THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

Cloud Computing (INFS3208)

Lecture 10: Distributed File System

Lecturer: Dr Sen Wang

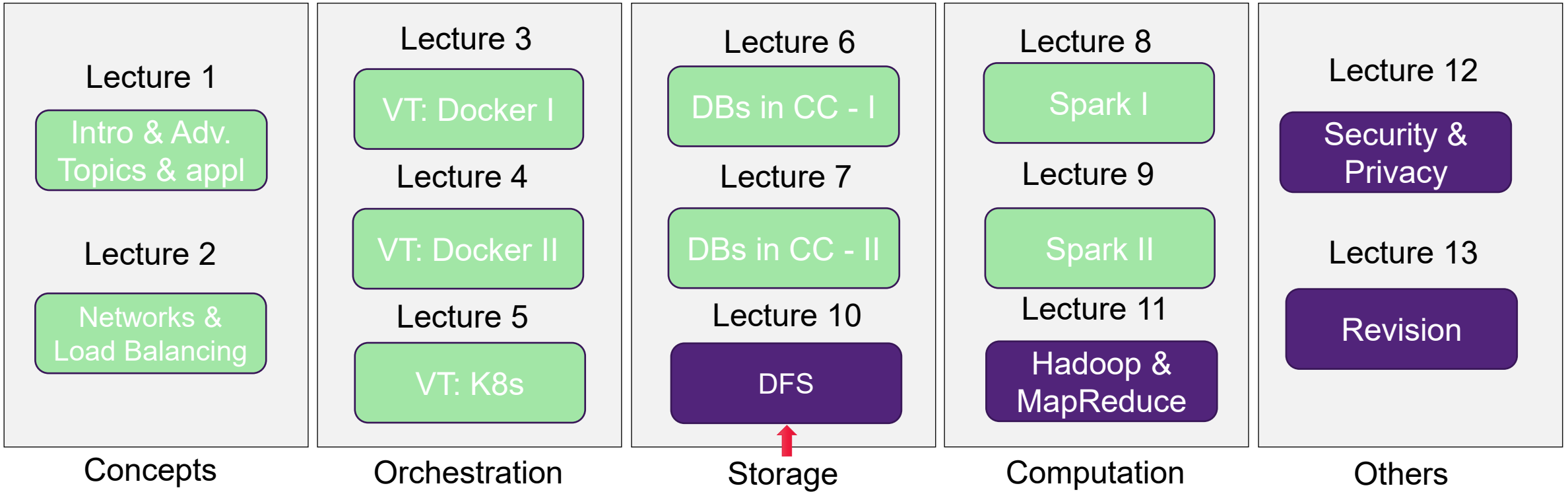
School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

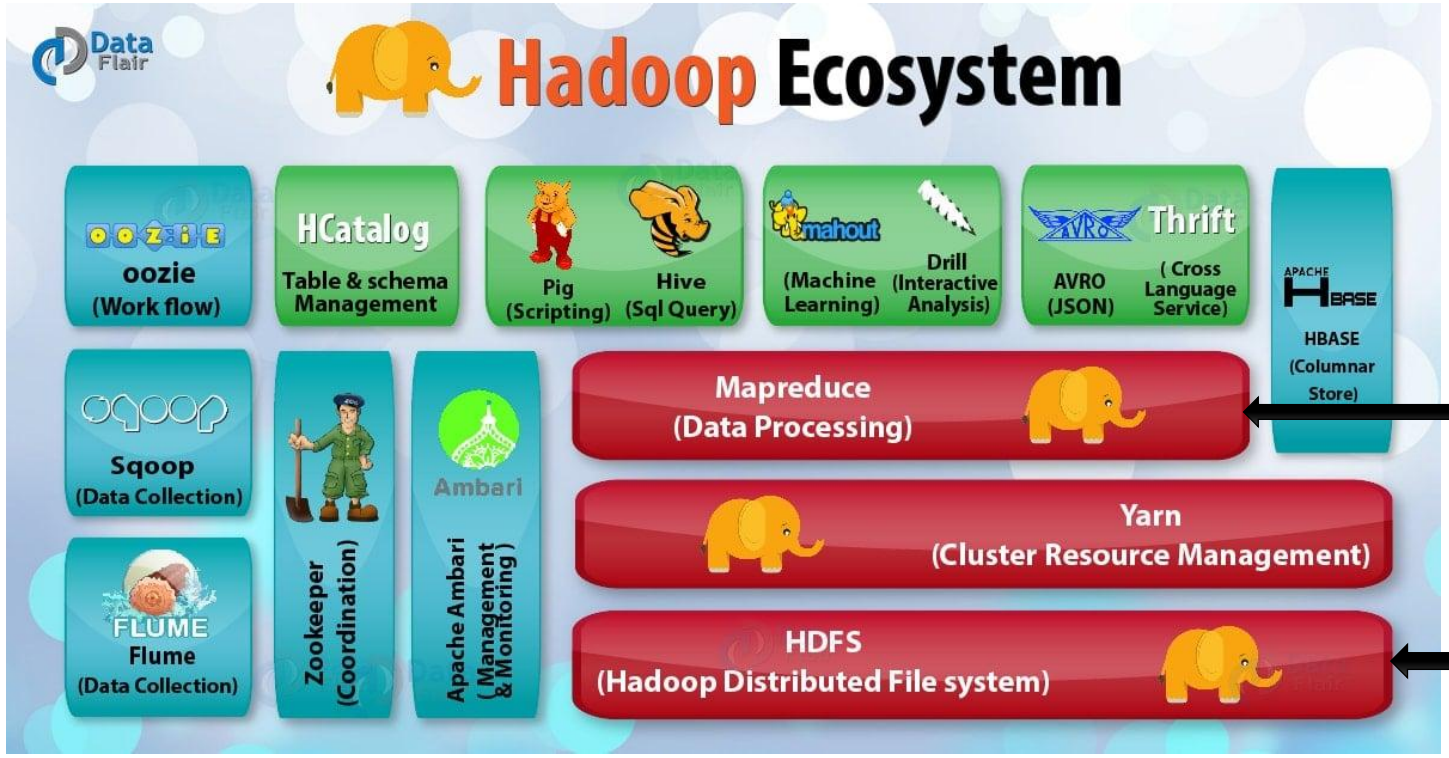
Progress – 9/12 (75% completed)

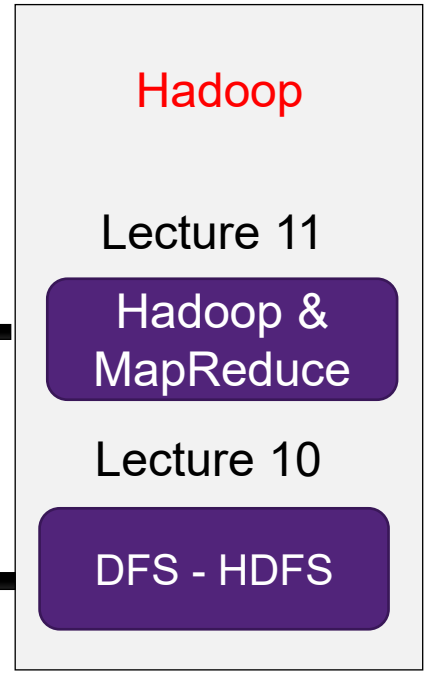
This course includes 13 lectures and 10 tutorial/practical sessions



Assessments: PT1&2 Completed, PT3 due on 17/10; Individual Project: 24/10, marking & competition.

Progress – 9/12 (75% completed)





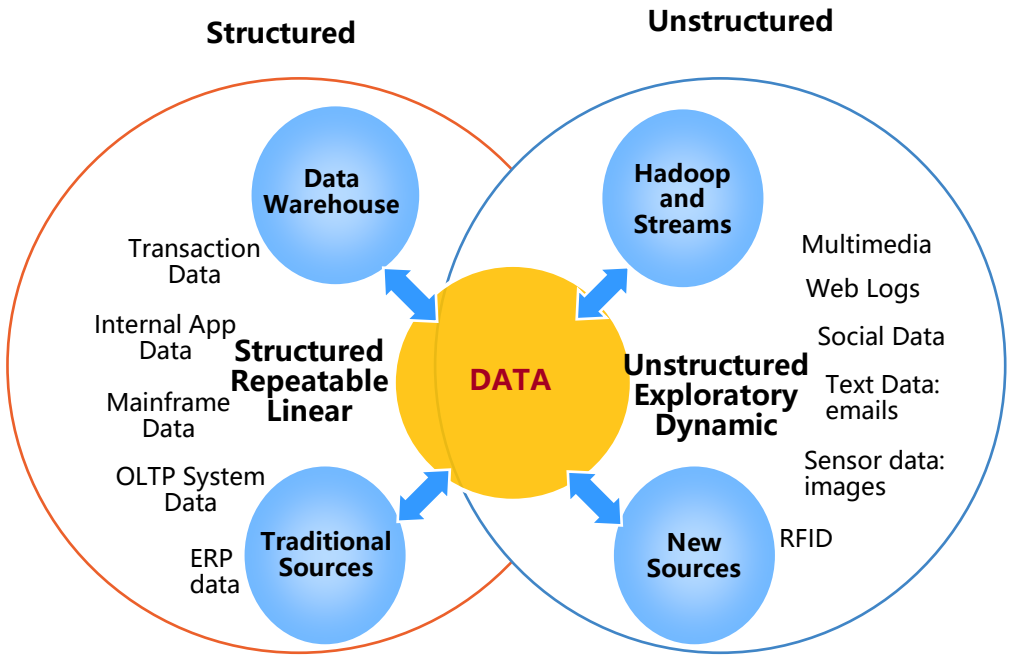
Industrial Open-Source
Big Data Analytic Solution

https://data-flair.training/blogs/hadoop-ecosystem-components/

Outline

- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

Issues of huge volume of data generated daily!!



WeChat:

- An estimated **85% of users actively engage with Moments daily**.
- There are **1.1 billion** users making online payments with the use of WeChat Pay.
- Roughly **614 million** WeChat users enter the WeChat mini-programs sub-platform at least once a day.
- WeChat's video communication features are now utilized daily and there are **205 million** video messages sent every day.

Facebook:

- Facebook has **2.93 billion** monthly active users.
- **36.7% of the world's population** uses FB monthly
- **~15%** of FB Feed content is recommended by AI from non-followed accounts

TikTok:

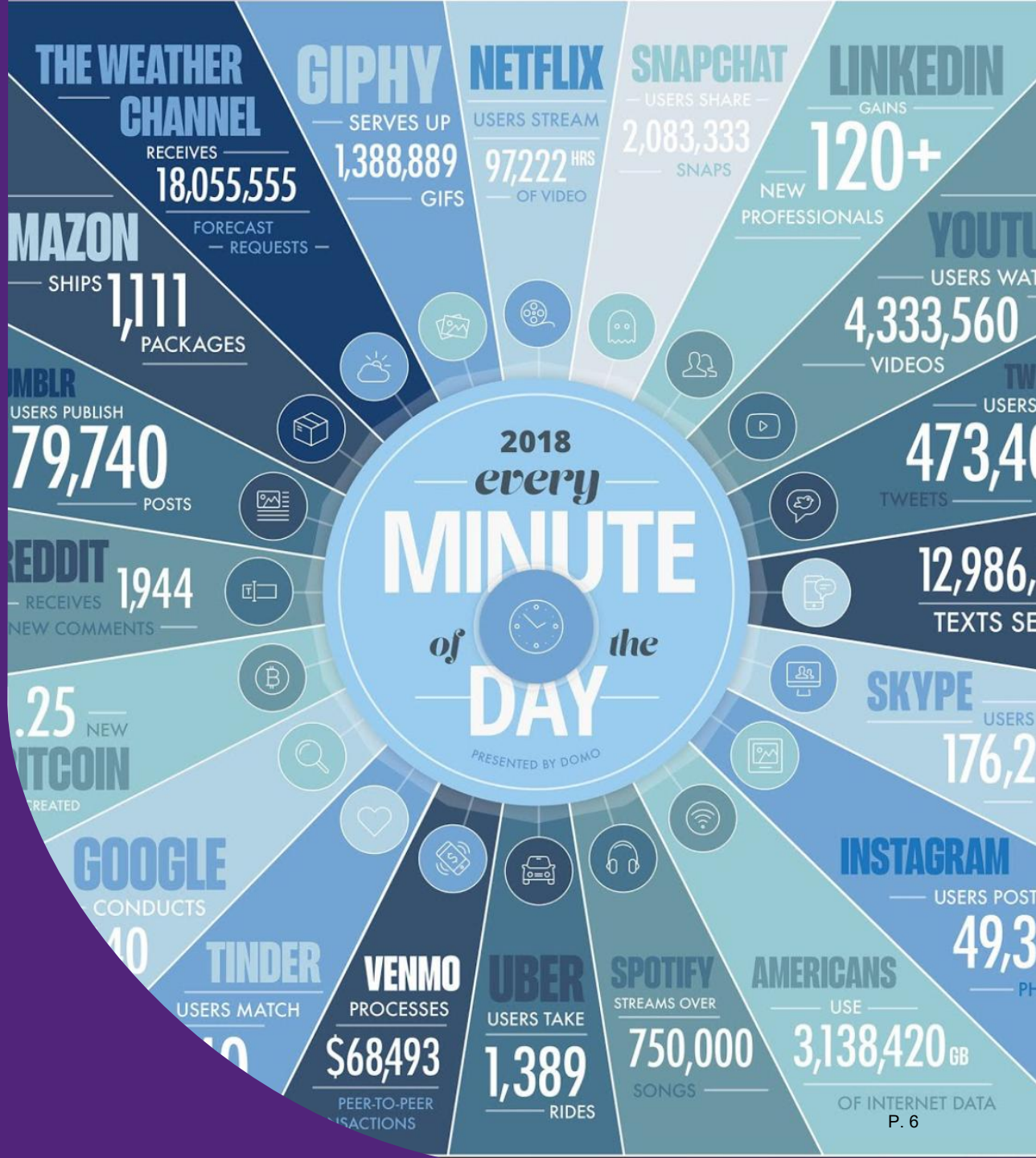
- TikTok generated an estimated **$9.4 billion** revenue in 2022, a 100% increase YoY
- TikTok had **1.7 billion** monthly active users in 2023 and is expected to reach two billion by the end of 2024.
- TikTok has been downloaded over three billion times

https://marketingtochina.com/wechat-statistics/#:~:text=WeChat%20will%20exceed%201.67%20users,the%20end%20of%20the%20year.
https://blog.hootsuite.com/facebook-statistics/

Big Data Statistics*

How much data are we generating?

- There were **79 zettabytes** of data generated worldwide in **2021 and 90%** of the data in the global datasphere is replicated data.
- By 2025, more than **150 zettabytes** of big data will need analysis.
- The COVID-19 pandemic increased the rate of data breaches by more than **400%**.
- By 2027, the use of big data application database solutions and analytics is predicted to grow to $12 billion.



Big Data Statistics 2023: How Much Data is in The World?
https://firstsiteguide.com/big-data-stats/

Measuring the Size of Data

Bytes (8 bits)
Kilobyte
1,024 bytes; 2^{10}; approx. 1,000 or 10^{3}
2 Kilobytes: Typewritten page

Megabyte
1,048,576 bytes; 2^{20};
approx 1,000,000 or 10^{6}
5 Megabytes: Complete works of Shakespeare

Gigabyte
1,073,741,824 bytes; 2^{30};
approx 1,000,000,000 or 10^{9}
20 Gigabytes: Audio collection of the works of Beethoven
Terabyte
1,099,511,627,776 or 2^{40};
approx. 1,000,000,000,000 or 10^{12}
10 Terabytes: Printed collection of the U. S. Library of Congress with 130 million items on about 530 miles of bookshelves, including 29 million book, 2.7 million recordings, 12 million photographs, 4.8 million maps, and 58 million manuscripts

Petabyte
1,125,899,906,842,624 bytes or 2^{50}
approx. 1,000,000,000,000,000 or 10^{15}
2 Petabytes: All U. S. academic research libraries
Exabyte
1,152,921,504,606,846,976 bytes or 2^{60}
approx. 1,000,000,000,000,000,000 or 10^{18}
5 Exabytes: All words ever spoken by human beings.
Zettabyte
1,180,591,620,717,411,303,424 bytes or 2^{70}
approx. 1,000,000,000,000,000,000,000 or 10^{21}
Yottabyte
1,208,925,819,614,629,174,706,176 bytes or 2^{80}
approx. 1,000,000,000,000,000,000,000,000 or 10^{24}

274,877,906,944 X

80mm

≈ 21,990,232.5 km

384,402 km between Earth and Moon: 57.2 times of the distance and light travels 73.3s

https://en.wikipedia.org/wiki/Floppy_disk

Data Centre Technology

A data centre is a specialised IT infrastructure that houses centralised IT resources

- Servers (rack in cabinet);
- Databases and software systems;
- Networking and telecommunication devices.

Typical technologies and components

- Virtualisation
- Standardisation and Modularity
- Remote Operation and Management
- High Availability
- Security-Aware Design, Operation and Management
- Facilities

Hardware: Array of Hard Disks

It's essential to balance the factors below when selecting **Commodity Hard Drives in DC:**

- •Capacity.
- •Performance: IOPS (Input/Output Operations Per Second), Latency, Throughput
- •Durability and Reliability
- •Form Factor
- •Power Consumption
- •Heat Dissipation
- •Noise Level
- •Hot-Swappable Capability
- •Etc.

When grouping arrays of hard disks in a data center, **RAID (Redundant Array of Independent Disks)** configurations or storage arrays are in use.

- •**RAID 0**: Data is split across all disks in the array. Increases performance since multiple disks can be read or written to concurrently. Offers no redundancy; if one disk fails, all data is lost.
- •**RAID 1**: Data is duplicated across two disks. Provides redundancy since data exists in two places. Read performance is improved, but write performance remains the same as a single disk. Storage capacity is 50% of the total raw capacity because of mirroring.
- •Etc.

https://www.wikihow.com/Recover-a-Dead-Hard-Disk

http://www.comnetdrc.com/

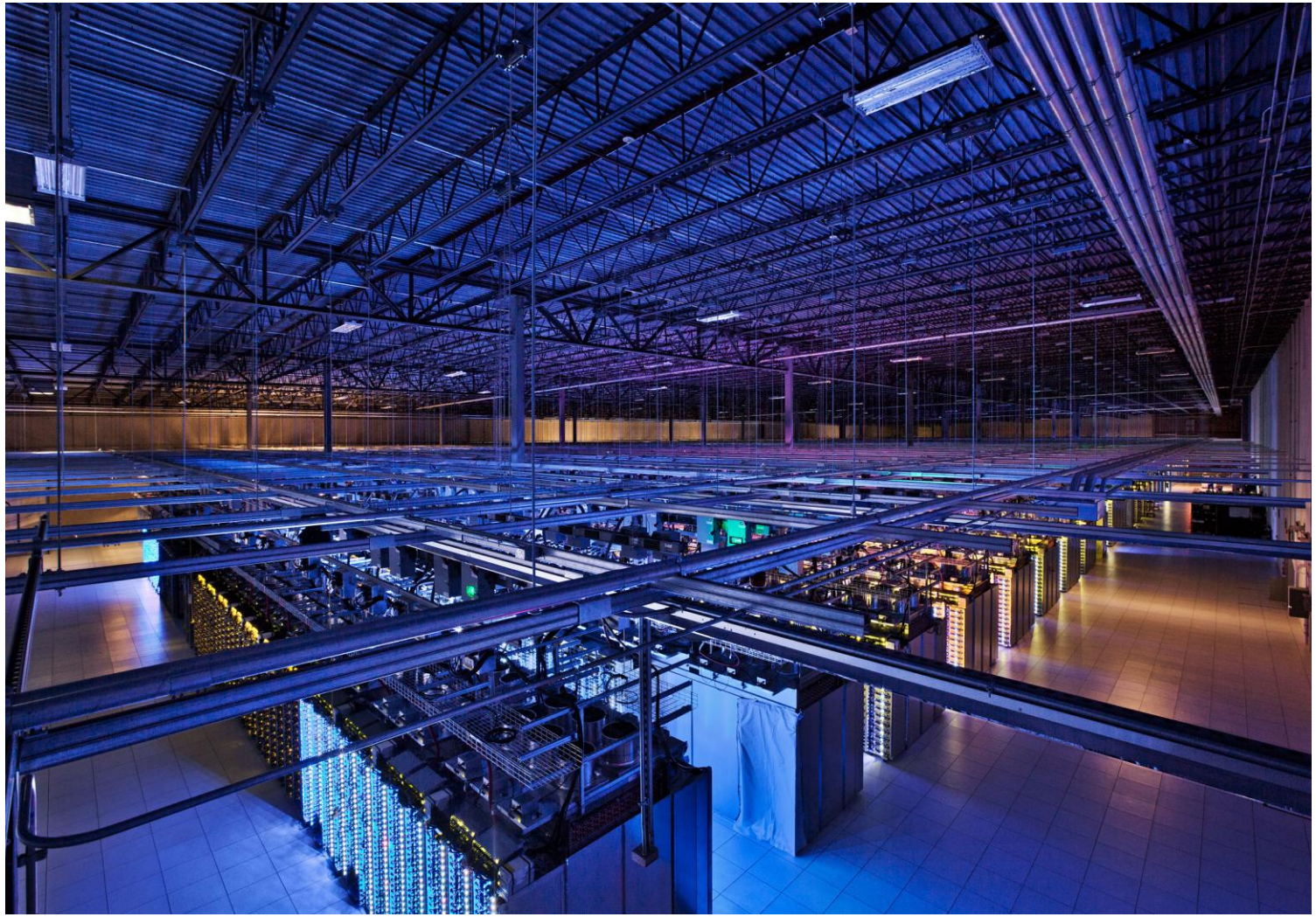
Hardware: Network Switch


Network **switches** and **routers** are crucial components in a data center, ensuring efficient communication and connectivity. Several factors to be considered:

- **Performance** and **Throughput**:
  - **Switching Capacity**: Ability of the switch to process data.
  - **Forwarding Rate**: Rate at which packets are processed.
  - **Bandwidth**: Adequate port speeds (e.g., 1Gbps, 10Gbps, 40Gbps, 100Gbps).
- **Port Density**: Number of ports available on the device. High-density switches can support more connected devices in a compact form factor, optimizing rack space.
- **Latency**: Time taken for a packet to traverse the switch/router. In many applications, especially financial or real-time processing, low latency is crucial.
- **Security**:
  - ACLs (Access Control Lists), port security, DDoS protection, and other security features to safeguard the network.
  - Support for 802.1X (port-based network access control) and other authentication mechanisms.
- Etc.


Network Switch Can Route 15 Terabits Per Second
http://exd-int.com/my-product/space-mind/

Google Data Centre

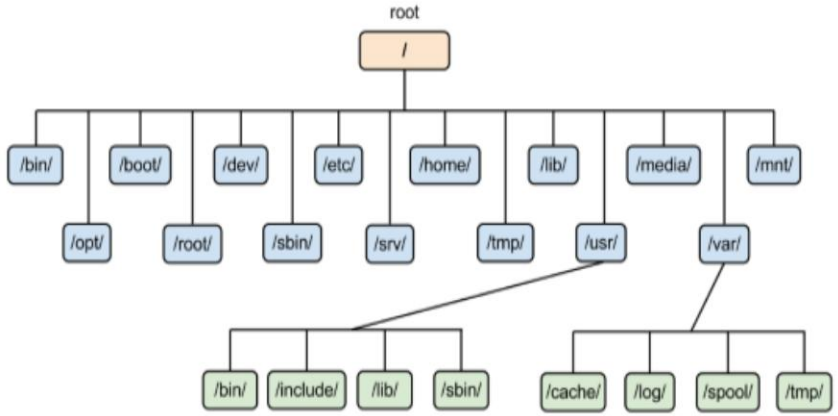


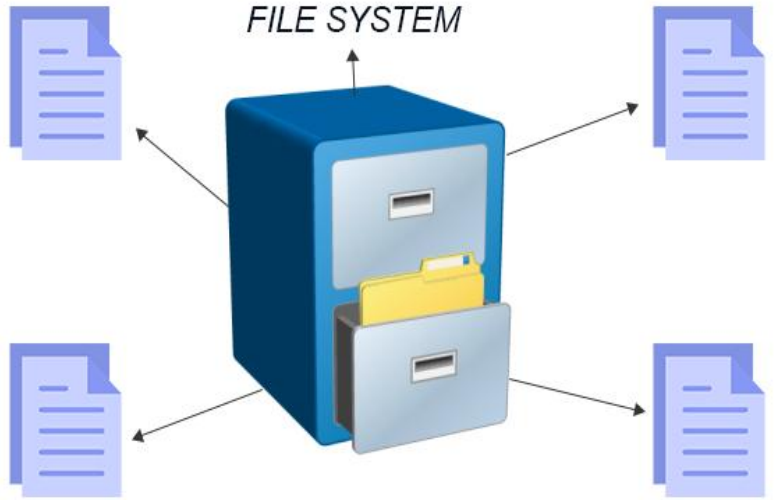
Outline

- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

What is a File System?

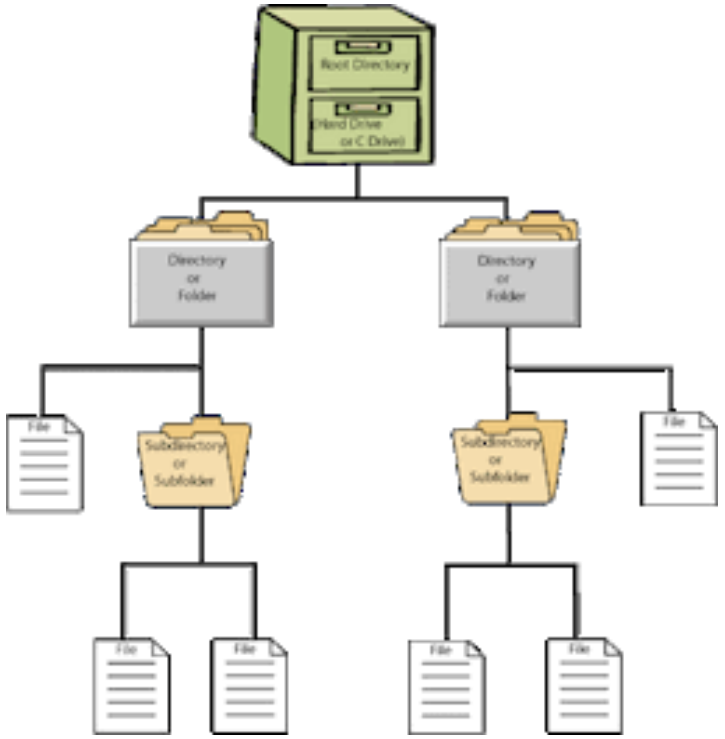
- A file system is an abstraction: enable users to manipulate and organize data.
- Typically, FS is in a hierarchical tree: files and directories.
- FS enables a uniform view, independent of the underlying storage devices: floppy/optical drives, hard drives and USB stick, etc.
- The connection between the logical file system and the storage device was typically a one-to-one mapping.
- Examples:
  - Windows: NTFS, FAT32, FAT
  - MacOS: Apple File System (AFS),
  - Linux: Ext4, etc.





What is a Distributed File System (DFS)?

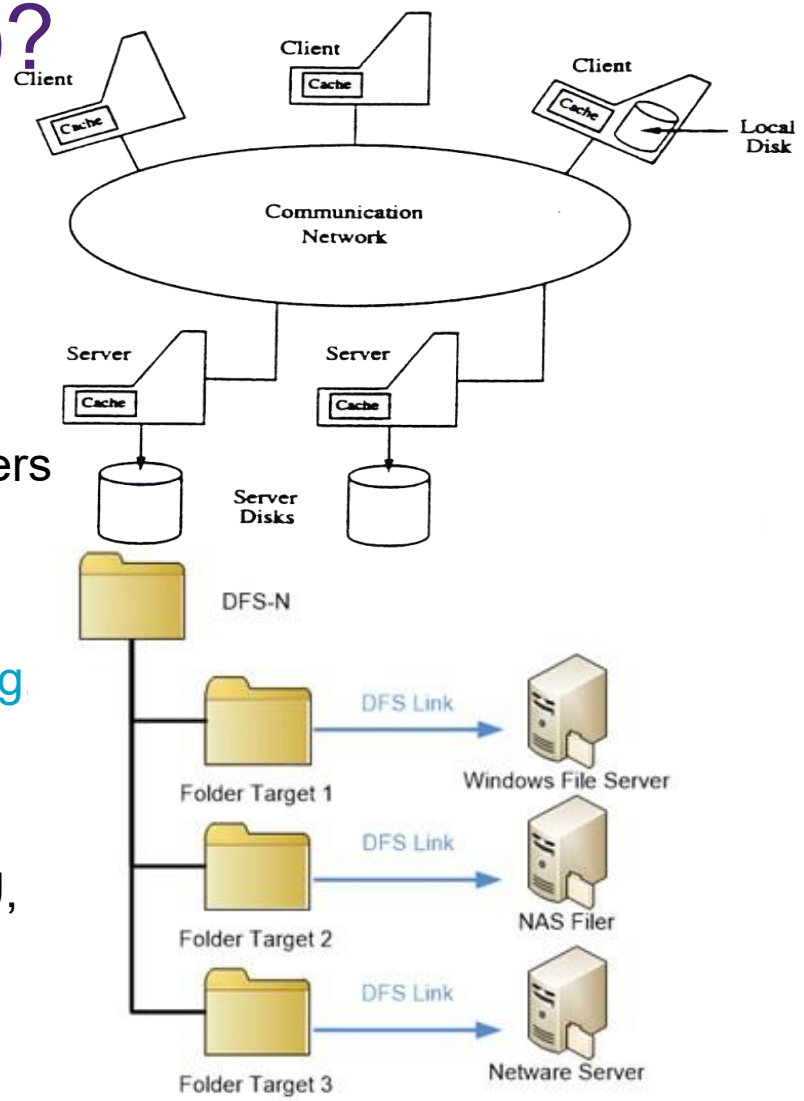
- DFS is a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources.
- A distributed file system spreads over multiple, autonomous computers.
- A distributed file system should have following characteristics:
  - Access transparency
  - Location transparency
  - Concurrency transparency
  - Failure transparency
  - Replication transparency
  - Migration transparency
  - Heterogeneity
  - Scalability



https://en.wikipedia.org/wiki/Clustered_file_system#Distributed_file_systems

What is a Distributed File System (DFS)?

- In general, files in a DFS can be located in "any" system.
  - Servers: data holders (the "source(s)" of files)
  - Clients: data users who are accessing the servers.
- Potentially, a server for a file can become a client for another file.
- However, most distributed systems distinguish between clients and servers in more strict way:
  - Clients simply access files and do not have/share local files.
  - Even if clients have disks, they (disks) are used for swapping, caching loading the OS, etc.
  - Servers are the actual sources of files.
  - In most cases, servers are more powerful machines (in terms of CPU, physical memory, disk bandwidth, ..)



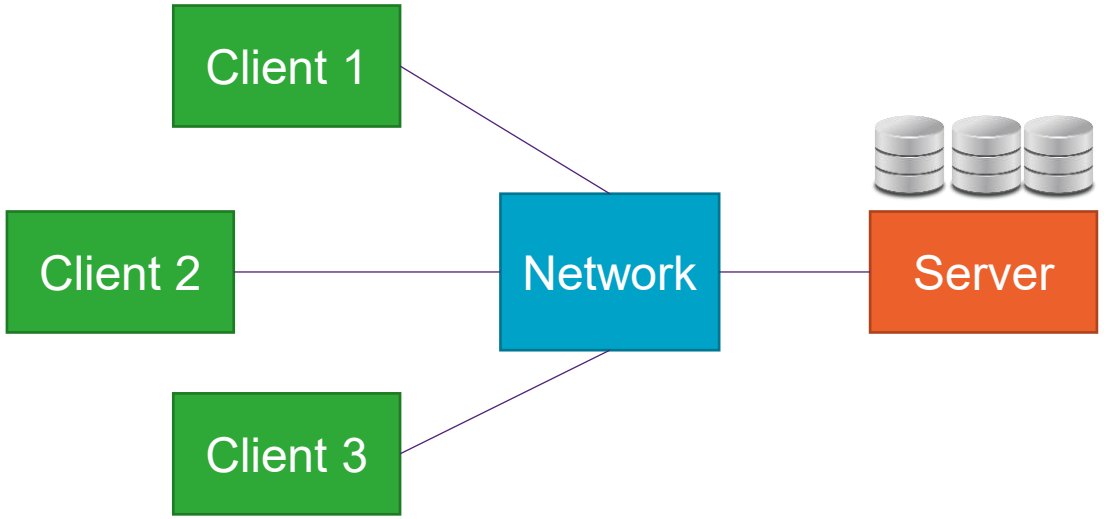
https://goo.gl/images/Nwq6tV https://www.slideshare.net/AnamikaSingh211/distributed-file-system-72294718

Sun Network File System (NFS)

- Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984.
- NFS allows a user on a client computer to access files over a computer network much like local storage is accessed.
- NFS is a client-server application, where a user can view, store and update the files on a remote computer.
- NFS builds on the Remote Procedure Call (RPC) system to route requests between clients and servers.
  - **RPC**: It enables programs to execute **procedures** in another address space (commonly on another computer on a shared network) as if they were local calls, making the process of developing network-shared interfaces more straightforward.
  - **Example**: Given a database server and a client; with RPC, the client can make requests to the server by **invoking procedures** rather than **requesting data**. (Efficiency)
- NFS protocol is designed to be independent of computer, operating system, network architecture, and transport protocol.
- NFS allows the user or system administrator to mount complete or a partial file system on a server.
- The portion of the file system that is mounted can be accessed by clients with different privileges (e.g. read-only or read-write)

Sun Network File System (NFS)

- Design:



Advantages:

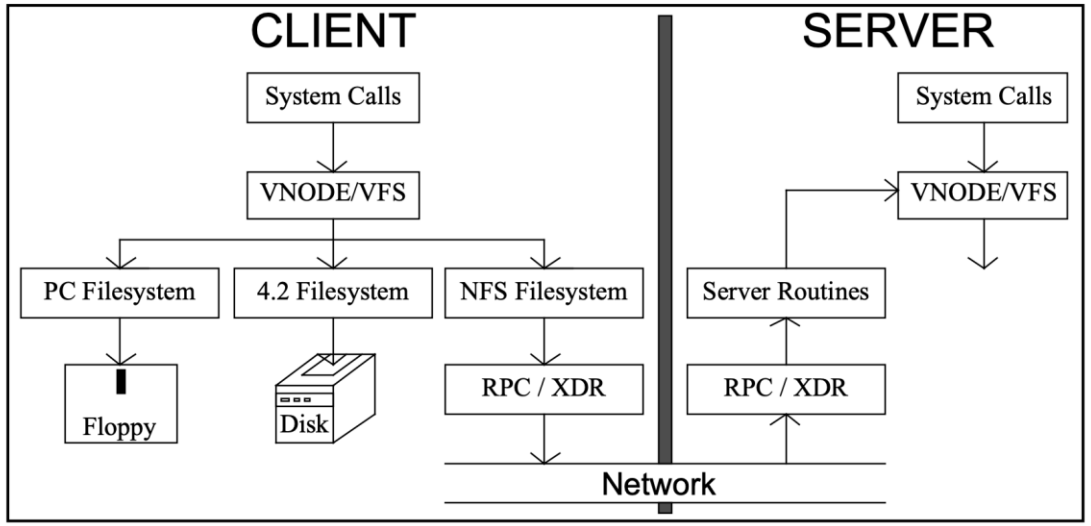
- easy sharing of data across clients
- centralized administration (backup done on multiple servers instead of many clients)
- security (put server behind firewall)

Network Attached Storage (NAS)

http://pages.cs.wisc.edu/~remzi/OSTEP/dist-nfs.pdf
https://www.westerndigital.com/en-au/solutions/network-attached-storage/NAS-for-family-sharing

Sun Network File System (NFS)

- Each file server presents a standard view of its local file system
- Transparent access to remote files
- Compatibility with multiple operating systems and platforms.
- Easy crash recovery and backup at server
- A software component namely, Virtual File System (VFS) is available in most OS (Operating Systems) as the interface to different local and distributed file systems.
- Virtual File System (VFS) in an OS (Operating System) acts as an interface between the system-call layer and all files in network nodes.
- 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.



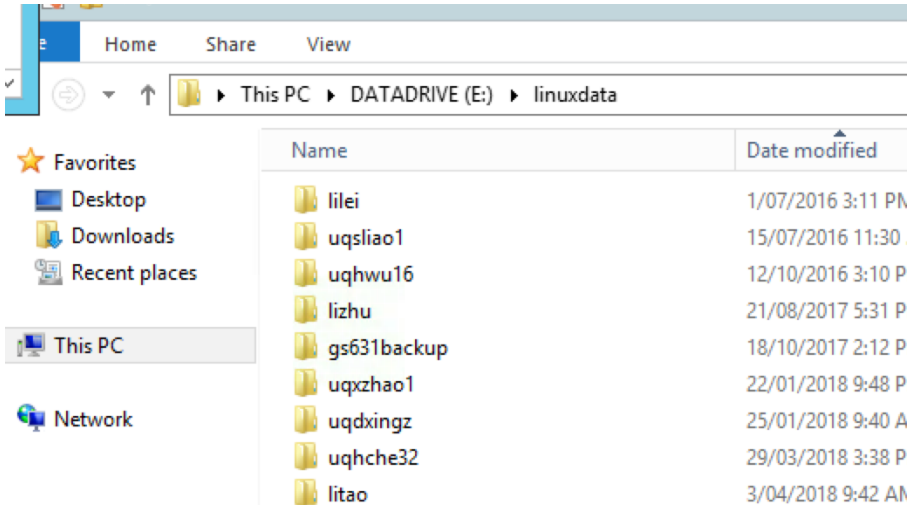
Making remote files as if local to the client

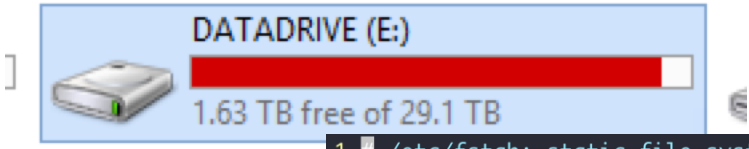
For Windows, the NFS is available via SMB - Server Message Block, or a version namely, CIFS – Common Internet File System.

http://pages.cs.wisc.edu/~remzi/OSTEP/dist-nfs.pdf

A Real Example of NFS

- A big hard drive (30 TB) on a windows server:





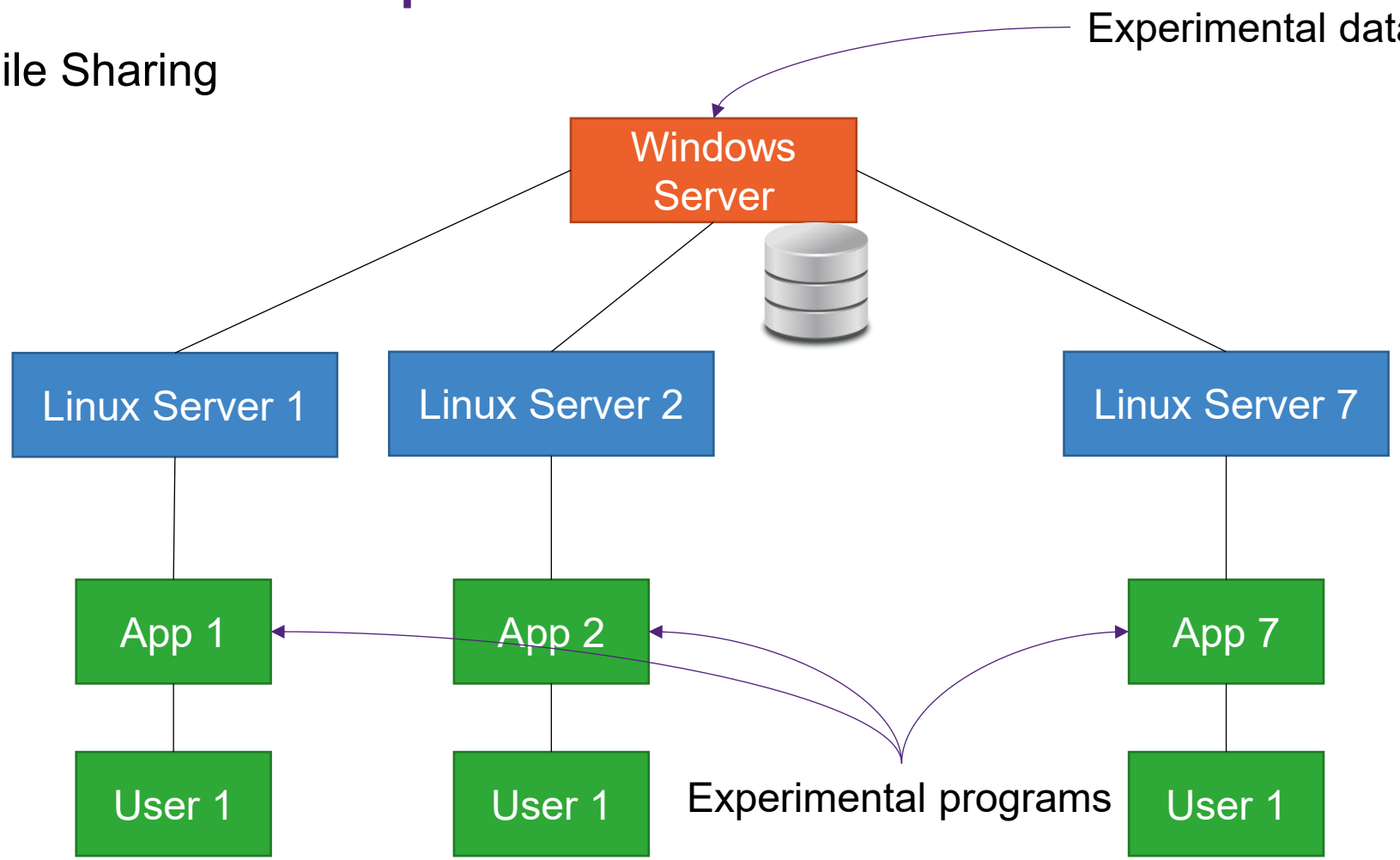
- The driver can be shared by users in Linux

```
*** System restart required ***
Last login: Fri Aug 30 14:39:28 2019 from 10.33.3.170
[redacted]~$ ls /media/
bigdata  dragon_data
```

```
1  # /etc/fstab: static file system information.
2  #
3  # Use 'blkid' to print the universally unique identifier for a
4  # device; this may be used with UUID= as a more robust way to name devices
5  # that works even if disks are added and removed. See fstab(5).
6  #
7  # <file system> <mount point>   <type>  <options>       <dump>  <pass>
8  # / was on /dev/sda1 during installation
9  UUID=da5ee708-72e0-4d0b-aa01-b5c12d9a11af /               ext4    errors=remount-ro
   0       1
10 # /home was on /dev/sda6 during installation
11 UUID=6a55da2f-1b5e-4a40-a566-6cd2707dcda4 /home           ext4    defaults        0
       2
12 # swap was on /dev/sda5 during installation
13 UUID=000ce652-a33d-459a-83e8-9c207785157e none            swap    sw              0
       0
14 #
15 /dev/sdc1 /local           ext4    defaults        0       2
16 #
17 //dragon/linuxdata  /media/dragon_data  cifs    username=[redacted],password=[redacted]
   [redacted],auto,dir_mode=0777,file_mode=0777    0    0
18 //daphne/linuxdata  /media/bigdata  cifs    username=[redacted],password=[redacted]
   [redacted]or,auto,dir_mode=0777,file_mode=0777  0    0
```

A Real Example of NFS

- File Sharing



Limitations:

- Data Scale
- High Availability
- Redundancy

Outline

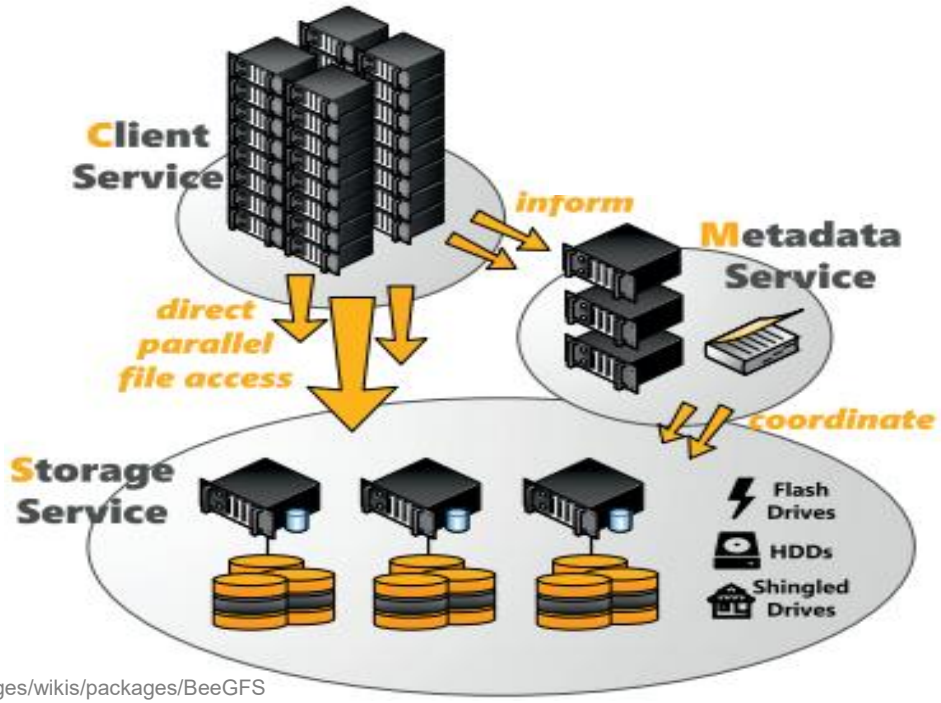
- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

Why using Clustered File System (CFS)?

- For bigger scale of data storage, a cluster of thousands of data servers is required.
- Scalability and availability should be provided for big data storage.
- Resiliency and load balancing are also very essential.

What is a Clustered File System (CFS)?

- Clustered File System (CFS) is not a single server with a set of clients, but instead a cluster of servers that all work together to provide high performance service to their clients.
- To the clients of CFS, the cluster is transparent.
- A CFS can organize the storage and access data across all clusters.

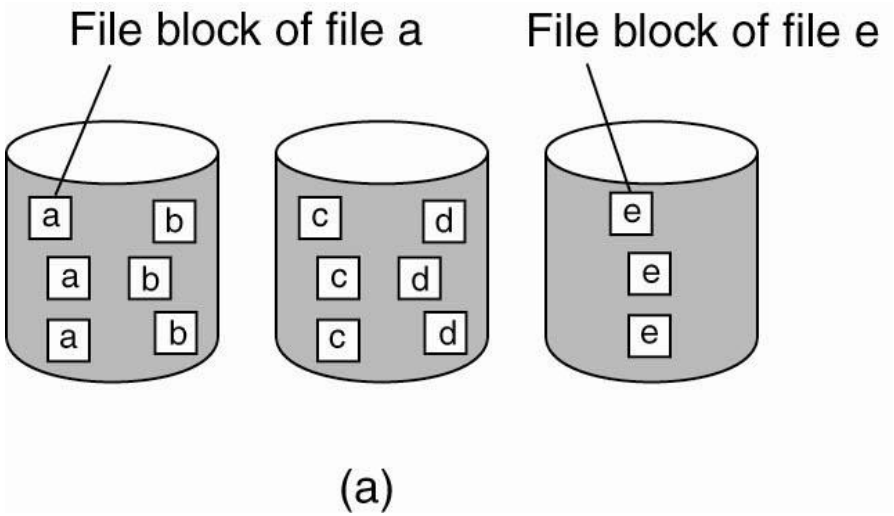


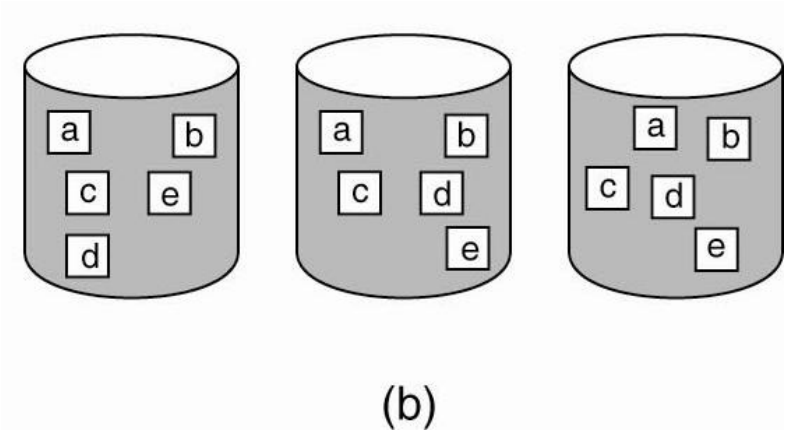
CFS is Managed at a Block Level

The difference between

(a) distributing whole files across several servers and
(b) striping files for parallel access.

File A can be divided into blocks





What are the differences among DFS, NFS, CFS?

Distributed File System (DFS): A file system that has its components spread across multiple systems. On the other hand, an NFS is inherently a distributed file system as well. The client component is on a different system than the underlying physical storage or its management.

Client-Server Architecture

Cluster Server Architecture

Network File system (NFS): Files are not local. They are served over a network, with the physical storage units and their management hosted by a different entity.

Clustered File System (CFS): It is built by pooling several different discrete components, typically multiple servers, multiple disks, working together to provide a unified namespace. A client is not aware of the physical boundaries that make up the file system.

https://www.quora.com/What-is-the-difference-between-a-distributed-file-system-clustered-file-system-and-a-network-file-system

Differences btw NFS and CFS

Feature	NFS (Network File System)	CFS (Clustered File System)
Architecture	Client-server architecture, where clients access a shared file system over the network.	Multiple nodes in a cluster simultaneously access and manage a shared file system.
File Access	Files are accessed over a network from a single server.	Files are accessed directly by multiple nodes in the cluster.
Scalability	Limited by the single-server architecture, which can become a bottleneck.	Highly scalable as it allows multiple nodes to access and manage the file system.
Performance	Performance is dependent on the server's capacity and network latency.	High performance due to parallel access by nodes, reducing bottlenecks.
Fault Tolerance	Limited fault tolerance; server failure can cause disruption in access.	High fault tolerance; distributed nature helps in recovery and avoiding single points of failure.
Data Consistency	Managed by the server; consistency is simpler but limited to single-server architecture.	Ensures data consistency across multiple nodes with complex consistency mechanisms.
Use Case	Suitable for sharing files over a network in smaller-scale environments.	Ideal for large-scale environments requiring high availability, like big data or high-performance computing clusters.
Management	Easier to manage as it involves a single server.	More complex to manage due to its distributed nature.
Concurrency	Limited concurrency as a single server handles requests.	Supports high concurrency as multiple nodes can access the file system simultaneously.

Outline

- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

Google File System

- **Google File System** (**GFS** or GoogleFS) is a scalable distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware.
- Shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability.
- GFS was internally used and became the creation basis of Hadoop Distributed File System (HDFS)

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

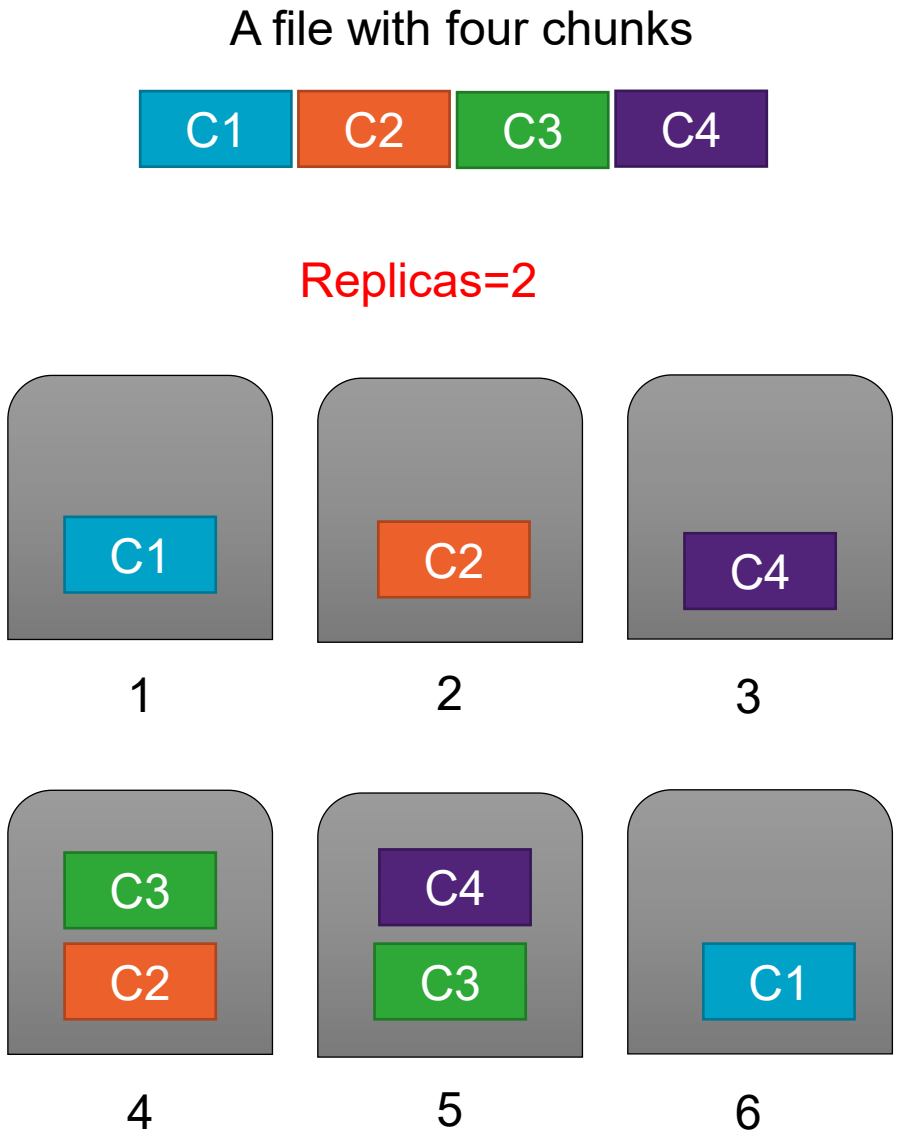
https://en.wikipedia.org/wiki/Google_File_System#/media/File:GoogleFileSystemGFS.svg

Design Assumptions

- Thousands of commodity computers
  - cheap hardware but distributed
- GFS has high component failure rates
  - System is built from many inexpensive commodity components
- Modest number of huge files
  - A few million files, each typically 100MB or larger (Multi-GB files are common)
  - No need to optimize for small files
- Workloads : two kinds of reads and writes
  - Large streaming reads (1MB or more) and small random reads (a few KBs)
  - Sequential appends to files by hundreds of data producers
- High sustained throughput is more important than latency
  - Response time for individual read and write is not critical

GFS Design Overview

- Files stored as chunks
  - With a fixed size of 64MB each.
  - With a 64-bit ID each.
- Reliability through replication
  - Each chunk is replicated across 3 (by default) or more chunk servers
  - Replication number can be manually set
- Single Master
  - Centralized management
  - Only store meta-data



GFS Architecture – Read

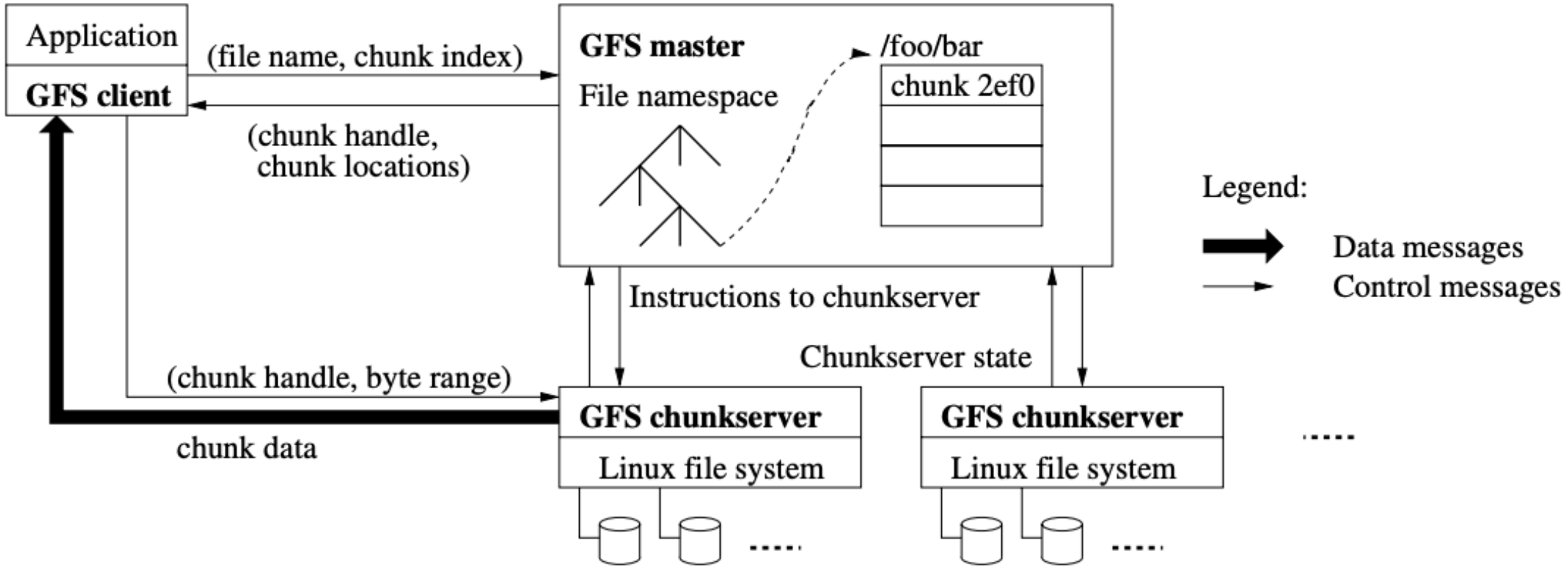


Figure 1: GFS Architecture

GFS Architecture – Read

The reading process in Google's File System (GFS) follows a series of specific key steps below:

1. **Client Request**: The client issues a request to read a file from GFS. The request specifies the file name and the byte range to be read.
2. **Communicating with the Master Server**:

- The client contacts the GFS Master server to obtain metadata about the file. Specifically, the client requests the file's chunk locations, including which chunk servers hold the replicas of the required file chunks.
- The Master responds by providing the client with the chunk metadata (i.e., the chunk handle and the locations of replicas on chunk servers).

3. **Contacting the Chunk Servers**:

- Once the client knows the chunk locations, it directly contacts the closest chunk server that holds the relevant replica of the chunk it needs to read.
- If multiple replicas exist, the client can select the closest chunk server to minimise network latency.

4. **Reading the Data**:

- The client sends a read request to the chosen chunk server, specifying the chunk handle and the byte range of the chunk it needs.
- The chunk server responds by reading the requested chunk data and sending it back to the client.

Handling Larger Files: If the requested data spans multiple chunks, the client will repeat the process for subsequent chunks by contacting other chunk servers until it has read the entire requested byte range.
Caching Metadata: The client caches the metadata (chunk locations) provided by the Master to reduce communication overhead for future reads, avoiding repetitive requests to the Master server.
Data Integrity Check: GFS includes mechanisms to ensure data integrity. Each chunk server stores a checksum for every chunk. When a chunk server sends data to the client, it verifies the checksum to ensure that the data has not been corrupted during storage or transmission.
This process ensures that file reads are optimized for performance, with **minimal involvement of the Master server** after the initial metadata lookup.

GFS Architecture – Write

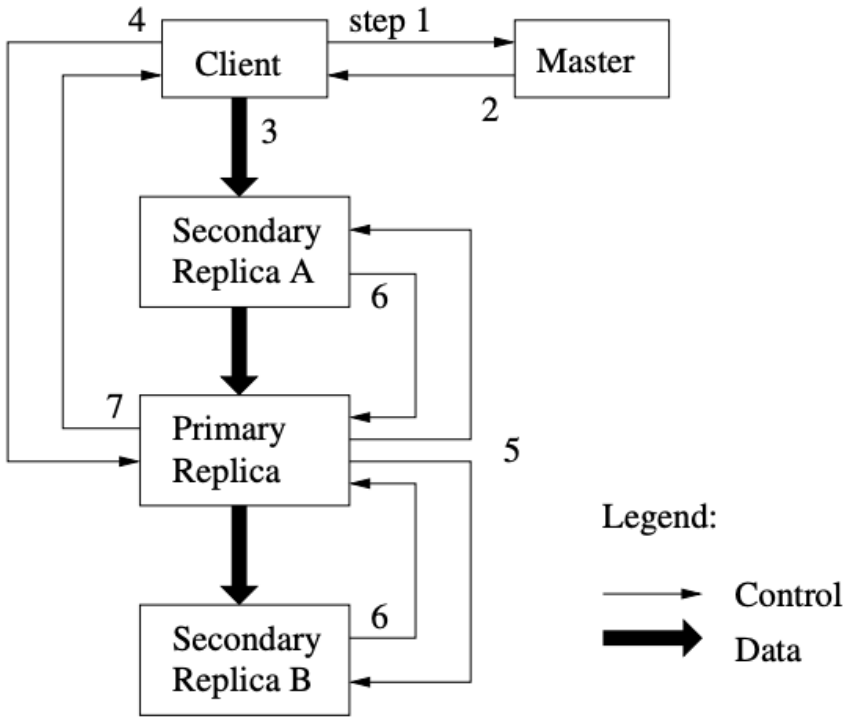


Figure 2: Write Control and Data Flow

1.Client asks master which chunk server holds current lease of chunk and locations of other replicas.

2.Master replies with identity of primary and locations of secondary replicas.

3.Client pushes data to all replicas

4.Once all replicas have acknowledged receiving the data, client sends write request to primary. The primary assigns consecutive serial numbers to all the mutations it receives, providing serialization. It applies mutations in serial number order.

5.Primary forwards write request to all secondary replicas. They apply mutations in the same serial number order.

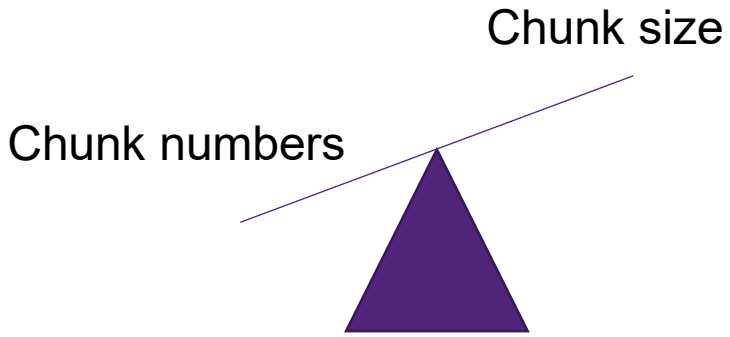
6.Secondary recplicas reply to primary indicating they have completed operation

7.Primary replies to the client with success or error message

GFS Components – Master

- Mater maintains all system metadata
  - Name space, access control info, file to chunk mappings, chunk locations, etc.
- Periodically communicates with chunk servers
  - Through HeartBeat messages
- Advantages:
  - Simplifies the design
- Disadvantages:
  - Single point of failure
- Solution
  - Replication of Master state on multiple machines
  - Operational log and check points are replicated on multiple machines

GFS Components – Chunks



- Fixed size of 64MB (vs 4kb of cluster size for NTFS)
- Advantages
  - Size of meta data is reduced
  - Involvement of Master is reduced
  - Network overhead is reduced
  - Lazy space allocation avoids internal fragmentation
- Disadvantages
  - Hot spots
    - A small file consists of a small number of chunks, perhaps just one. The chunkservers storing those chunks may become hot spots if many clients are accessing the same file.
    - Solutions: increase the replication factor and stagger application start times; allow clients to read data from other clients

https://support.microsoft.com/en-us/help/140365/default-cluster-size-for-ntfs-fat-and-exfat

GFS Components – Metadata and Operational Log

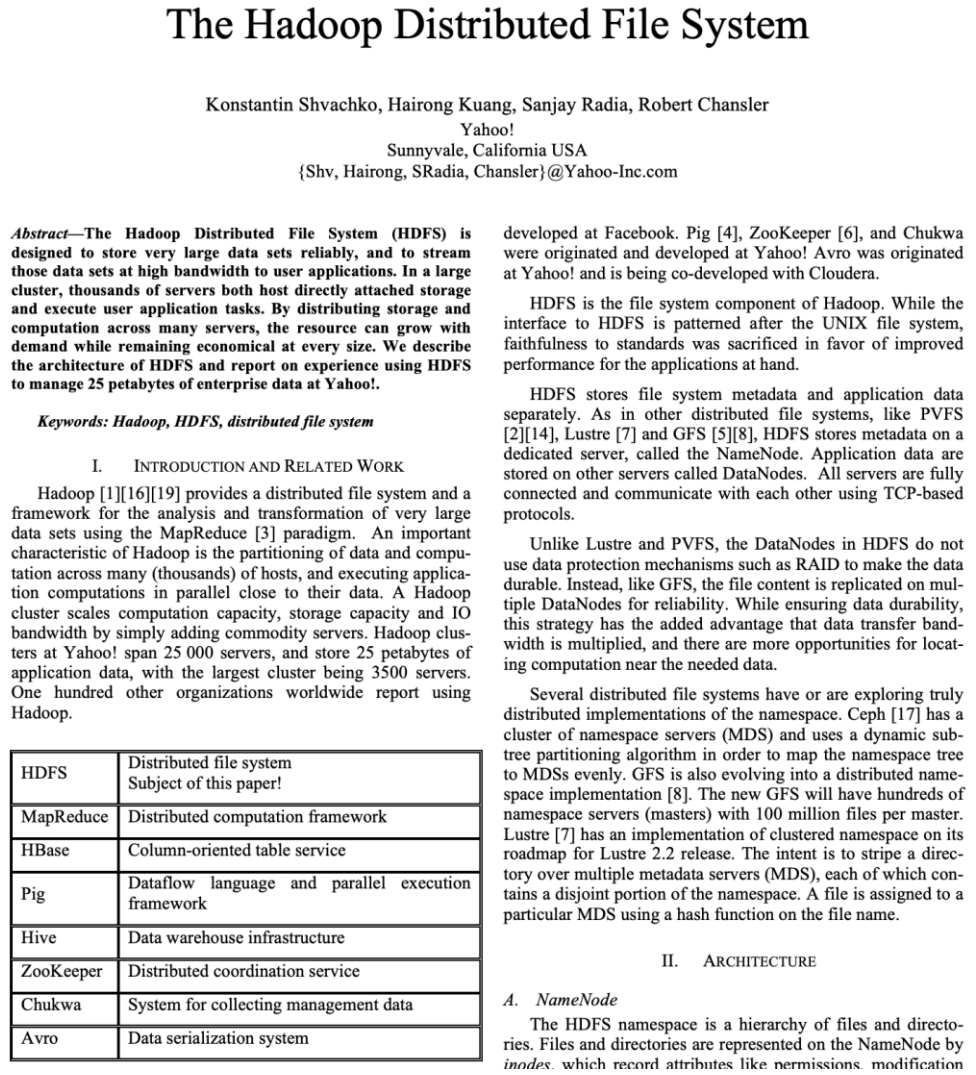
- Three major types of metadata
  - The file and chunk namespaces
  - The mapping from files to chunks
  - Locations of each chunk's replicas
- All the metadata is kept in the Master's memory
- 64MB chunk has 64 bytes of metadata
- Chunk locations
  - Chunk servers keep track of their chunks and relay data to Master through HeartBeat messages
- Master "operation log"
  - Consists of namespaces and file to chunk mappings
  - Replicated on remote machines

Outline

- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

Hadoop Distributed File System

- Apache Hadoop was proposed in 2010 as a collection of open-source software utilities to deal with big data problem.
- The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model.
  - Hadoop splits files into large blocks and distributes them across nodes in a cluster.
  - It then transfers packaged code into nodes
  - It takes advantage of data locality.



The Hadoop Distributed File System

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler
Yahoo!
Sunnyvale, California USA
{Shv, Hairong, SRadia, Chansler}@Yahoo-Inc.com

***Abstract*—The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 25 petabytes of enterprise data at Yahoo!.**

Keywords: Hadoop, HDFS, distributed file system

I. Introduction and Related Work

Hadoop [1][16][19] provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce [3] paradigm. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! span 25 000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations worldwide report using Hadoop.

HDFS	Distributed file system Subject of this paper!
MapReduce	Distributed computation framework
HBase	Column-oriented table service
Pig	Dataflow language and parallel execution framework
Hive	Data warehouse infrastructure
ZooKeeper	Distributed coordination service
Chukwa	System for collecting management data
Avro	Data serialization system

developed at Facebook. Pig [4], ZooKeeper [6], and Chukwa were originated and developed at Yahoo! Avro was originated at Yahoo! and is being co-developed with Cloudera.

HDFS is the file system component of Hadoop. While the interface to HDFS is patterned after the UNIX file system, faithfulness to standards was sacrificed in favor of improved performance for the applications at hand.

HDFS stores file system metadata and application data separately. As in other distributed file systems, like PVFS [2][14], Lustre [7] and GFS [5][8], HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols.

Unlike Lustre and PVFS, the DataNodes in HDFS do not use data protection mechanisms such as RAID to make the data durable. Instead, like GFS, the file content is replicated on multiple DataNodes for reliability. While ensuring data durability, this strategy has the added advantage that data transfer bandwidth is multiplied, and there are more opportunities for locating computation near the needed data.

Several distributed file systems have or are exploring truly distributed implementations of the namespace. Ceph [17] has a cluster of namespace servers (MDS) and uses a dynamic subtree partitioning algorithm in order to map the namespace tree to MDSs evenly. GFS is also evolving into a distributed namespace implementation [8]. The new GFS will have hundreds of namespace servers (masters) with 100 million files per master. Lustre [7] has an implementation of clustered namespace on its roadmap for Lustre 2.2 release. The intent is to stripe a directory over multiple metadata servers (MDS), each of which contains a disjoint portion of the namespace. A file is assigned to a particular MDS using a hash function on the file name.

II. Architecture

A. NameNode

The HDFS namespace is a hierarchy of files and directories. Files and directories are represented on the NameNode by *inodes*, which record attributes like permissions, modification

Shvachko, Konstantin, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In *MSST*, vol. 10, pp. 1-10. 2010

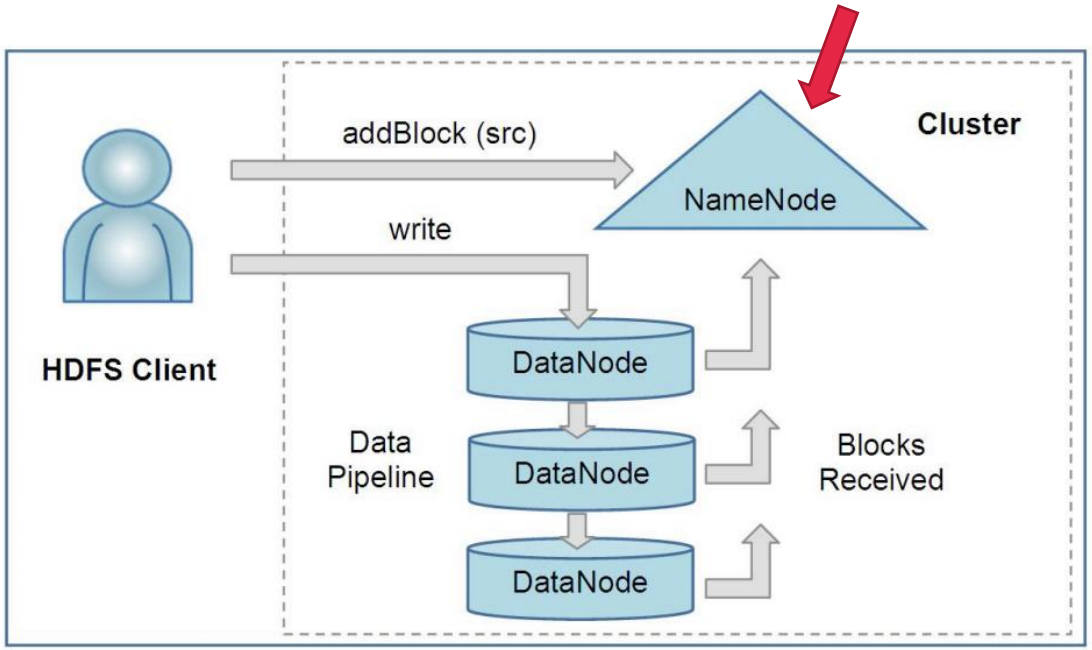
Design Motivations

- Many inexpensive commodity hardware and failures are very common
- Many big files: millions of files, ranging from MBs to GBs
- Two types of reads
  - Large streaming reads
  - Small random reads
- Once written, files are seldom modified
  - Random writes are supported but do not have to be efficient
- High sustained bandwidth is more important than low latency

HDFS Component – NameNode

NameNode

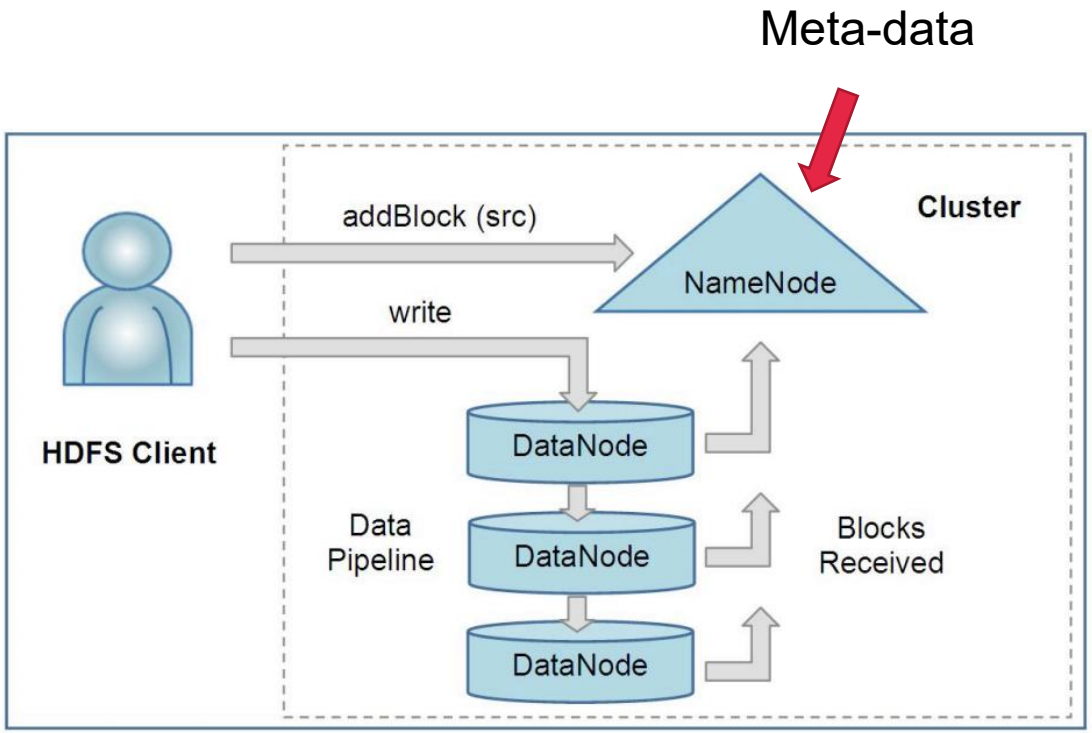
- Equivilent to Master Node in GFS
- represents files and directories on the NameNode as inode
- records attributes like permissions, modification and access times, namespace and disk space quotas.
- maintains the namespace tree and the mapping of file blocks to DataNodes
- When writing data, NameNode nominates a suite of three DataNodes to host the block replicas.
- The client then writes data to the DataNodes in a pipeline fashion.
- keeps meta-data in RAM



HDFS Component – Meta-Data

Meta-Data

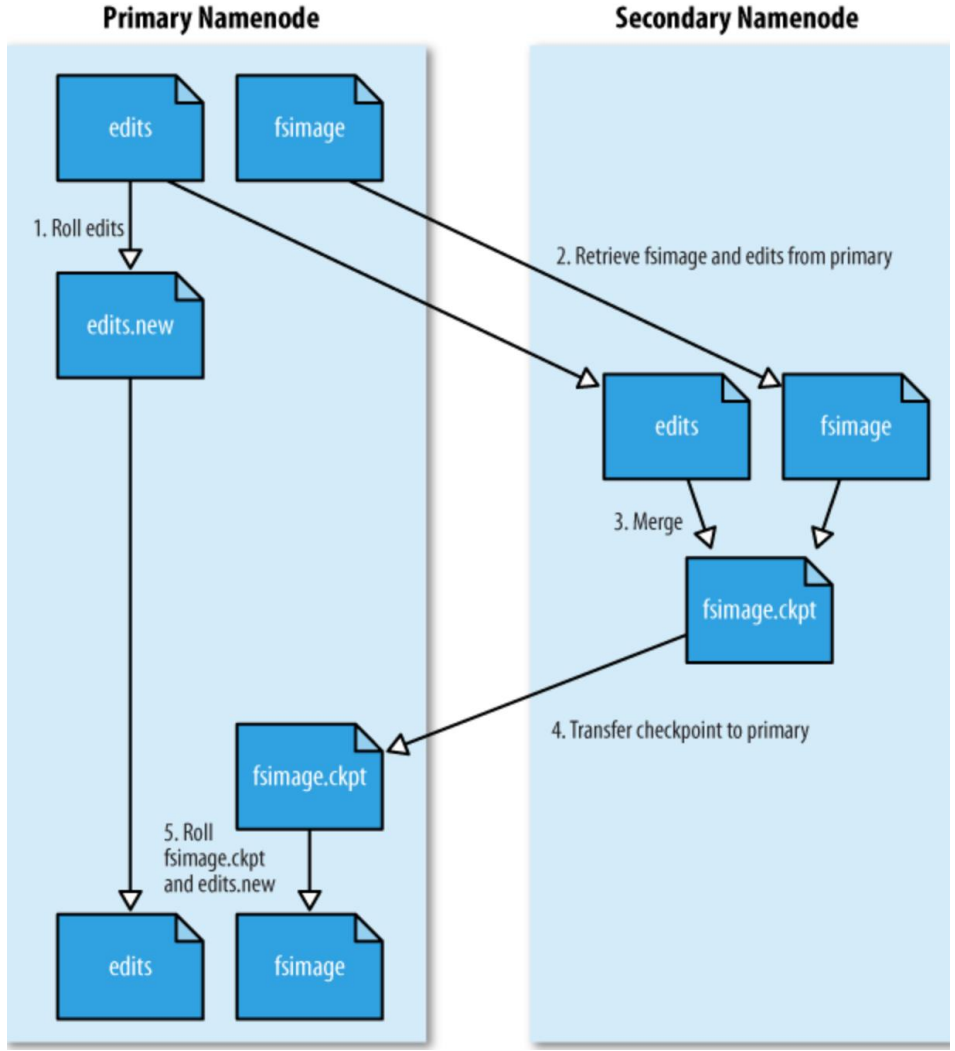
- fsimage:
  - contains the entire filesystem namespace at the latest checkpoint
    - Blocks information of the file (location, timestamp, etc.)
    - Folder information (ownership, access, etc.)
  - stored as an *image* file in the NameNode's local file system.
- editlog:
  - contains all the recent modifications made to the file system on the most recent fsImage.
  - create/update/delete requests from the client



HDFS Component – Checkpoint Node

Checkpoint Node (Secondary NameNode)

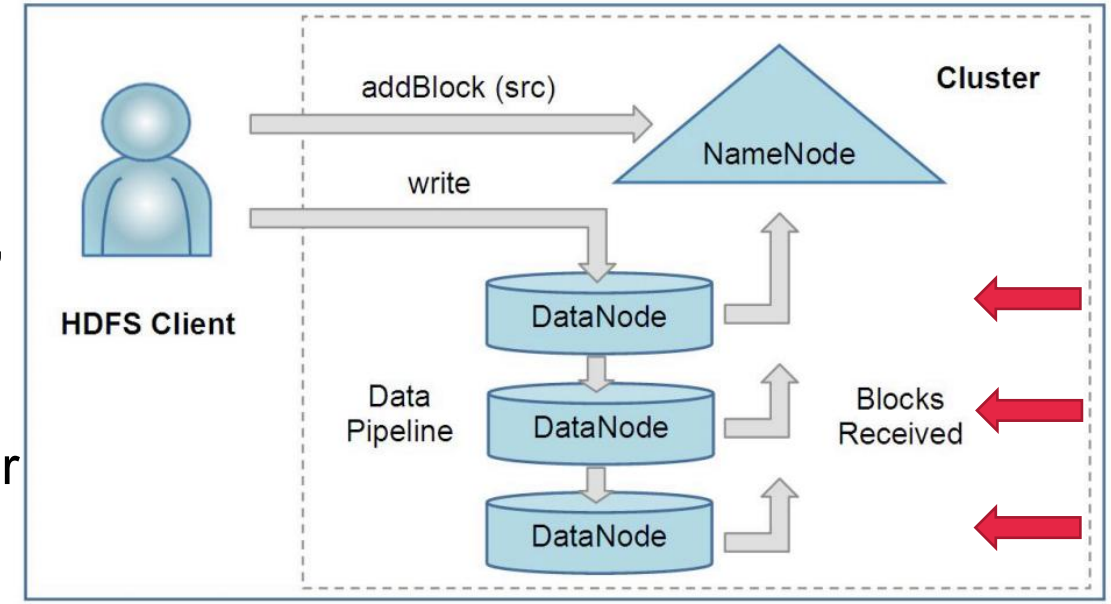
- "Secondary" does not mean "2nd" NameNode that acts as same or similar as the primary one.
- Regularly query for fsimage and editlogs (2)
- Primary NameNode will stop to write editlogs and copy edits and fsimage to secondary NameNode
- All new edits after that will be fed into edits.new (1).
- Copied edits and fsimage on secondary NameNode will be merged (3)
- Copy the merged fsimage.ckpt back to primary NameNode and use it as the new fsImage (edits.new will be used as the latest editlogs)
- Finally, editlogs file gets smaller and fsimage gets updated.



HDFS Component – DataNode

DataNode

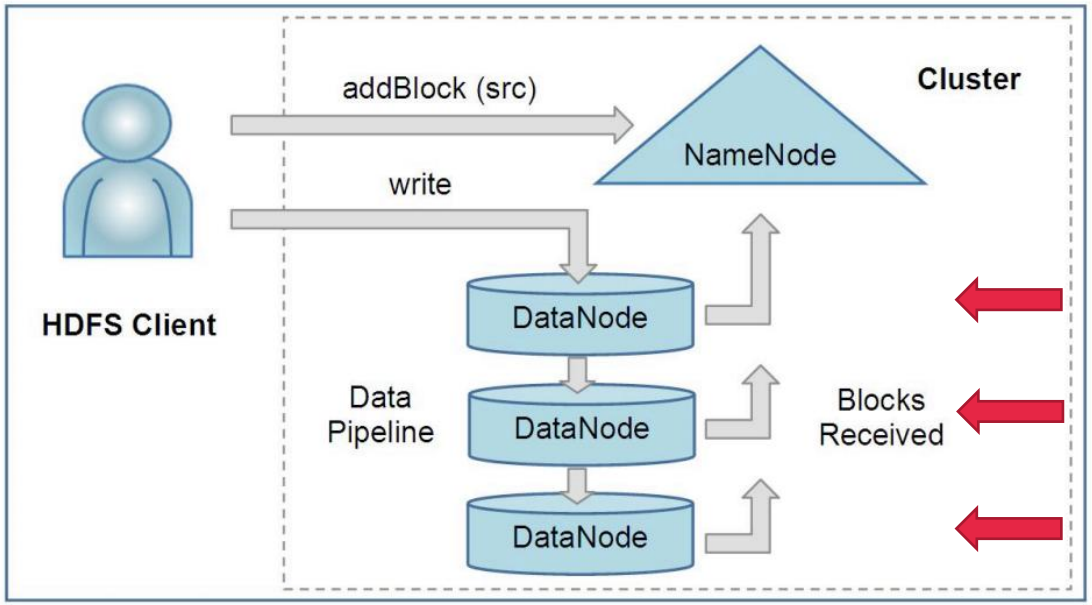
- Equivalent to chunk server in GFS
- Each block replica contains two files:
  - data itself
  - block's meta-data
- When startup, handshakes with NameNode
  - Verify namespace ID & software version
- Namespace ID is persistently stored on DataNodes, thus preserves the integrity.
- Internal storage ID is an identifier of the DataNode within the cluster (IP:port changed) and will be never changed after registration.



HDFS Component – DataNode

DataNode

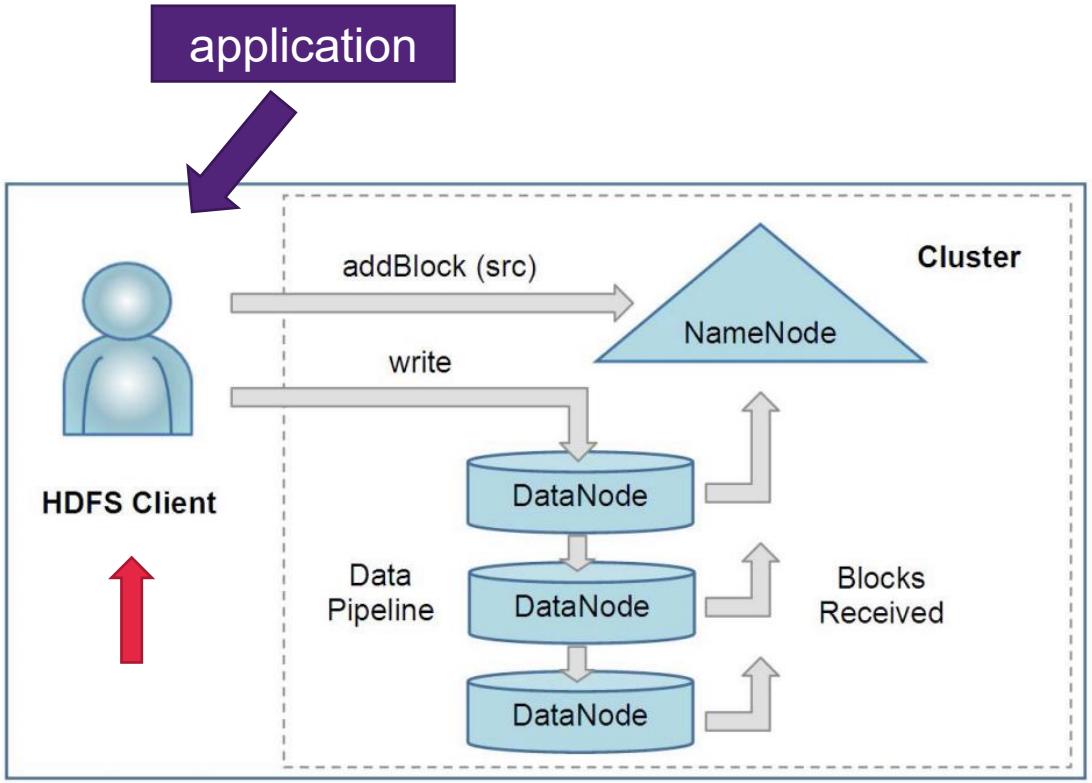
- During a normal operation, the available block replicas are also sent to NameNode.
- sends heartbeats (by default every 3 secs) to the NameNode
- No heartbeats from a DataNode in ten minutes:
  - DataNode is out of service
  - the block replicas hosted by that DataNode are unavailable
  - NameNode schedules creation of new replicas of those blocks on other DataNodes.
- receives maintenance commands from the NameNode indirectly (in replies to heartbeats).
  - replicate blocks to other nodes;
  - remove local block replicas;
  - re-register or to shut down the node;
  - send an immediate block report.



HDFS Component – Client

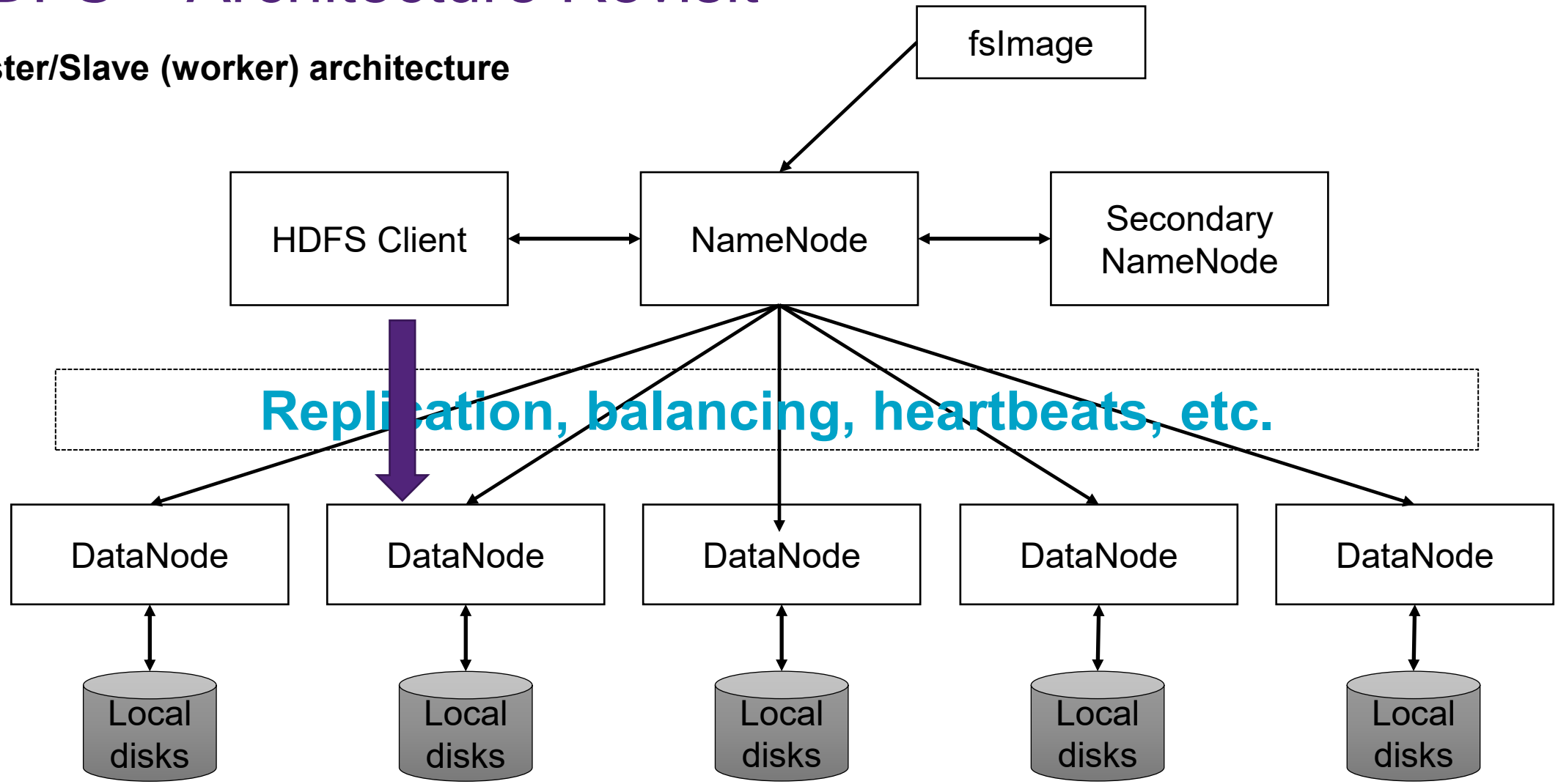
HDFS Client

- User applications access the file system using the HDFS client
- User application knows nothing about data storage.
- Reading:
  - first asks the NameNode for the list of DataNodes.
  - then contacts a DataNode directly and requests the transfer.
- Writing:
  - first asks the NameNode to choose DataNodes to host replicas of the first block of the file and organizes a pipeline from node-to-node and sends the data.
  - then requests new DataNodes to be chosen to host replicas of the next block as well as a new pipeline.
  - Each choice of DataNodes is likely to be different.



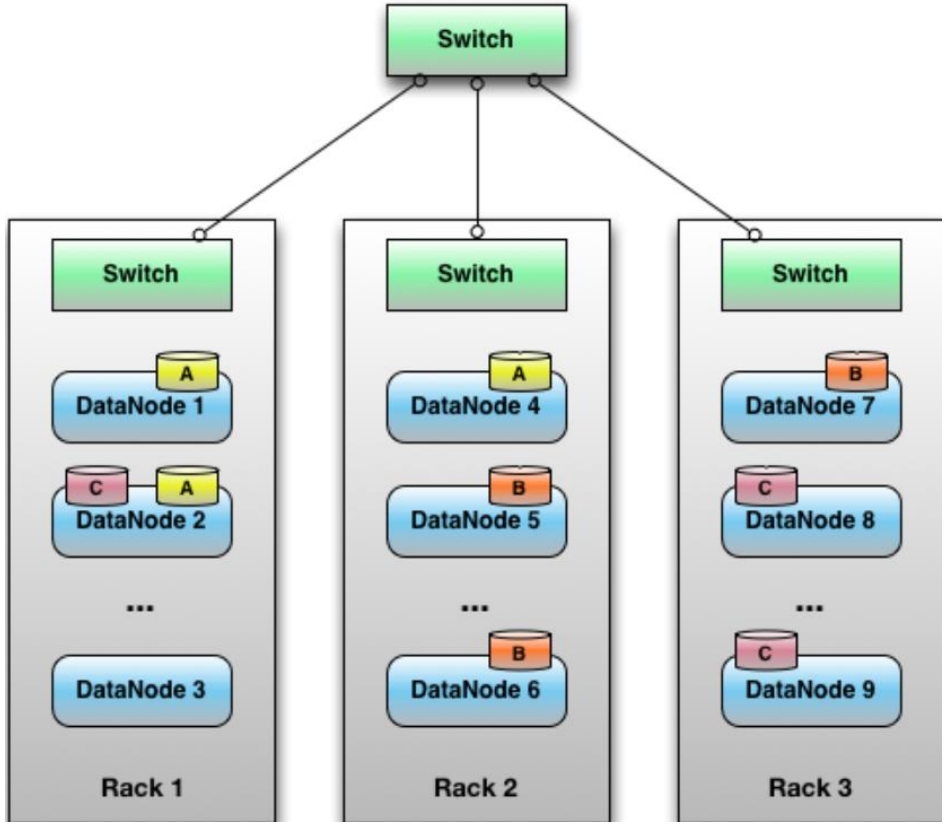
HDFS – Architecture Revisit

Master/Slave (worker) architecture



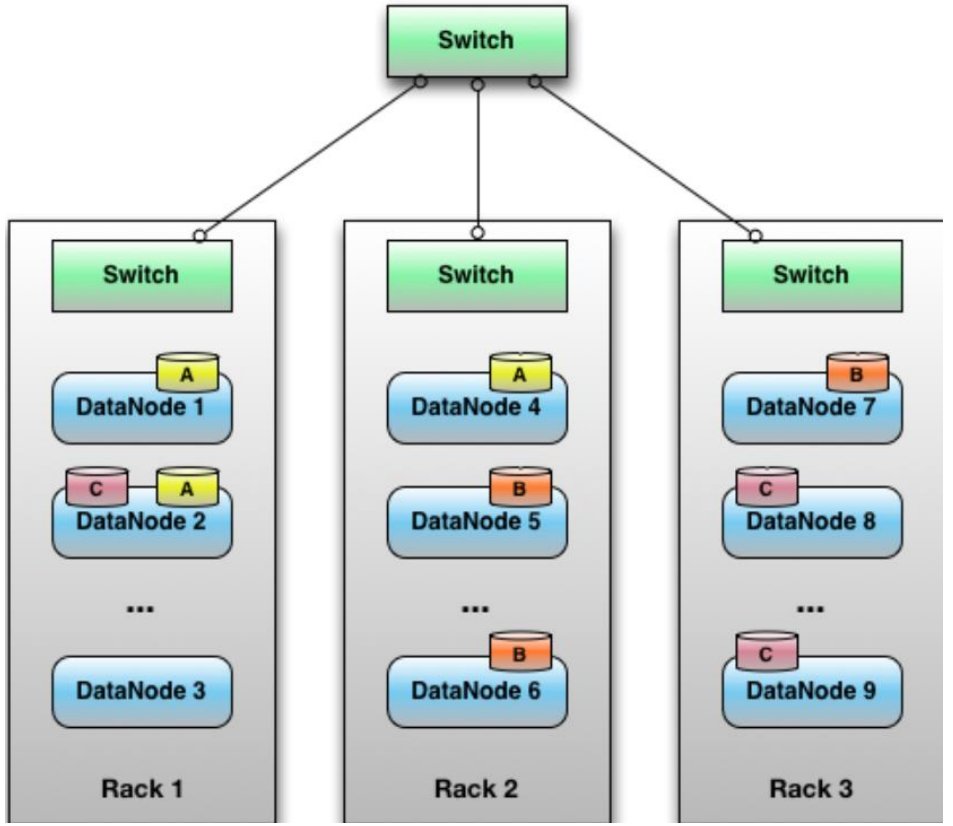
HDFS Block Placement

- For a large cluster, it may not be practical to connect all nodes in a flat topology.
- A common practice is to spread the nodes across multiple racks.
- Nodes of a rack share a switch, and rack switches are connected by one or more core switches.
- Communication between two nodes in different racks has to go through multiple switches.
- In most cases, network bandwidth between nodes in the same rack is greater than network bandwidth between nodes in different racks.



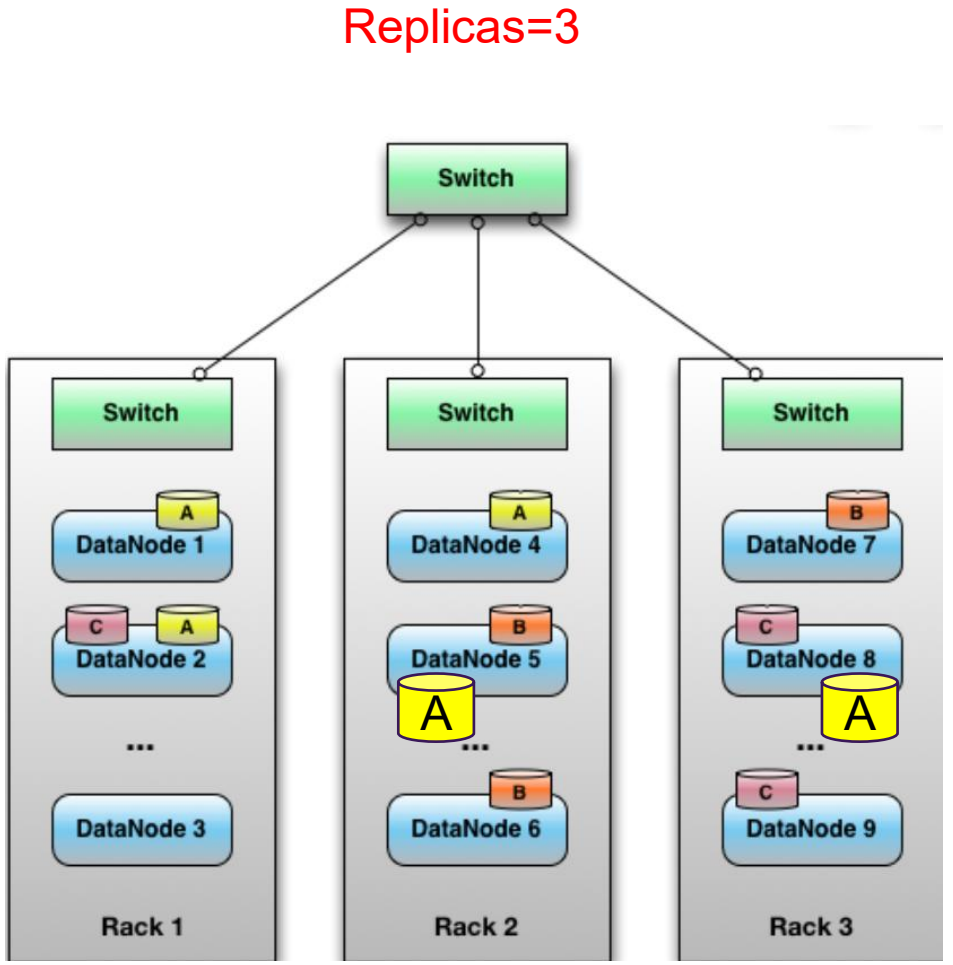
HDFS Block Placement Policy

- The default HDFS block placement policy provides a tradeoff between minimizing the write cost, and maximizing data reliability, availability and aggregate read bandwidth.
- When a new block is created, the policy is as follows:
  - HDFS places the first replica on the node where the writer is located,
  - the second and the third replicas on two different nodes in a different rack,
  - and the rest are placed on random nodes with restrictions
    - no more than one replica is placed at one node
    - no more than two replicas are placed in the same rack when the number of replicas is less than twice the number of racks.



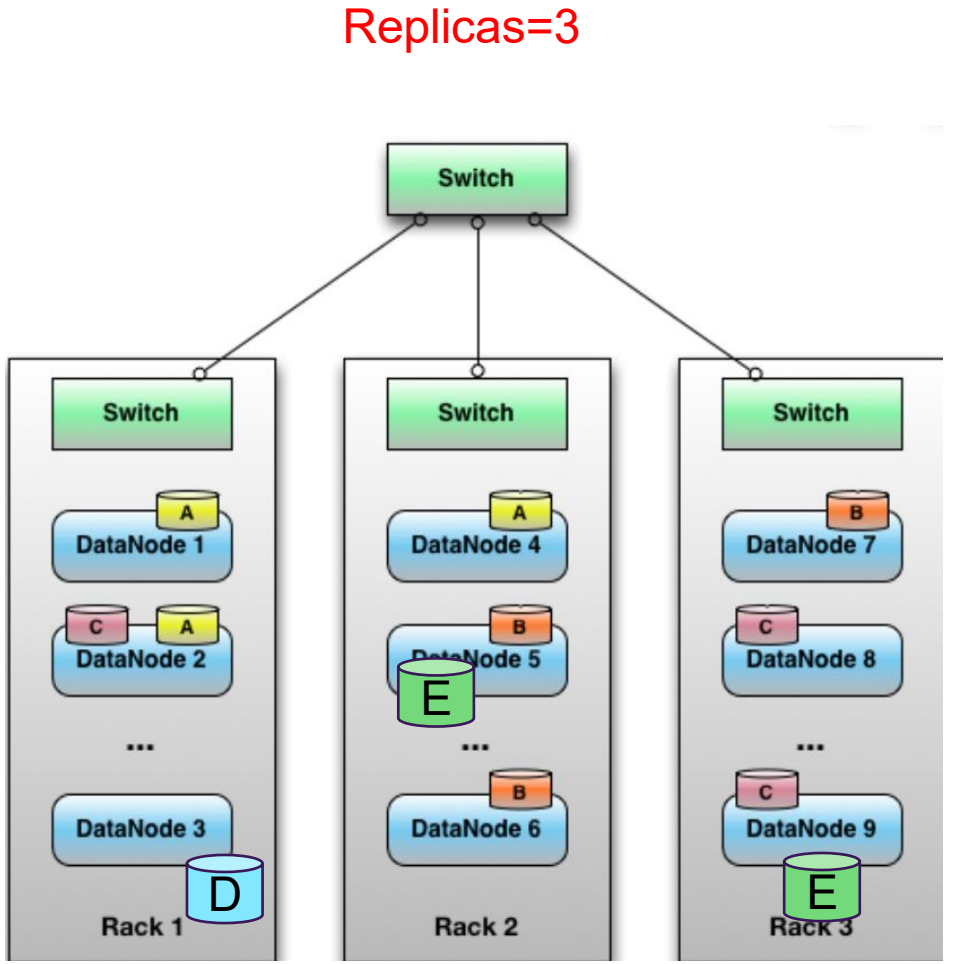
HDFS Replication – Over-replicated

- The NameNode detects that a block has become under- or over-replicated.
  - When a block becomes over-replicated, the NameNode chooses a replica to remove.
    - Firstly prefer not to reduce the number of racks that host replicas,
    - Secondly prefer to remove a replica from the DataNode with the least amount of available disk space.
    - The goal is to balance storage utilization across DataNodes without reducing the block's availability.



HDFS Replication – Under-replicated

- The NameNode detects that a block has become under- or over-replicated.
  - When a block becomes under-replicated, it is put in the replication priority queue.
    - Replication priority will be decided according to the number of replicas
    - E.g. A block with only one replica has the highest priority.
    - A background thread periodically scans the head of the replication queue to decide where to place new replicas.



HDFS vs GFS

Differences

	Hadoop Distributed File System (HDFS)	Google File System (GFS)
Platform	Cross Platform (Linux, Mac, Windows)	Linux
Development	Developed in Java environment	Developed in C,C++ environment
Chunk Size	128 MB (**Hadoop 1.0** – 64 MB and in **Hadoop 2.0** -128 MB)	64 MB
Node	NameNode / DataNodes	Master node & Chunk Server
Log	Editlog	Operational log
Write Operation	No more than one writer at one time	Can have multiple writers to one file at one time.
File Deletion	Deleted files are renamed into particular folder and then it will removed via garbage	Deleted files are not reclaimed immediately and are renamed in hidden name space and it will deleted after three days if it's not in use

https://sensaran.wordpress.com/2015/11/24/gfs-vs-hdfs/

Outline

- Background of Distributed File Systems
  - Big Data, Data Centre Technology, and Storage Hardware
- Distributed File System
  - File System
  - Server/Client System
  - Sun's Network File System (NFS)
- Clustered File System (CFS)
  - Google File System (GFS)
  - Hadoop Distributed File System (HDFS)
  - HDFS Shell Commands

HDFS – Shell Commands

There are two types of shell commands

- User Commands
  - `hdfs dfs` – runs filesystem commands on the HDFS (ls, du, df, etc.)

```
hdfs dfs -ls
hdfs dfs -ls /
hdfs dfs -ls -R /dir
hdfs dfs -du -h /
```

  - `hdfs fsck` – runs a HDFS filesystem checking command
- Administration Commands
  - `hdfs dfsadmin` – runs HDFS administration commands

```
dr_wang1982@instance-3:~$ hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
```

Use “ls” to display files and directories

```
dr_wang1982@instance-3:~$ hdfs dfs -ls /
Found 4 items
drwxr-xr-x   - dr_wang1982 supergroup          0 2019-09-04 05:24 /home
-rw-r--r--   2 dr_wang1982 supergroup         33 2019-08-19 07:35 /input
drwx-wx-wx   - dr_wang1982 supergroup          0 2019-08-25 15:59 /tmp
drwxr-xr-x   - dr_wang1982 supergroup          0 2019-08-25 15:57 /user
```

Use “du” to display disk usage information

```
dr_wang1982@instance-3:~$ hdfs dfs -du -h /
396        /home
33         /input
506.5 M    /tmp
15         /user
```

Use “df” to display disk free information

```
dr_wang1982@instance-3:~$ hdfs dfs -df -h
Filesystem                     Size     Used  Available  Use%
hdfs://sparkmaster:9000  195.9 G  510.5 M    177.4 G    0%
```

HDFS – Shell Commands: COPY

- From Local to HDFS:
  - hdfs dfs -copyFromLocal [path_to_local_file] [path_to_hdfs_location]
  - Example:
    - Make a directory in /home (this is the home directory on HDFS)

```
dr_wang1982@instance-3:~$ hdfs dfs -mkdir /home/data
```

    - Copy a data file on local machine (VM) to the created folder on HDFS

```
dr_wang1982@instance-3:~$ hdfs dfs -copyFromLocal /home/dr_wang1982/infs3208/data/sales.txt /home/data
```

    - Check the copied file

```
dr_wang1982@instance-3:~$ hdfs dfs -ls /home/data
Found 1 items
-rw-r--r--   2 dr_wang1982 supergroup        396 2019-09-04 06:25 /home/data/sales.txt
```

HDFS – Shell Commands: COPY

- From HDFS to Local:
  - hdfs dfs -copyToLocal [path_to_hdfs_location] [path_to_local_file]
  - Example:
    - Copy ”sales.txt” back to a new location on Local machine (VM)

```
dr_wang1982@instance-3:~$ hdfs dfs -copyToLocal /home/data/sales.txt /home/dr_wang1982/infs3208/sales_new.txt
```

    - Check the copied file on Local (VM)

```
dr_wang1982@instance-3:~$ ls /home/dr_wang1982/infs3208/
Lect5.InClassDemo.ipynb              cf_1.ipynb                          pracs
Lecture.4.InClassDemo.ipynb          data                                sales_new.txt
Lecture.6.InClassDemo.MLlib.ipynb    mysql-connector-java-5.1.48.jar     spark-warehouse
```

    - Check the md5sum of ”sales.txt” on HDFS

```
dr_wang1982@instance-3:~$ hadoop fs -cat /home/data/sales.txt|md5sum
674780dc2c75d0334e9df5e037bb77f8  -
```

    - Check the md5sum of “sales_new.txt” on Local (VM)_

```
dr_wang1982@instance-3:~$ md5sum /home/dr_wang1982/infs3208/sales_new.txt
674780dc2c75d0334e9df5e037bb77f8  /home/dr_wang1982/infs3208/sales_new.txt
```

HDFS – Shell Commands DELETE

- To remove a file on HDFS:
  - hdfs dfs –rm [option] [path]
  - Exmaple:
    - Create a copy of sales.txt as sales.txt on HDFS

```
dr_wang1982@instance-3:~$ hdfs dfs -cp /home/data/sales.txt /home/data/sales_copy.txt
```

    - Show the copy

```
dr_wang1982@instance-3:~$ hdfs dfs -ls /home/data
Found 2 items
-rw-r--r--   2 dr_wang1982 supergroup        396 2019-09-04 06:25 /home/data/sales.txt
-rw-r--r--   2 dr_wang1982 supergroup        396 2019-09-04 06:58 /home/data/sales_copy.txt
```

    - Delete the copy version and show the result.

```
dr_wang1982@instance-3:~$ hdfs dfs -rm /home/data/sales_copy.txt
dr_wang1982@instance-3:~$ hdfs dfs -ls /home/data
Found 1 items
-rw-r--r--   2 dr_wang1982 supergroup        396 2019-09-04 06:25 /home/data/sales.txt
```

HDFS – Shell Commands

- You can use fsck to display some file information:
  - hdfs fsck [path] [option]
  - Exmaple:
    - hdfs fsck /home/data/sales

```
Status: HEALTHY
 Total size:    396 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      1 (avg. block size 396 B)
 Minimally replicated blocks:   1 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       1 (100.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              1 (50.0 %)
 Number of data-nodes:          1
 Number of racks:               1
FSCK ended at Wed Sep 04 07:13:24 UTC 2019 in 1 milliseconds
```

HDFS – Shell Commands Administration

- You can use dfsadmin to check status of HDFS status:
  - hdfs dfsadmin [option]
  - Exmaple:
    - hdfs dfsadmin –report

```
dr_wang1982@instance-3:~$ hdfs dfsadmin -report
Configured Capacity: 210304475136 (195.86 GB)
Present Capacity: 190991863808 (177.88 GB)
DFS Remaining: 190456541184 (177.38 GB)
DFS Used: 535322624 (510.52 MB)
DFS Used%: 0.28%
Under replicated blocks: 60
Blocks with corrupt replicas: 0
Missing blocks: 5
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (1):

Name: 10.152.0.5:50010 (sparkmaster)
Hostname: sparkmaster
Decommission Status : Normal
Configured Capacity: 210304475136 (195.86 GB)
DFS Used: 535322624 (510.52 MB)
Non DFS Used: 8558415872 (7.97 GB)
DFS Remaining: 190456541184 (177.38 GB)
DFS Used%: 0.25%
DFS Remaining%: 90.56%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Wed Sep 04 07:15:46 UTC 2019
```

hdfs dfsadmin –printTopology

```
dr_wang1982@instance-3:~$ hdfs dfsadmin -printTopology
Rack: /default-rack
   10.152.0.5:50010 (sparkmaster)
```

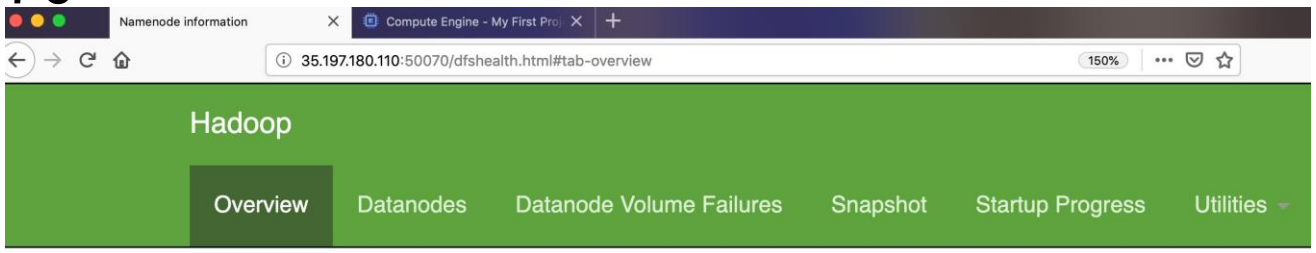
HDFS – Shell Commands Administration

- You can dumpe the NameNode fsimage to XML file:
  - hdfs oiv –i [fsimage file] –o [output file] –p XML
  - Exmaple:

```
<?xml version="1.0"?>
<fsimage><NameSection>
<genstampV1>1000</genstampV1><genstampV2>11(
NameSection>
<INodeSection><lastInodeId>16801</lastInode
mtime><permission>dr_wang1982:supergroup:rw
<inode><id>16386</id><type>FILE</type><name
perferredBlockSize><permission>dr_wang1982:
</blocks>
</inode>
<inode><id>16387</id><type>DIRECTORY</type>
nsquota><dsquota>-1</dsquota></inode>
<inode><id>16398</id><type>DIRECTORY</type>
```

HDFS – Graphical User Interface

- You can visit the GUI using HTTP protocal:
  - http://IP:50070



Overview 'sparkmaster:9000' (active)

Started:	Wed Sep 04 04:51:30 UTC 2019
Version:	2.7.7, rc1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac
Compiled:	2018-07-18T22:47Z by stevel from branch-2.7.7
Cluster ID:	CID-7e921fdd-f52b-4b7a-8052-71aacaa25a92
Block Pool ID:	BP-423617889-10.152.0.5-1566188788273

Summary

When to Use DFSs or Distributed DBs

Distributed file systems (**DFS**) and distributed **databases** are both designed to store vast amounts of data across multiple machines, but they have distinct characteristics and serve different purposes.

Choosing between them depends on the specific needs of a project or application.

- Data Type:
  - DFS: vast amounts of unstructured data (e.g., videos, images) → HDFS or Amazon S3.
  - Distributed DB: require structured data and transactional capabilities → distributed databases.
- Write and Read Patterns:
  - DFS: suitable for write-once, read-many workloads. If your primary operation is appending data and not updating existing data frequently, DFS is a good choice.
  - Distributed DB: More suited for scenarios where frequent updates and random reads are required.
- Data Processing (Big Data Analytical Tool Integration):
  - DFS: use batch processing frameworks (e.g. Hadoop MapReduce) to process and analyze data → DFSs.
  - Distributed DB: need real-time querying capabilities and indexing → a distributed DB.
- ACID or BASE properties: → a distributed DB.
- Cost and Flexibility: → DFSs.

References

1.HDFS Tutorial – A Complete Hadoop HDFS Overview. https://data-flair.training/blogs/hadoop-hdfs-tutorial/

2.HDFS Overview. https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm

3.Machine Learning Library (Mllib) Guide, https://spark.apache.org/docs/latest/ml-guide.html

4.https://www.bmc.com/blogs/using-logistic-regression-scala-spark/

5.http://www.cse.chalmers.se/~tsigas/Courses/DCDSeminar/Files/afs_report.pdf

6.https://courses.cs.washington.edu/courses/cse490h/11wi/CSE490H_files/gfs.pdf

7.https://www.slideshare.net/YuvalCarmel/gfs-vs-hdfs

8.https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/FileSystemShell.html