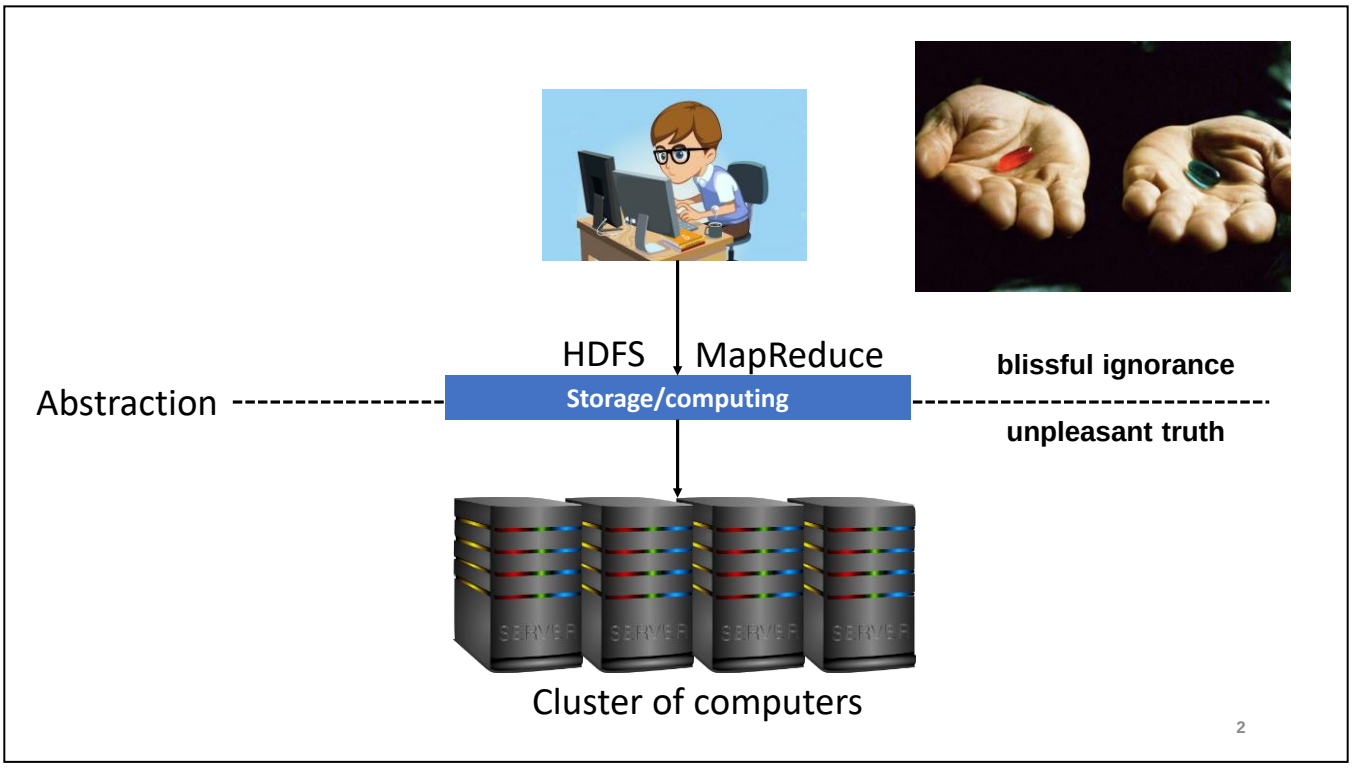
Data-Intensive Distributed Computing

431/451/631/651 (Fall 2021)

Part 2: MapReduce Algorithm Design (2/3)

Ali Abedi

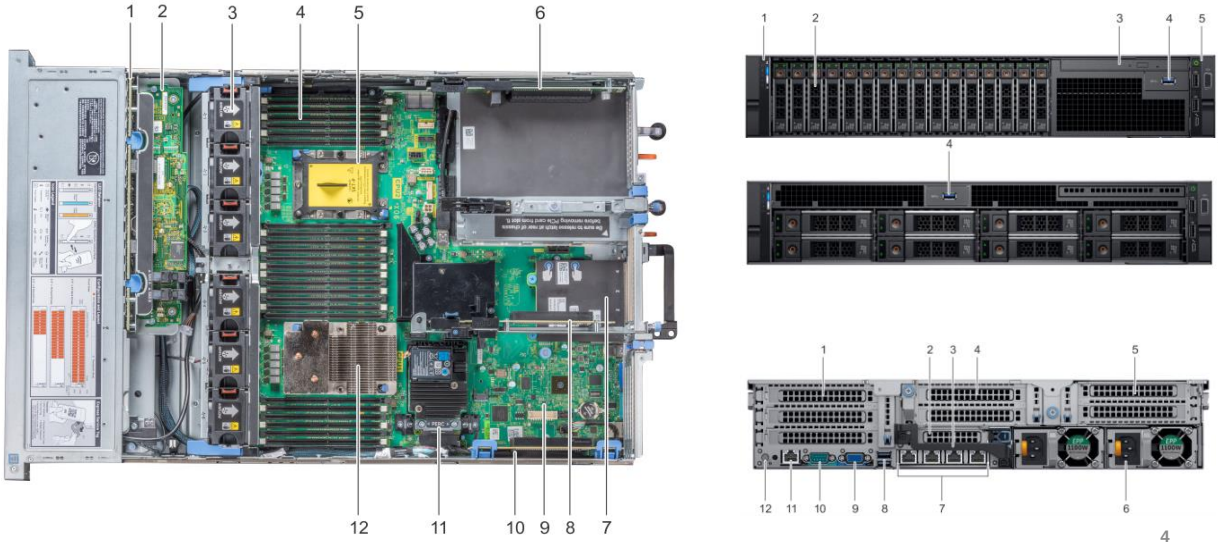
These slides are available at https://www.student.cs.uwaterloo.ca/~cs451/



Although we argued about having an abstraction layer to hide the complexities of underlying infrastructure, today we want to have a quick look at the architecture of datacenters. This will help us later to understand the performance trade offs of different algorithms. It also makes us appreciate these systems more ☺


A quick review of data center architecture
3

The anatomy of a server



Left: Top view of a server

Right: the two top figures are the front of the server with two storage configurations: 1)16 2.5 inch drives 2) 8 3.5 inch drivers

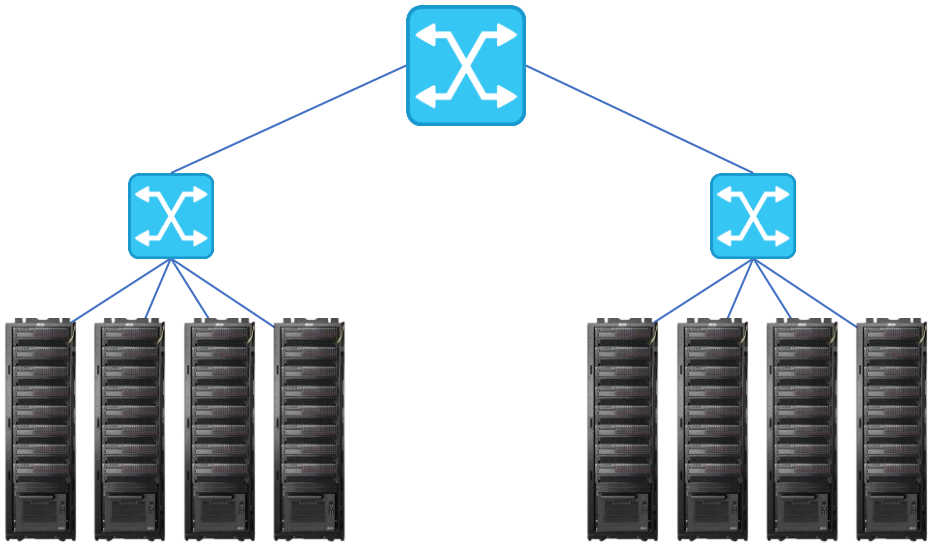
Right: bottom is the back of the server. We can see network interfaces (7)

The anatomy of a server rack

5

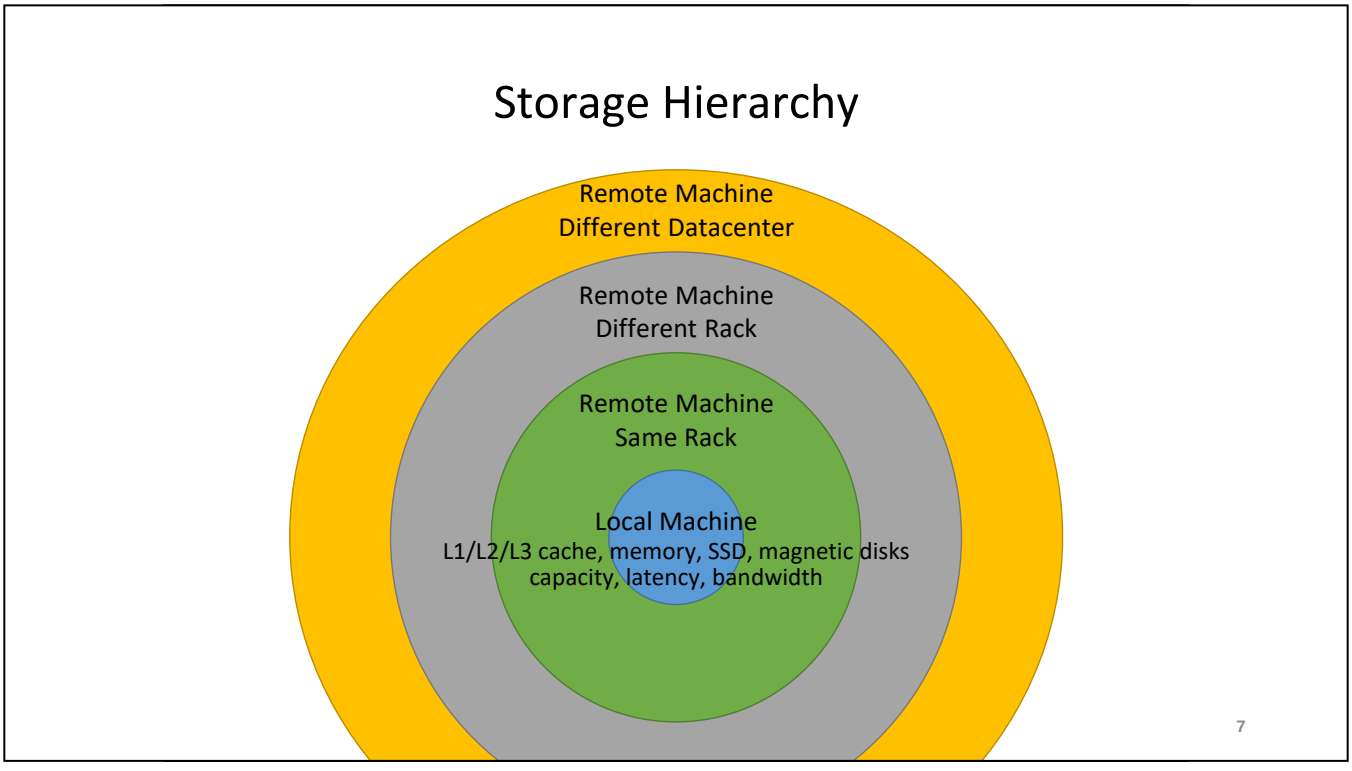
We put multiple servers in a server rack. There is a network switch that connects the servers in a rack. This switch also connects the rack to other racks.

The anatomy of a data center



6

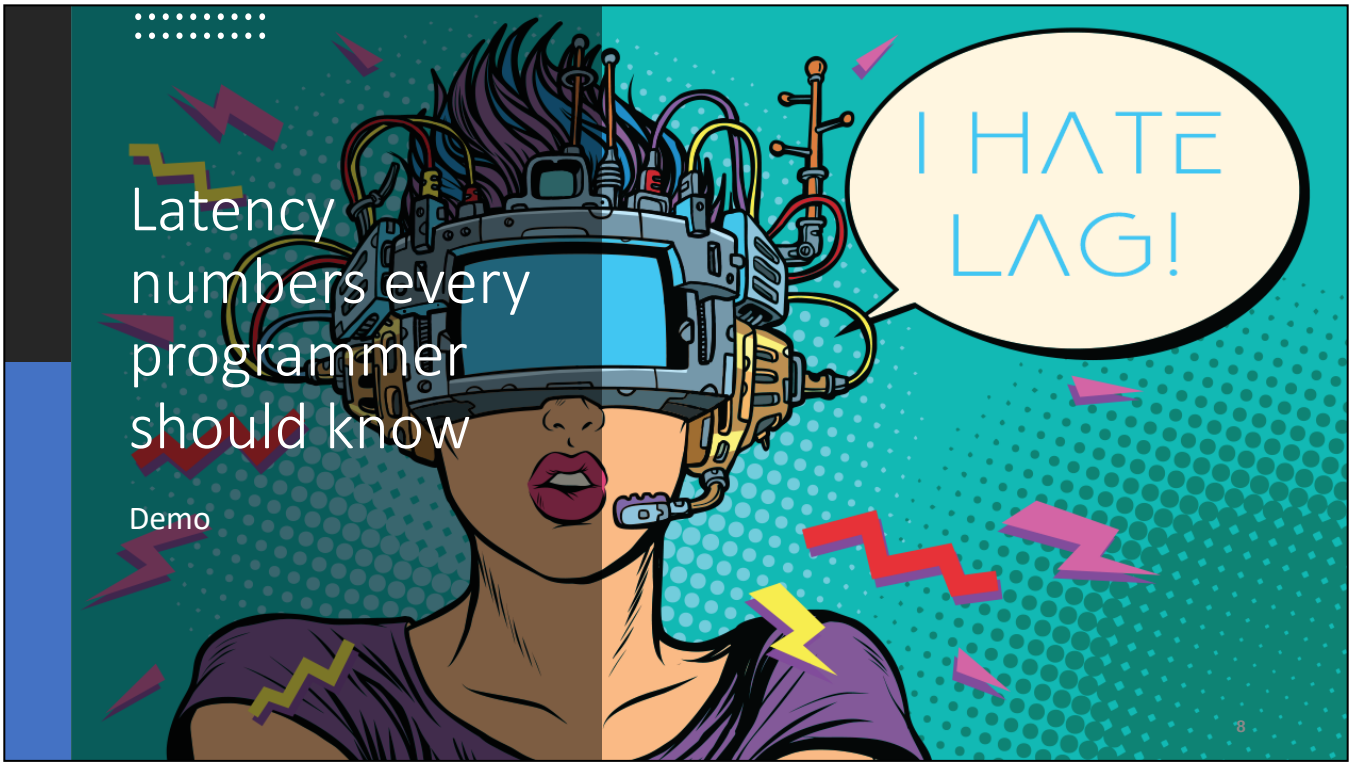
Clusters of racks of servers build a data center. This is a very simplistic view of a data center.



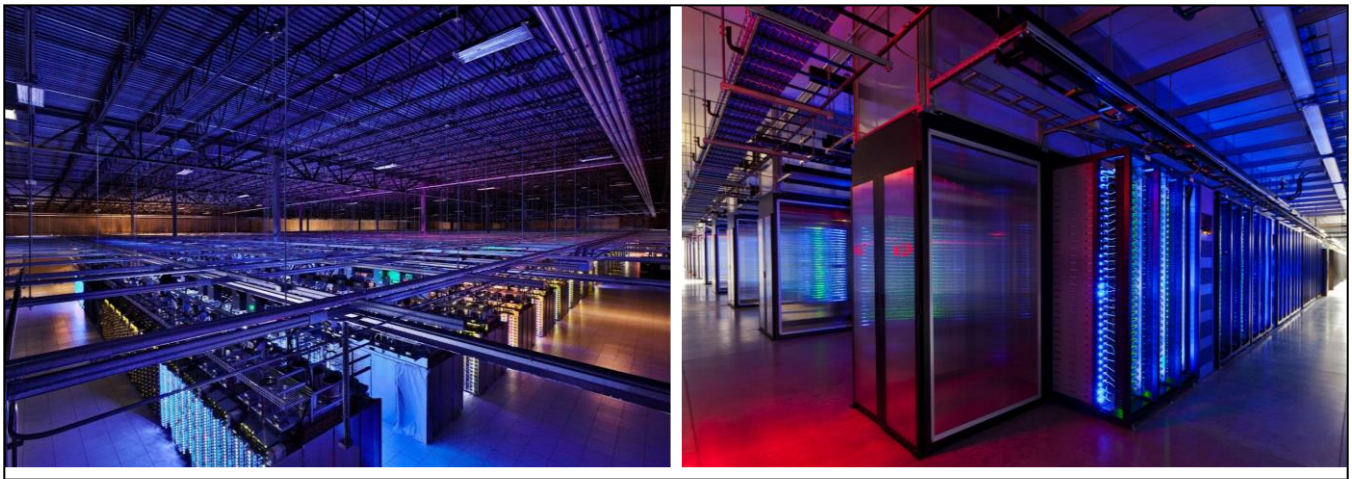
Capacity, latency, and bandwidth for reading data change depending on where the data is.

The lowest latency and highest bandwidth is achieved when the data we need is on our local server.

We can increase capacity by utilizing other servers but at the cost of higher latency and lower bandwidth.



https://colin-scott.github.io/personal_website/research/interactive_latency.html

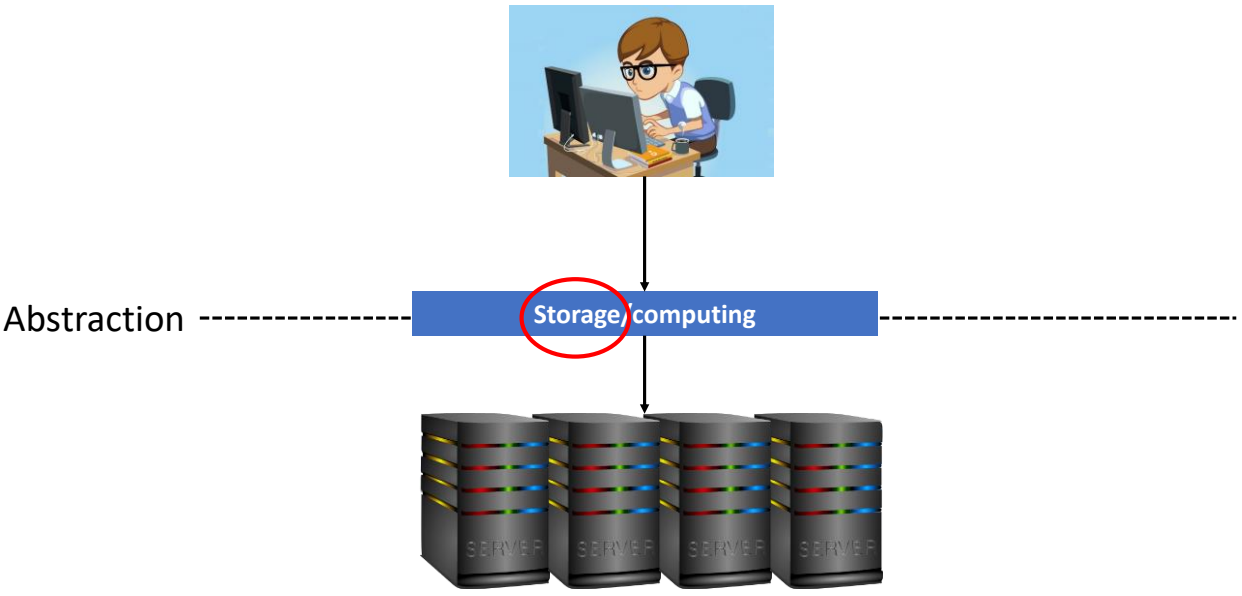


The anatomy of a data center

Google’s data center video

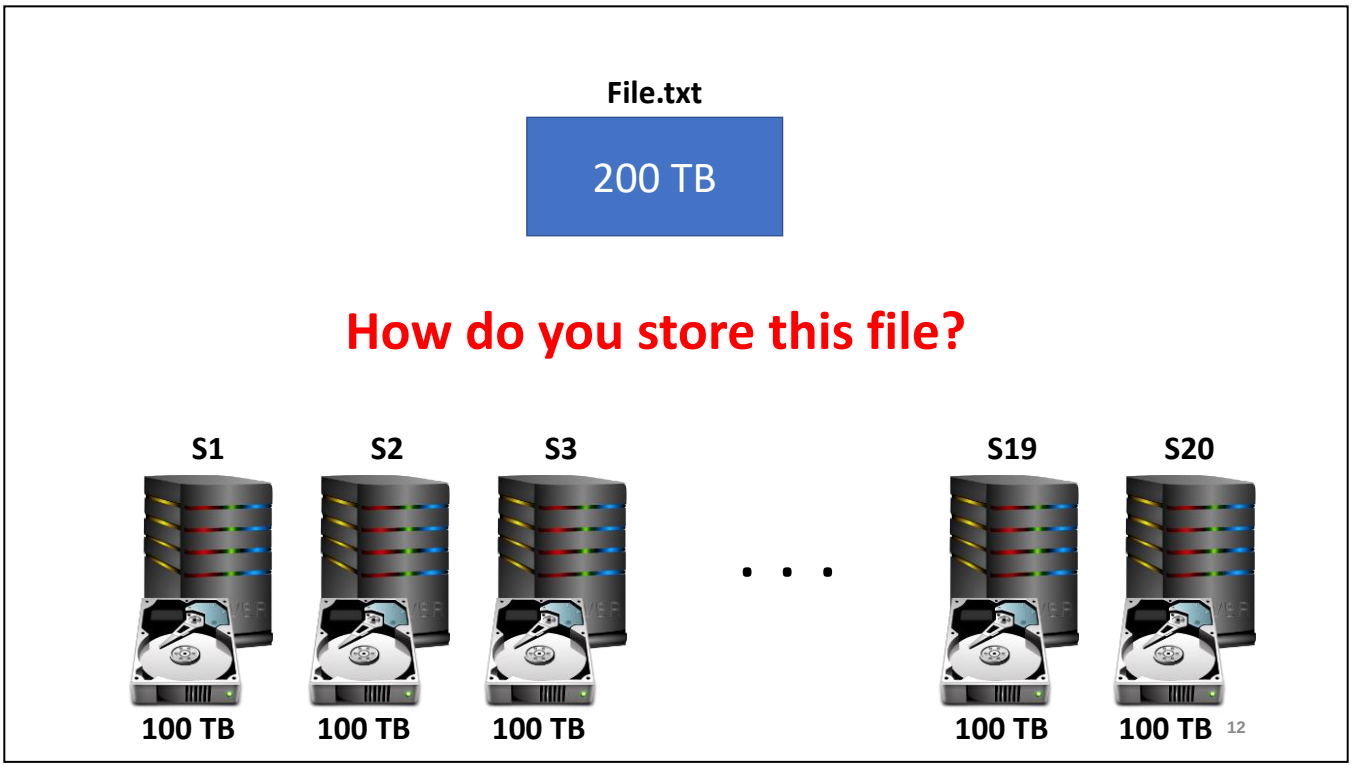
9

https://youtu.be/XZmGGAbHqa0

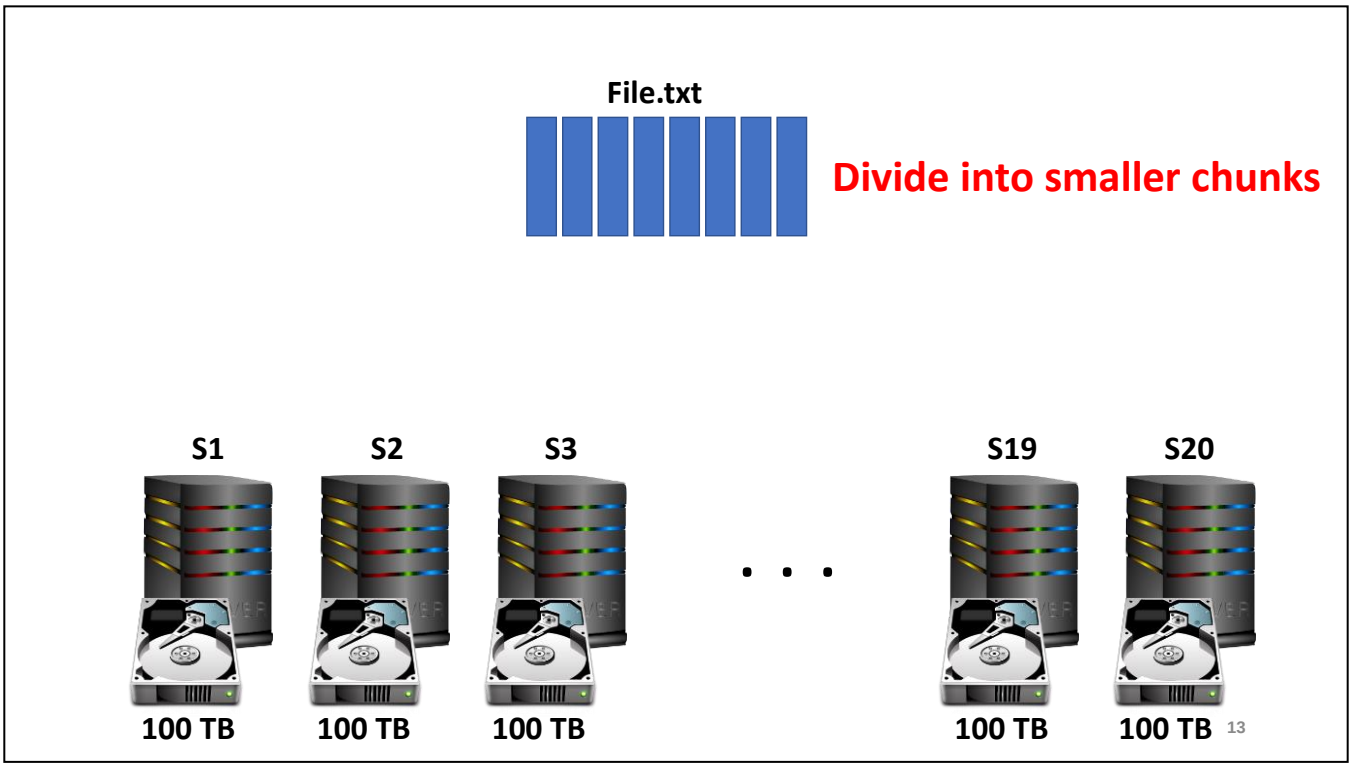

Abstraction
Storage/computing
Cluster of computers

Distributed File System

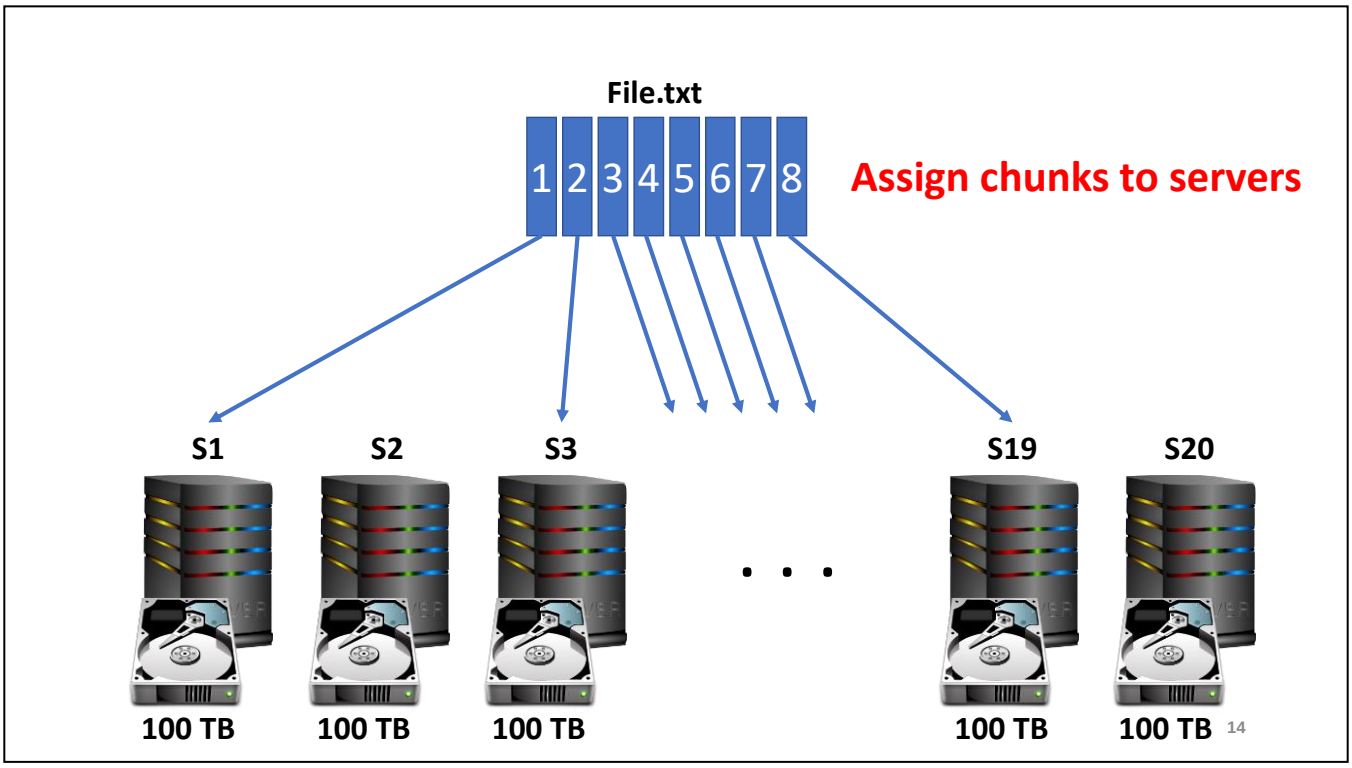
How can we store a large file on a distributed system?



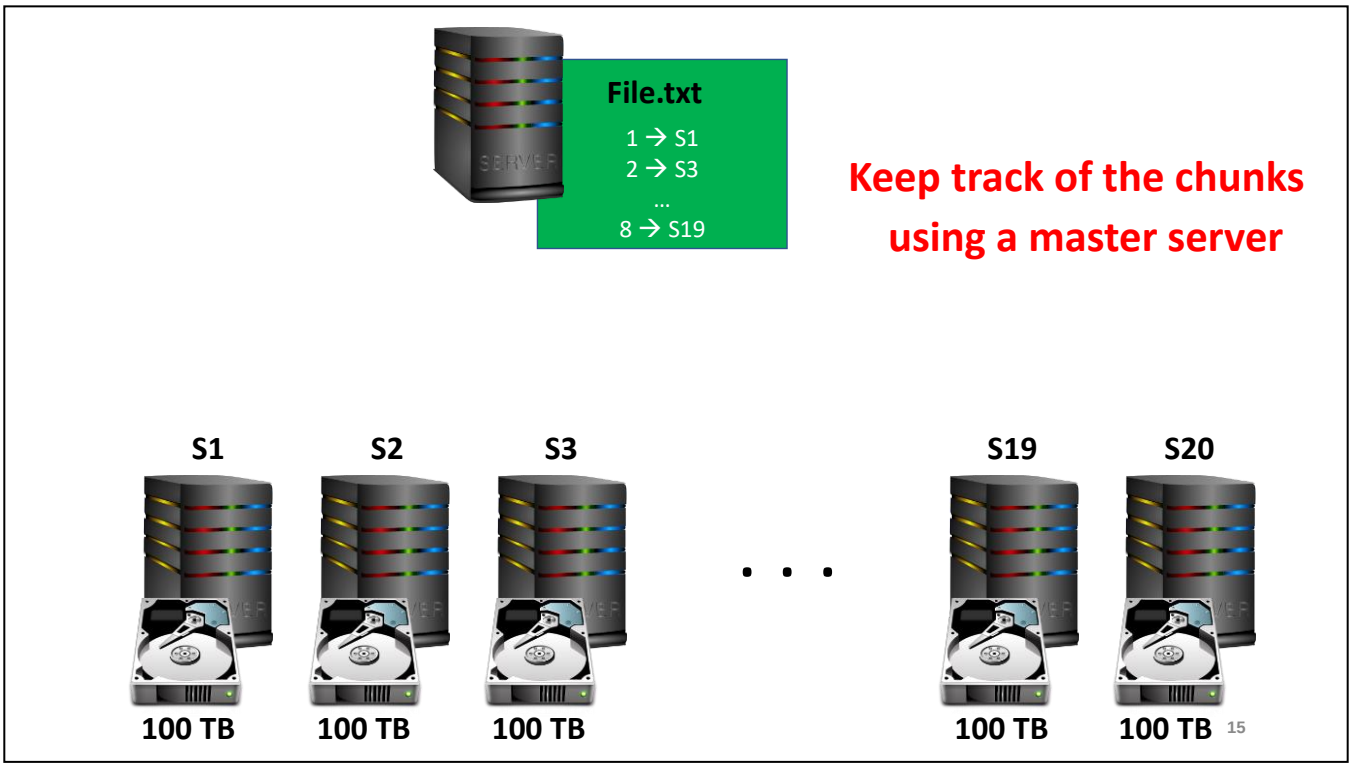
Assume that we have 20 identical networked servers each with 100 TB of disk space. How would you store a file on these server? This is the fundamental question in distributed file systems.



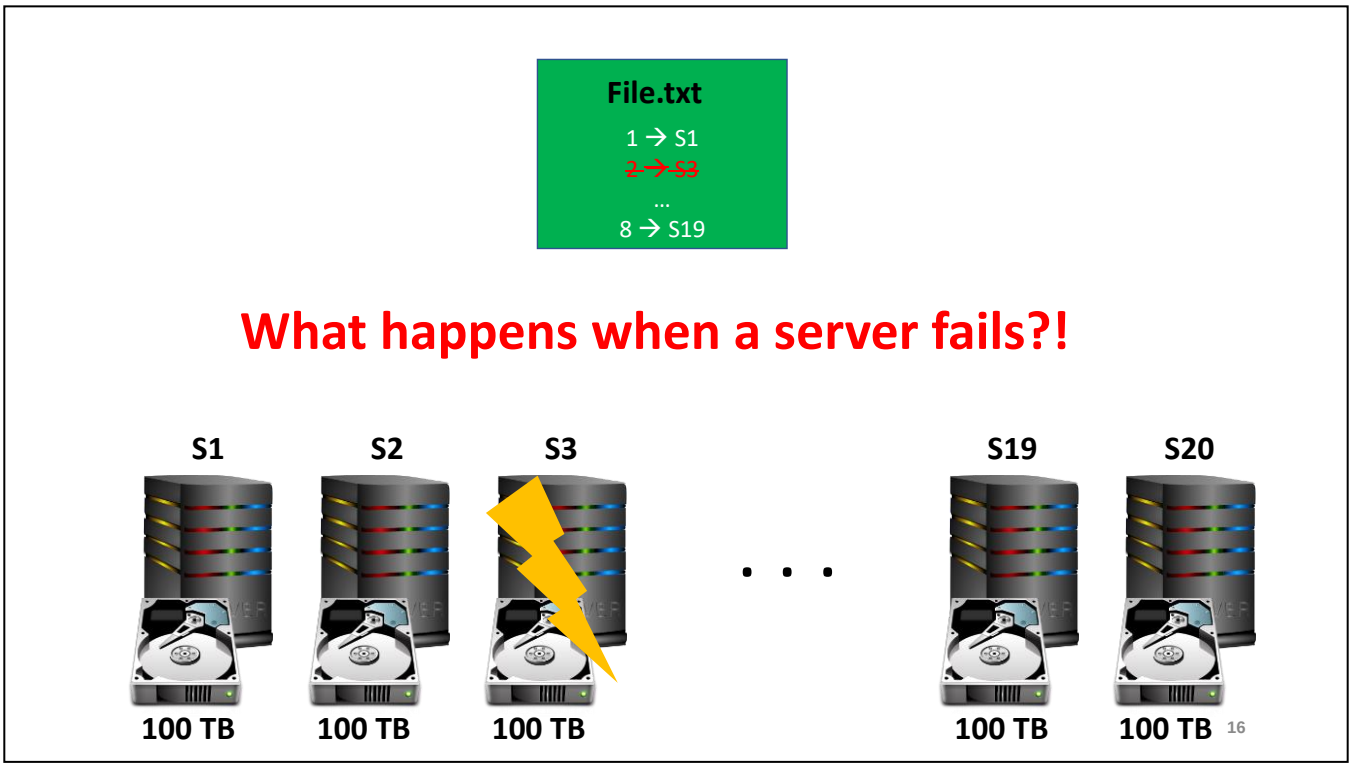
We can split the file into smaller chunks.



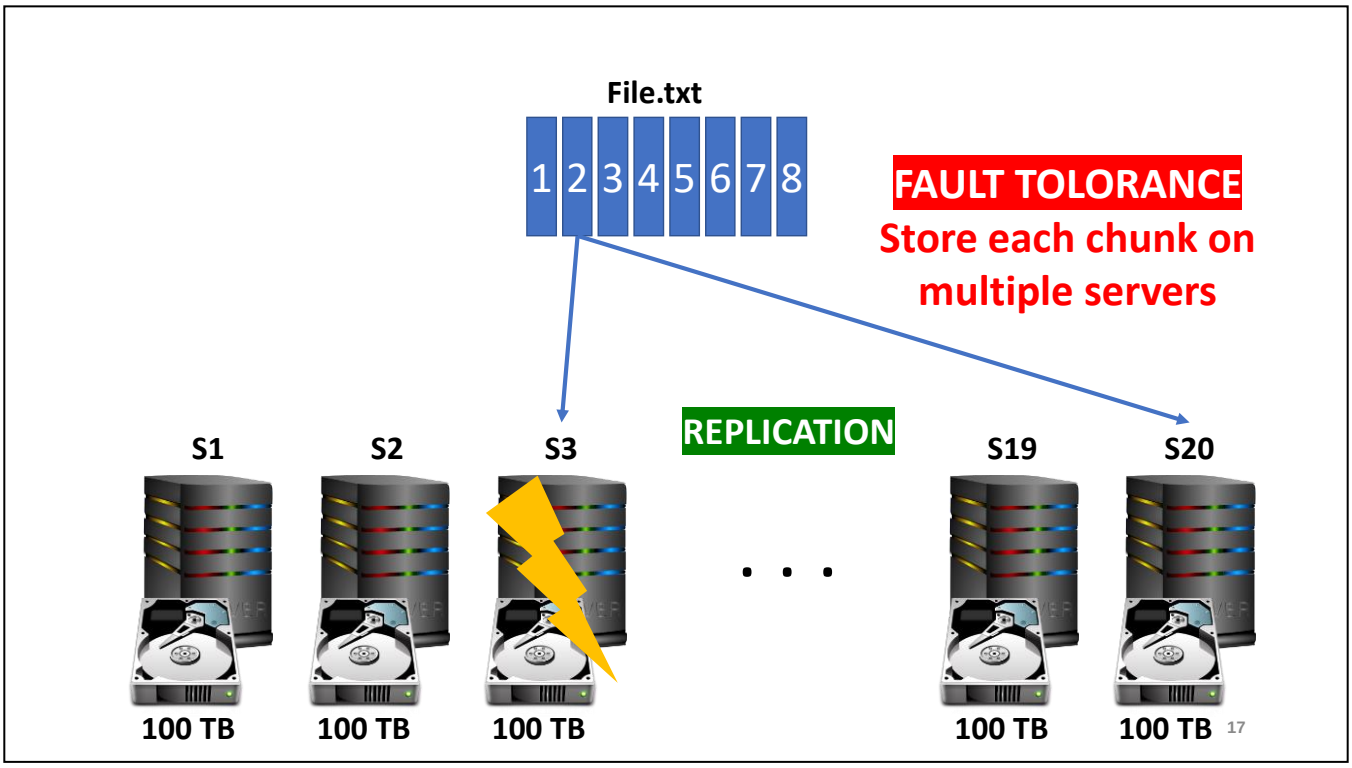
And assign the chunks (e.g., randomly) to the servers.



We need to track where each chunk is stored so that we can retrieve the file.



If a server that contains one of the chunks fails, the files become corrupted. Since failure rate is high on commodity servers, we need to figure out a solution.



If each chunk is stored on multiple server, if a server fails there is a backup. The number of copies determines how much resilience we want.

From our made-up distributed file system to a real one

Hadoop Distributed File System (HDFS)

Adapted from form Erik Jonsson (UT Dallas)

Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Provides very high aggregate bandwidth

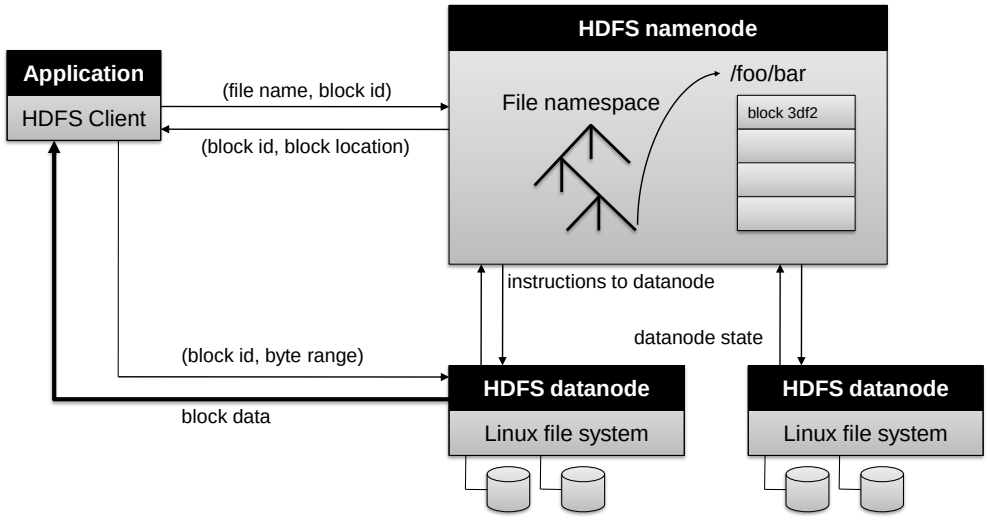
Distributed File System

- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

21

HDFS is not like a typical file system you use on Windows or Linux. It was specifically designed for Hadoop. It cannot perform some of the typical operations that other file systems can do like random write. Instead it is optimized for large sequential reads and append only writes.

HDFS Architecture



Adapted from (Ghemawat et al., SOSP 2003)

22

Note that the namenode is relatively lightweight, it's just storing where the data is located on datanodes not the actual data.

May still have a redundant namenode in the background if the primary one fails

HDFS client gets data information from namenode and then interacts with datanodes to get that data

Note that namenode has to communicate with datanodes to ensure consistency and redundancy of data (e.g., if a new clone of the data needs to be created)

Functions of a NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Current Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replicas

Heartbeats

- DataNodes send hearbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

HDFS Demo

Google File System (GFS)

Terminology differences:

GFS master = Hadoop namenode

GFS chunkservers = Hadoop datanodes

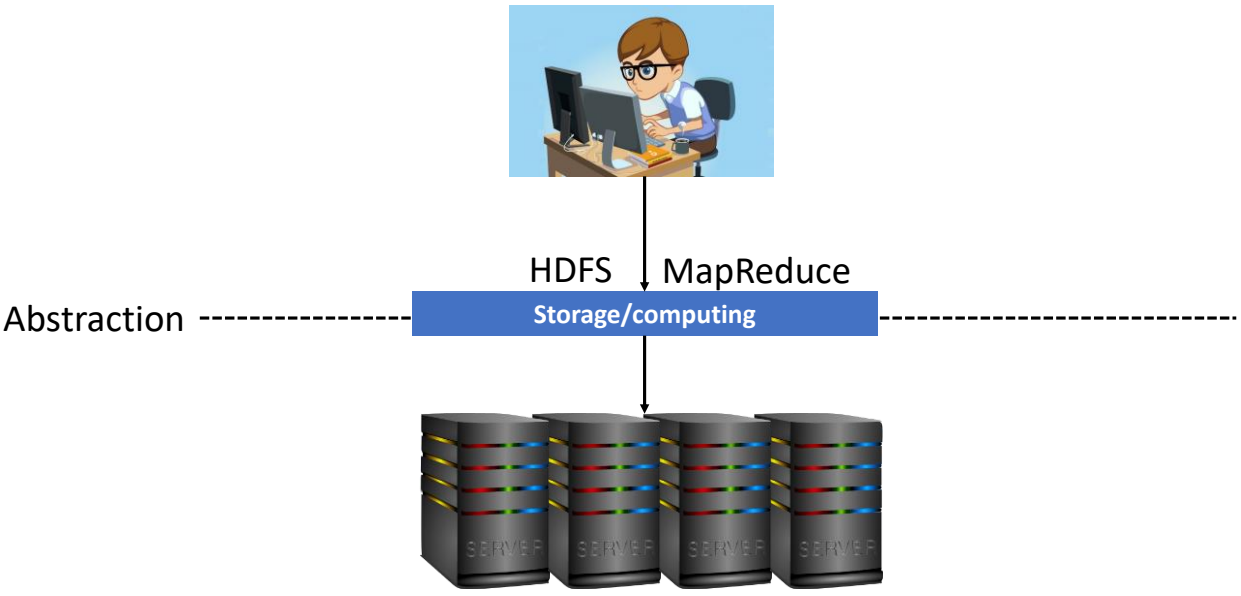
Implementation differences:

Different consistency model for file appends

Implementation language

Performance

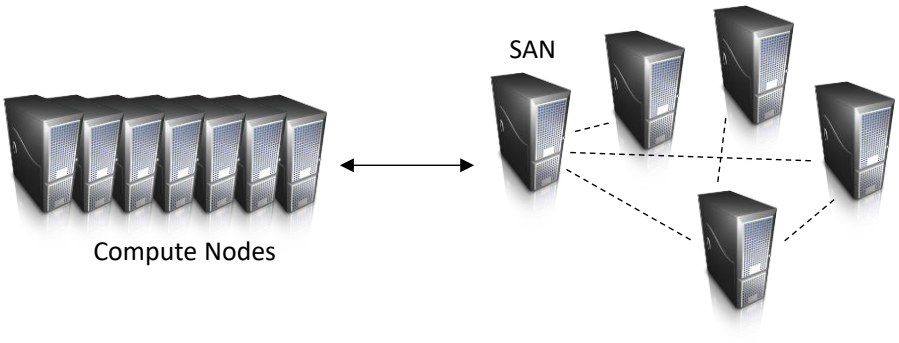
"THAT'S your Ark for the Big Data flood? Noah, you will need a lot more storage space!"


HDFS
MapReduce
Abstraction
Storage/computing
Cluster of computers


G14
G14
Hadoop Cluster Architecture
33

How do we get data to the workers?

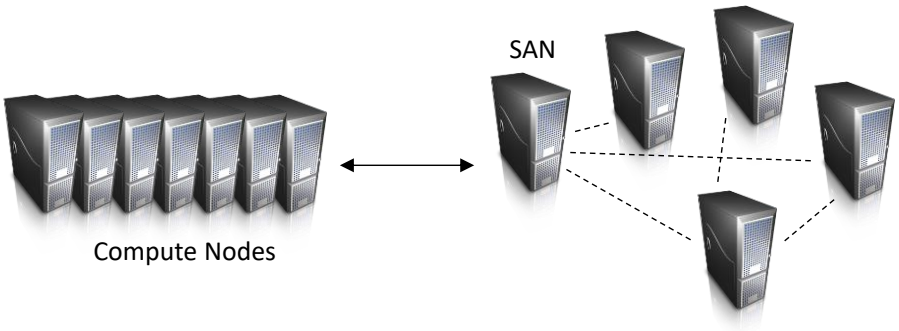
Let's consider a typical supercomputer…



34

SAN: Storage Area Network

Compute-Intensive vs. Data-Intensive



Why does this make sense for compute-intensive tasks?
What's the issue for data-intensive tasks?

35

This makes sense for compute-intensive tasks as the computations (for some chunk of data) are likely to take a long while even on such sophisticated hardware, so the communication costs are greatly outweighed by the computation costs. For data-intensive tasks, the computations (for some chunk of data) aren't likely to take nearly as long, so the computation costs are greatly outweighed by the communication costs. Likely to experience latency and bottleneck even with high speed transfer.

What's the solution?

Don't move data to workers… move workers to the data!

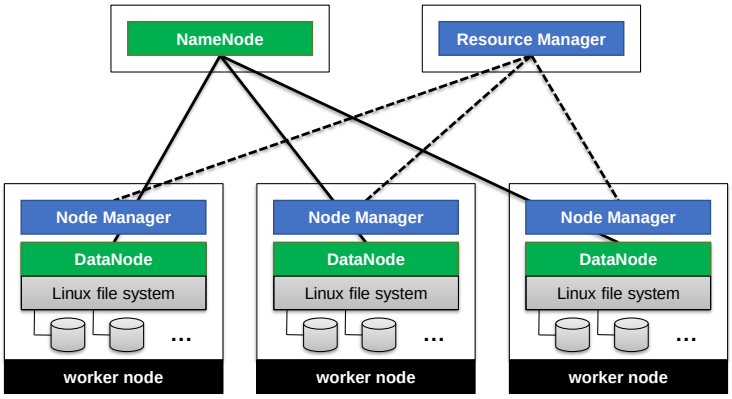
Key idea: co-locate storage and compute

Start up worker on nodes that hold the data

36

If a server is responsible for both data storage and processing, Hadoop can do a lot of optimization. For example, when assigning mapreduce tasks to servers, Hadoop considers which servers contain what part of the file locally to minimize copy over network. If all of the data can be process locally where it is stored there will be no need to move the data.

Putting everything together…

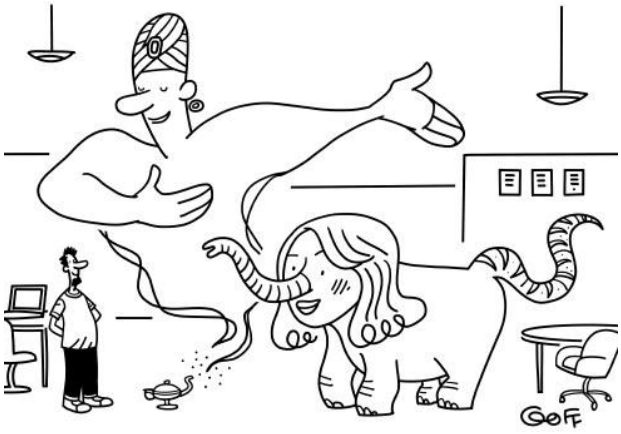


37

This figure shows how computation and storage is co-located on a Hadoop cluster.

Node manager manages running tasks on a node (e.g., if we have spare resources, do the next job assigned to us)

Resource manager is responsible for managing available resources in the cluster



"Meet Scarledoopython - you did ask for Scarlett Johansson, Hadoop, and Python?"

KDnuggets Cartoon