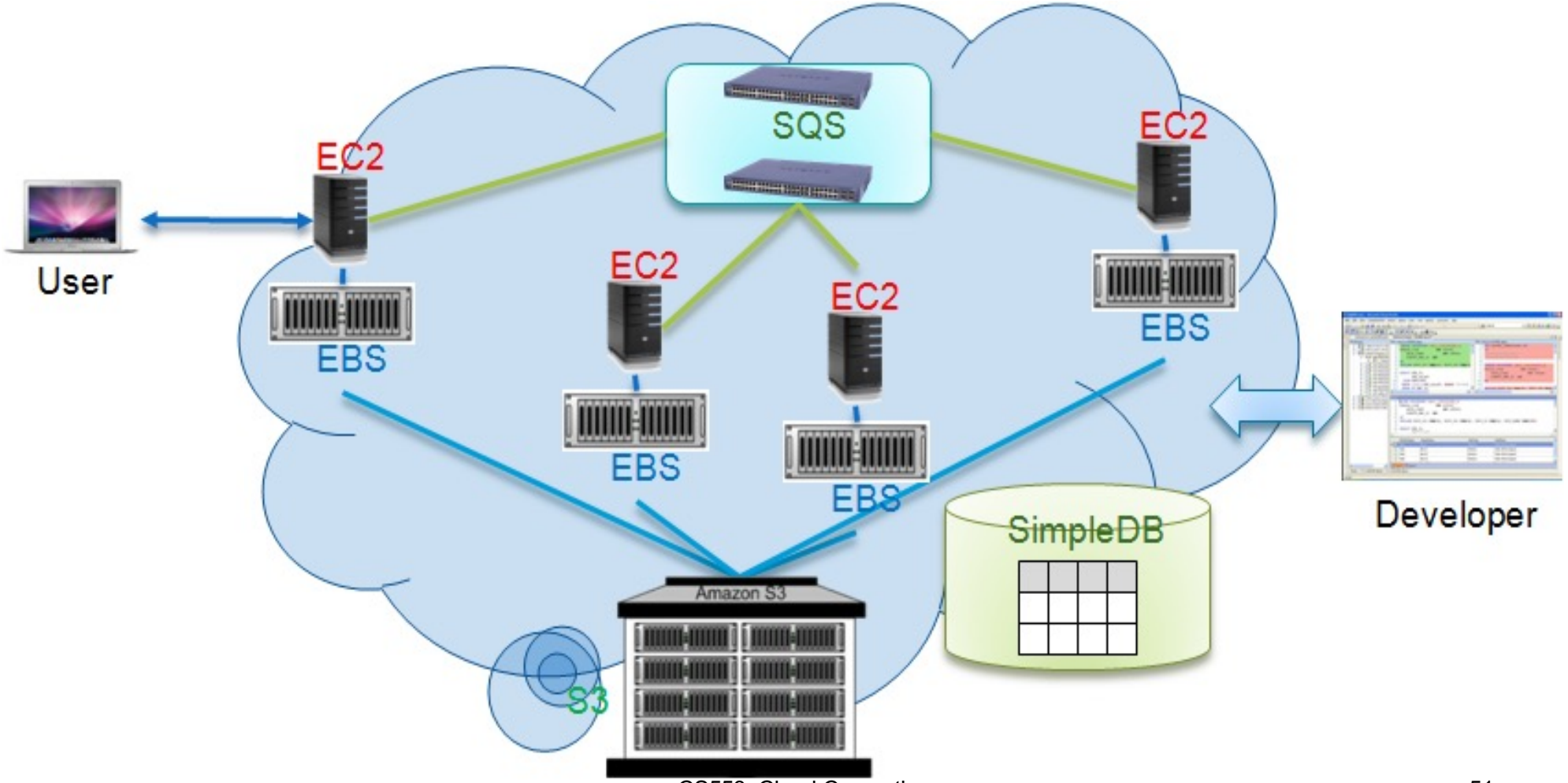
- S3
- SimpleDB
  - NOSQL support
- Relational Database Service (RDS)
- Simple Queue Service (SQS)
- Simple Notification Service (SNS)
- More services to discuss later
  - Elastic MapReduce
    - Capability is equivalent to Hadoop running on the basic EC2 offering.
  - EC2

Amazon Web Services (AWS)

- Amazon has been a leader in providing public cloud services
- Amazon applies the IaaS model in providing its services
- Different from Google, Amazon provides a more flexible cloud computing platform for developers to build cloud applications.
- Elastic Computing Cloud (EC2) provides the virtualized platforms to the host VMs where the cloud application can run
- S3 (Simple Storage Service) provides the object-oriented storage service for users
- EBS (Elastic Block Service) provides the block storage interface which can be used to support traditional applications
- SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between two processes
  - The message can be kept reliably even when the receiver processes are not running
  - Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard.

Amazon Web Services (AWS)

- **Amazon cloud computing infrastructure**



Amazon S3

- Amazon S3 provides a simple Web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the Web
- S3 provides the object-oriented storage service for users
- Users can access their objects through *Simple Object Access Protocol* (SOAP)
  - Supports both browsers and client programs which support SOAP
- SQS is responsible for ensuring a reliable message service between various processes, even if the receiver processes are not running

Amazon S3

- Fundamental operation unit of S3 ➔ *object*
- Each object is stored in a *bucket* and retrieved via a unique, developer-assigned key
  - A bucket is the container of the object
  - Objects have other attributes such as values, metadata, and access control information
- S3 is essentially a key-value pair
  - Users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each
- Two types of Web service interfaces to access data stored in Amazon clouds
  - REST (Web 2.0)
  - SOAP

Amazon S3

- Key Features of S3:
  - Redundant through geographic dispersion
  - Designed to provide 99.999999999 percent durability and 99.99 percent availability of objects over a given year with cheaper reduced redundancy storage (RRS)
  - Authentication mechanisms to ensure that data is kept secure from unauthorized access
    - Objects can be made private or public, and rights can be granted to specific users
  - Per-object URLs and ACLs (access control lists)
  - Default download protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution
  - First 5GB are free, and $0.023 per GB/month (first 50TB); down to $0.00099 per GB/month for infrequent access
  - Data in is free; data out costs, with first 1GB per month free and then $.09 per GB for transfers outside an S3 region

Amazon Elastic Block Store (EBS)

- Traditional EC2 instances will be destroyed after use
- Note that S3 is "Storage as a Service" with a messaging interface
- The *Elastic Block Store* (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances onto S3
  - The status of EC2 can now be saved in the EBS system after the machine is shut down
  - Users can use EBS to save persistent data and mount to the running instances of EC2
  - EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms
  - EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as EC2 instances

Amazon Elastic Block Store (EBS)

- Multiple volumes can be mounted to the same instance
- These storage volumes behave like raw, unformatted block devices, with user-supplied device names and a block device interface
- You can create a file system on top of Amazon EBS volumes, or use them in any other way you would use a block device (like a hard drive)
- Snapshots are provided so that the data can be saved incrementally
  - This can improve performance when saving and restoring data
- Pricing ➔ similar pay-per-use schema as EC2 and S3
  - Volume storage charges are based on the amount of storage users allocate until it is released, and is priced at $0.025 to $0.10 per GB/month depending on the type of storage (SSD, HDD)
  - AWS Free Tier includes 30GB of Storage, 2 million I/Os, and 1GB of snapshot storage with Amazon Elastic Block Store (EBS)

Amazon SimpleDB Service

- SimpleDB provides a simplified data model based on the relational database data model
  - Structured data from users must be organized into domains
  - Each domain can be considered a table
  - The items are the rows in the table
  - A cell in the table is recognized as the value for a specific attribute (column name) of the corresponding row
- SimpleDB is similar to a table in a relational database
  - However, it is possible to assign multiple values to a single cell in the table
  - This is not permitted in a traditional relational database which wants to maintain data consistency

Amazon SimpleDB Service

- Many developers simply want to quickly store, access, and query the stored data
- SimpleDB removes the requirement to maintain database schemas with strong consistency
- SimpleDB is priced at $0.140 per Amazon SimpleDB Machine Hour consumed
  - First 25 Amazon SimpleDB Machine Hours consumed per month free (as of October 6, 2010).
- SimpleDB, like Azure Table, could be called “LittleTable,” as they are aimed at managing small amounts of information stored in a distributed table
- BigTable (from Google) is aimed at basic big data, whereas LittleTable is aimed at metadata
- Amazon Dynamo is an early research system along the lines of the production SimpleDB system

More on Storage/Data Management

- FUSE
- FTP
- GridFTP
- Globus Online
- Best way to transfer lots of data long distance?

Questions