The University for world-class professionals

Manchester Metropolitan University

Advanced Operating Systems

Lecture: Distributed File Systems

Soufiene Djahel
Office: John Dalton E114
Email: s.djahel@mmu.ac.uk
Telephone: 0161 247 1522
Office hours: Monday 13 -15, Tuesday 11-12

This lecture was adapted and extended from the slides originally designed by Emma Norling, MMU, UK

Recap from last week

- Cloud computing essential characteristics
- Types of cloud deployment: public, private and hybrid
- Cloud service models: Iaas, Paas and Saas
- Cloud OS
- Fog and Mist Computing

Today's Objectives

- To extend the knowledge of basic file systems to distributed file systems.
- To study examples of distributed file system implementations
- To compare the needs of cloud computing and Big Data with standard file system needs

What is a file system?

- File systems determine how data is stored and retrieved
- Must support several features
  - Naming: relate file names to file IDs
  - Storage management: relate file IDs to storage location
  - Access control, etc.
- Goal: allow users to associate names with chunks of data
- Pretty straightforward on a single computer…

Why distributed file systems?

- Users want to access files on multiple systems
  - Network of computers
  - Access from anywhere in the world
- Users want to share their data
  - Use on multiple computers
  - One copy of a file used by several people
- Users want better performance
  - Single file server can be a **bottleneck**
  - Distribute the storage across servers

Distributed file systems

- Distributed file systems manage the storage across a network of machines
  - Added complexity due to the network
- What are they good for?
  - Sharing information with others
  - Accessing information remotely
  - Remote backup
- Why are they difficult?
  - Consistency
  - Transparency
  - Replication

Distributed File System Requirements:

Transparency

- Access transparency:
  - clients unaware of the distribution of files
  - same interface/methods for local/remote files
- Location transparency:
  - uniform file name space from any client workstation
- Mobility transparency:
  - files can be moved from one server to another without affecting the client

Distributed File System Requirements: Transparency (cont,)

- Performance transparency:
  - client performance not affected by load on service
- Scaling transparency:
  - expansion possible if numbers of clients increase

More Distributed File System Requirements

- Concurrent file updates
  - Files can be modified by multiple clients at the same time
    - Requires good concurrency control
- File replication
  - for load sharing, fault-tolerance
- Heterogeneity
  - interface platform-independent
- Fault-tolerance
  - continue to operate in case of client and server failures

More Distributed File System Requirements (cont.)

- Consistency
  - one-copy-update semantics or slight variations
- Security
  - access control
- Efficiency
  - performance comparable to conventional file systems

Distributed File Systems

- File service: specification of what the file system offers
  - Client primitives, API, etc
- File server: process that implements file service
  - Can have several servers on one machine

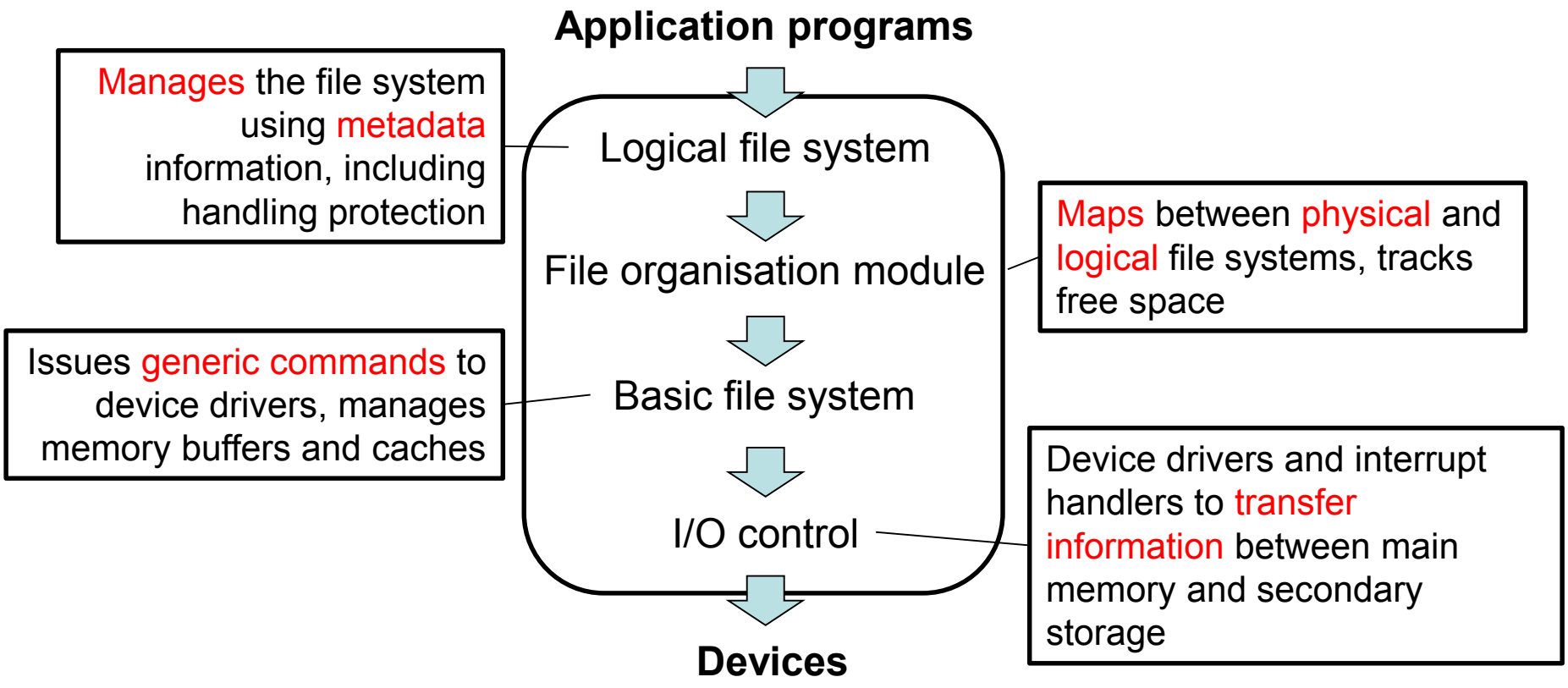
System Structure: Server Type

- Stateless server
  - No information is kept at server between client requests
  - All information needed to service a requests must be provided by the client with each request
    - e.g., complete file names, location for reading, etc
  - More tolerant to server crashes

System Structure: Server Type

- Stateful server
  - Server maintains information about client accesses
  - Shorted request messages
  - Better performance
  - Consistency is easier to achieve

Layered File System



Examples of Distributed file systems

- Sun Network File System (NFS)
- Andrew File System (AFS)
- Google File System (GFS)
- Hadoop Distributed File System (HDFS)

Sun Network File System (NFS)

- Sun's Network File System (NFS) – widely used distributed file system
  - Part of the Solaris and SunOS operating systems running on Sun workstations
  - builds on the Open Network Computing Remote Procedure Call (ONC RPC) system
- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)

Sun Network File System (NFS)

- 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 is mounted over a local file system directory
  - Specification of the remote directory for the mount operation is non-transparent;
    - the host name of the remote directory has to be provided
    - Files in the remote directory can then be accessed in a transparent manner

NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
  - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
  - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them

NFS Mount Protocol

- Following a mount request that conforms to its export list, the server returns a file handle — a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system

NFS Protocol

- Provides a set of remote procedure calls (RPC) for remote file operations. The procedures support the following operations:
  - searching for a file within a directory
  - reading a set of directory entries
  - manipulating links and directories
  - accessing file attributes
  - reading and writing files

NFS Protocol

- NFS servers are stateless; each request has to provide a full set of arguments (NFS V4 – RFC 7530 published on March 2015 – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

Andrew File System (AFS)

- Started as a joint effort of Carnegie Mellon University and IBM
- Today basis for DCE/DFS: the distributed file system included in the Open Software Foundations' Distributed Computing Environment

AFS (cont.)

- Made open-source by IBM under the name OpenAFS in 2000, and continues to be in widespread use at Carnegie Mellon and many other institutions.
- Callbacks: Server records who has copy of file
  - On changes, server immediately tells all with old copy
  - No polling bandwidth (continuous checking) needed

AFS (cont.)

- Write through on close
  - Changes not propagated to server until close()
  - Session semantics: updates visible to other clients only after the file is closed
    - As a result, do not get partial writes: all or nothing!
    - Although, for processes on local machine, updates visible immediately to other programs who have file open
- In AFS, everyone who has file open sees old version
  - Don't get newer versions until reopen file

AFS (cont.)

- Data cached on local disk of client as well as memory
  - On open with a cache miss (file not on local disk):
    - Get file from server, set up callback with server
  - On write followed by close:
    - Send copy to server; server informs clients
- What if server crashes? Lose all callback state!
  - Poll clients

AFS (cont.)

- AFS Pro: Relative to NFS, less server load:
  - Disk as cache ⇒ more files can be cached locally
  - Callbacks ⇒ server not involved if file is read-only
- For both AFS and NFS: central server is bottleneck!
  - Performance: all writes→server, cache misses→server
  - Availability: Server is single point of failure
  - Cost: server machine's high cost relative to workstation

Windows NTFS

- The developers of Windows NT designed a new file system, the New Technology File System (NTFS) which is intended to meet high-end requirements for workstations and servers
- Key features of NTFS:
  - recoverability
  - security
  - large disks and large files
  - multiple data streams
  - journaling
  - compression and encryption
  - hard and symbolic links

NTFS Volume and File Structure

- Sector
  - the smallest physical storage unit on the disk
  - the data size in bytes is a power of 2 and is almost always 512 bytes
- Cluster
  - one or more contiguous sectors
  - the cluster size in sectors is a power of 2
- Volume
  - a logical partition on a disk, consisting of one or more clusters and used by a file system to allocate space
  - can be all or a portion of a single disk or it can extend across multiple disks
  - the maximum volume size for NTFS is 2^{64} clusters

10 Minute Break

Managing Big Data

- Previous examples assumed "normal" file usage
- "Big Data" brings a whole new set of considerations
  - Performance
  - Scalability
  - Reliability
  - Availability

Google File System (GFS)

- Developed by engineers at Google around 2003
  - Built on principles in parallel and distributed Processing
- Important Google papers
  - The Google file system by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (2003)
  - MapReduce: Simplified Data Processing on Large Clusters. By Jeffrey Dean and Sanjay Ghemawat (2004)

Google File System (GFS)

- Google went the cheap commodity route…
  - Lots of data on cheap machines
- Why not use an existing file system?
  - Unique problems
  - GFS is designed for Google apps and workloads
  - Google apps are designed for GFS

GFS Assumptions

- Failures are the norm
  - Detect errors, recover, tolerate faults, etc.
  - Software errors (we're only human)
- Huge files
  - Multi-GB files are common
  - But there aren't THAT many files

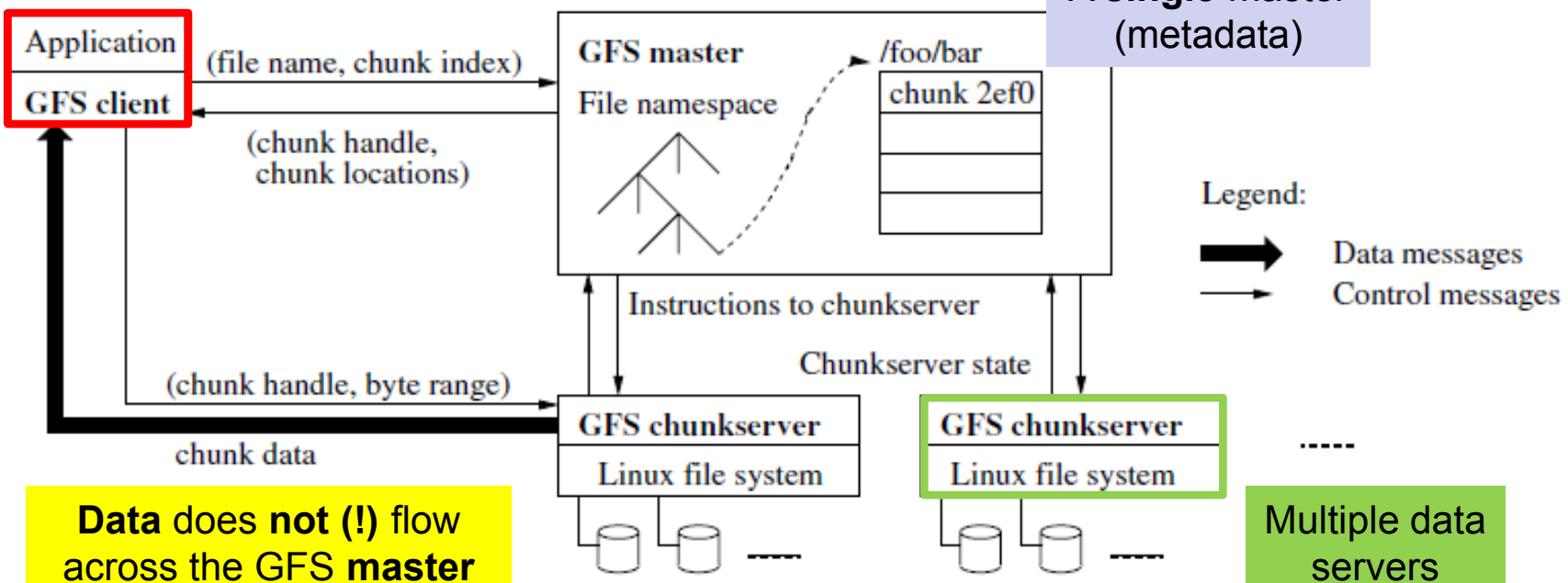
GFS Interface

- Supports usual commands
  - Create, delete, open, close, read, write
- Snapshot
  - Copies a file or a directory tree
- Record Append
  - Allows multiple concurrent appends to same file

GFS Architecture

Multiple clients

A **single** master
(metadata)



Data does **not (!)** flow
across the GFS **master**

Multiple data
servers

GFS Architecture

- Files divided into fixed-sized chunks (64 MB)
  - Each chunk gets a chunk handle from master
    - unique 64 bit identifiers
  - Stored as linux files
- One Master
  - Maintains all files system metadata
  - Talks to each chunksever periodically

GFS Architecture (cont.)

- Multiple Chunkservers
  - Store chunks on local disks
  - No caching of chunks. (Why not?)
- Multiple Clients
  - Clients talk to master for metadata operations
    - clients do cache metadata for a limited time
  - Read / write data from chunkservers
    - Thus minimizing the master involvement

Single Master

- From distributed systems we know this is a:
  - Single point of failure
  - Scalability bottleneck
- GFS solutions:
  - Shadow masters
  - Minimize master involvement
    - never move data through it, use only for metadata
      - and cache metadata at clients
    - large chunk size (64 MB)
- Simple, and good enough!

Files on GFS

- A single file can contain many objects (e.g. Web documents)
- Files are divided into fixed size **chunks** (64MB) with unique 64 bit identifiers
  - IDs assigned by GFS master at chunk creation time
- **chunkservers** store chunks on local disk as "normal" Linux files
  - Reading & writing of data specified by the tuple

    (chunk handle, byte range)

Files on GFS (cont.)

- Files are replicated (by default 3 times) across all chunk servers
- **Master** maintains all file system **metadata**
  - Namespace, access control information, mapping from files to chunks, locations of chunks, chunk migration, …

Files on GFS (cont.)

- **Heartbeat** messages between master and chunkservers
  - Is the chunkserver still alive? What chunks are stored at the chunkserver?
- To read/write data: client code communicates with master (metadata operations) **and** chunk servers (data)

Want to know more?

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In *ACM SIGOPS Operating Systems Review* (Vol. 37, No. 5, pp. 29-43). ACM.

Hadoop Distributed File System (HDFS)

Designed to be robust to a number of the problems that other DFS's such as NFS are vulnerable to. In particular:

- HDFS is designed to store a very large amount of information (terabytes or petabytes).
- HDFS should store data reliably
- HDFS should provide fast, scalable access to this information
- HDFS should integrate well with Hadoop MapReduce, allowing data to be read and computed upon locally when possible.

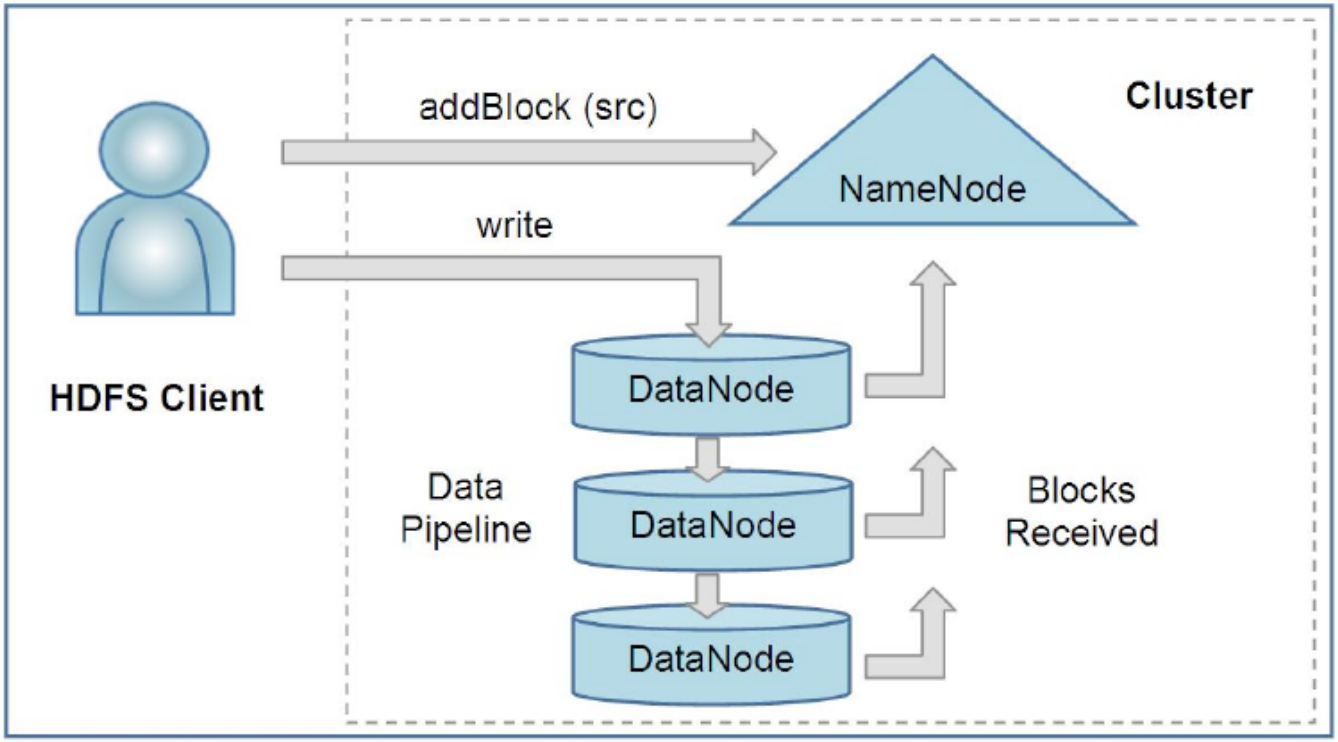
GFS vs. HDFS

GFS	HDFS
Master	NameNode
chunkserver	DataNode
operation log	journal, edit log
chunk	block
random file writes possible	**only append is possible**
multiple writer, multiple reader model	single writer, multiple reader model
chunk: 64KB data and 32bit checksum pieces	per HDFS block, two files created on a DataNode: data file & metadata file (checksums, timestamp)
periodic data rebalancing by the master	rebalancing initiated by cluster admin, “Balancer” application
default block size: **64MB**	default block size: **128MB**

HDFS architecture (0.X, 1.X)

- NameNode
  - Master of HDFS, directs the slave DataNode daemons to perform low-level I/O tasks
  - Keeps track of file splitting into blocks, replication, which nodes store which blocks, etc.
- Secondary NameNode: takes snapshots of the NameNode
- DataNode
  - Each slave machine hosts a DataNode daemon

HDFS architecture (0.X, 1.X)



Hadoop in Practice: Yahoo! (2010)

- **40 nodes/rack** sharing one IP switch
- **16GB RAM** per cluster node, 1-gigabit Ethernet
- **70%** of disk space allocated to HDFS
  - remainder: operating system, data emitted by Mappers (not in HDFS)
- NameNode: up to **64GB RAM**
  - application tasks are never run here
- Total storage: **9.8PB** -> 3.3PB net storage (**replication: 3**)
- **60 million** files, 63 million blocks
- 54,000 blocks hosted per DataNode
- 1-2 nodes lost per day
  - Time for cluster to re-replicate lost blocks: 2 minutes

Want to know more?

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (pp. 1-10). IEEE.

What About Google Drive, Dropbox, and so on?

- What you see on your desktop is a client that you install
- Provides an interface to the distributed file system
  - the type of the file system depends on the provider
- The client is similar to the user-side of the Linux VFS
  - but not part of the operating system

Summary

- Historic approaches to distributed file systems have focused on providing a seamless user interface
  - NFS, AFS, NTFS
- More recently, a demand for distributed file systems with different characteristics
  - particularly for dealing with Big Data
  - GFS, Hadoop DFS

References

- NFS V4: RFC7530
- NFS V4.2: RFC 7862
- Credits for material to:
  - Gayle Laakmann, University of Washington
  - Behzad Bordbar, University of Birmingham
  - Claudia Hauff, TU Delft
  - Apache Hadoop tutorial

Next lecture: Virtual machines