

Hadoop: Basic Concepts

Learning Outcomes

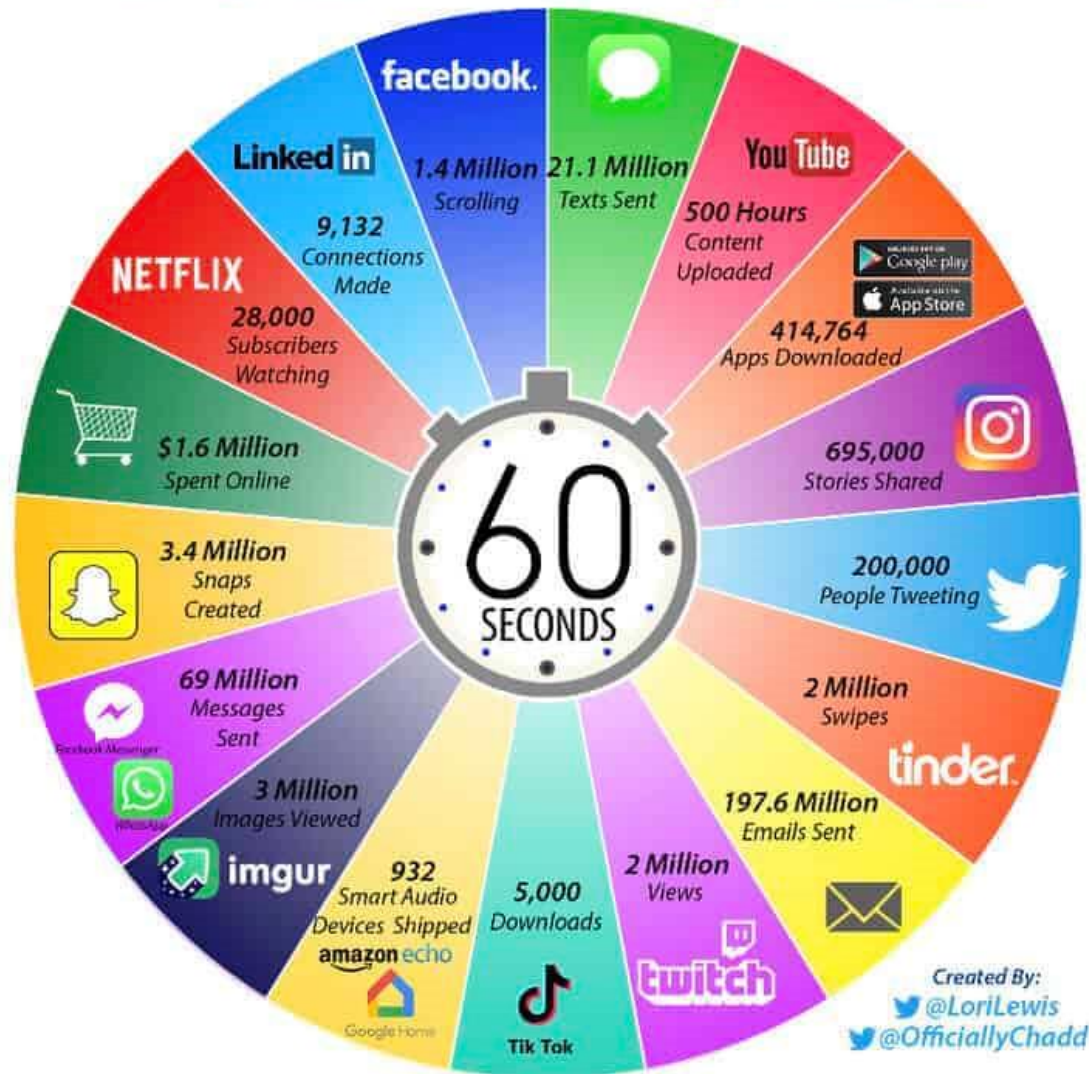
At the end of this topic, You should be able to

- Comprehensive understanding of Hadoop framework and its principles in distributed computing.
- Proficiency in practical use of Hadoop's HDFS and MapReduce for data processing.

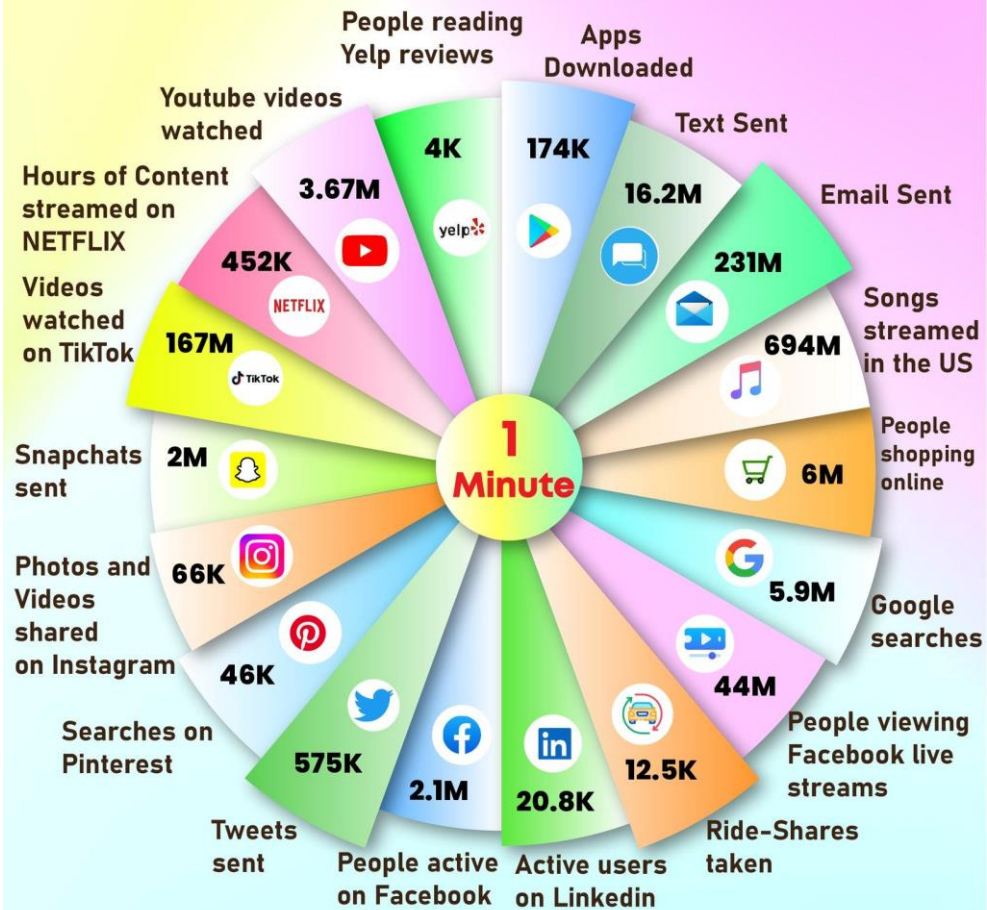
Key Terms You Must Be Able To Use

- Hadoop
- HDFS
- MapReduce
- Five steps MapReduce programming model


2021 *This Is What Happens In An Internet Minute*



A Minute on the Internet in 2022



Source: LocaliQ

 RankingRoyals

THE INTERNET IN 2023 EVERY MINUTE



Created by: eDiscovery Today & LTMG

Data Structures

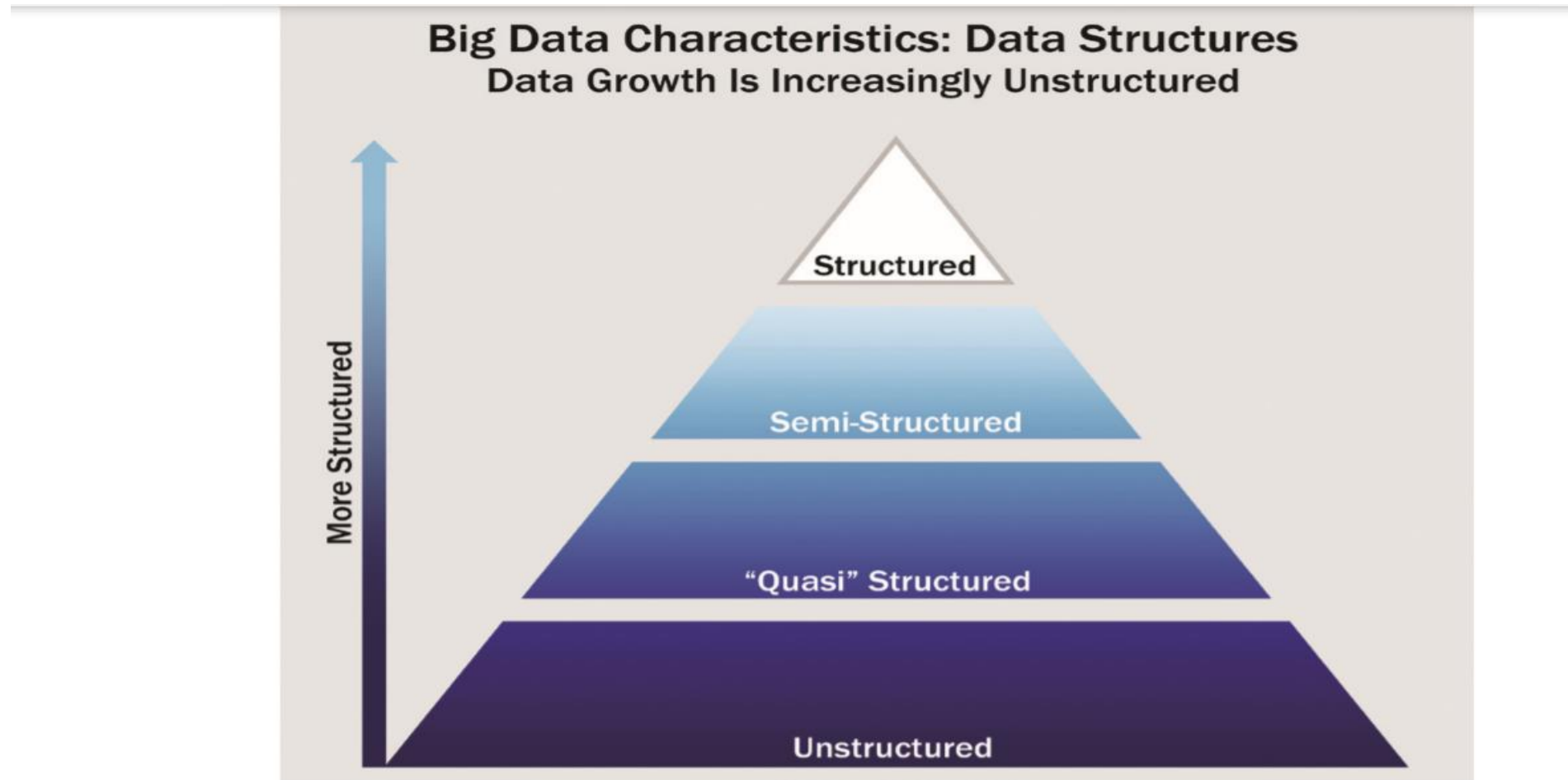


FIGURE 1-3 Big Data Growth is increasingly unstructured

Structured Data

- Data containing a defined data type, format, and structure (that is, transaction data, traditional RDBMS, CSV files, and even simple spreadsheets).

SUMMER FOOD SERVICE PROGRAM 1]				
(Data as of August 01, 2011)				
Fiscal Year	Number of Sites	Peak (July) Participation	Meals Served	Total Federal Expenditures 2]
	-----Thousands-----		--Mil.--	---Million \$---
1969	1.2	99	2.2	0.3
1970	1.9	227	8.2	1.8
1971	3.2	569	29.0	8.2
1972	6.5	1,080	73.5	21.9
1973	11.2	1,437	65.4	26.6
1974	10.6	1,403	63.6	33.6
1975	12.0	1,785	84.3	50.3
1976	16.0	2,453	104.8	73.4
TQ 3]	22.4	3,455	198.0	88.9
1977	23.7	2,791	170.4	114.4
1978	22.4	2,333	120.3	100.3
1979	23.0	2,126	121.8	108.6
1980	21.6	1,922	108.2	110.1
1981	20.6	1,726	90.3	105.9
1982	14.4	1,397	68.2	87.1
1983	14.9	1,401	71.3	93.4
1984	15.1	1,422	73.8	96.2
1985	16.0	1,462	77.2	111.5
1986	16.1	1,509	77.1	114.7
1987	16.9	1,560	79.9	129.3
1988	17.2	1,577	80.3	133.3
1989	18.5	1,652	86.0	143.8
1990	19.2	1,692	91.2	163.3

FIGURE 1-4 Example of structured data

Semi-structured data

- Textual data files with a discernible pattern that enables parsing (such as Extensible Markup Language [XML] data files that are self-describing and defined by an XML schema).

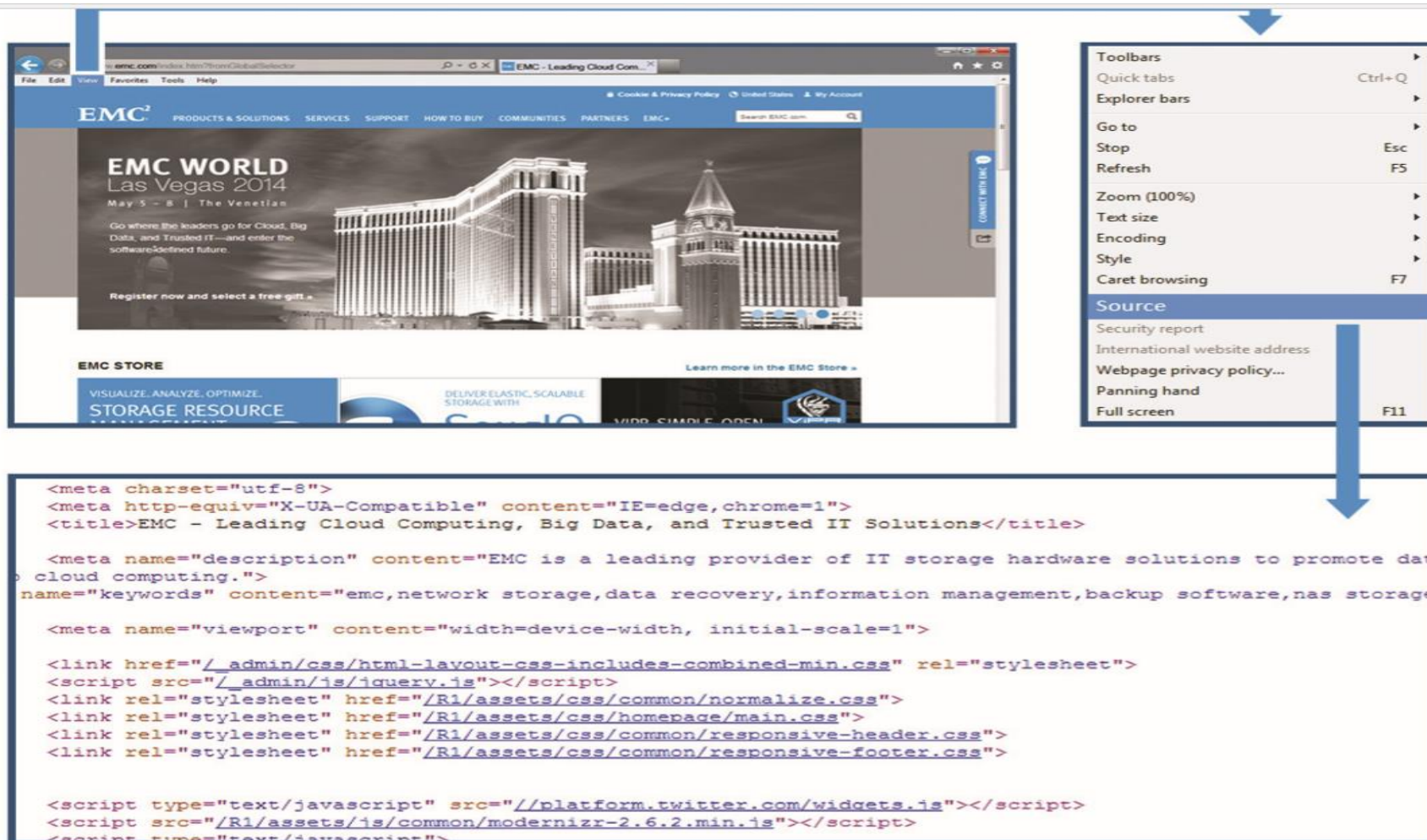


FIGURE 1-5 Example of semi-structured data

Quasi-structured data

- Textual data with erratic data formats that can be formatted with effort, tools, and time (for instance, web clickstream data that may contain inconsistencies in data values and formats).

.aspx, that displays the page shown as (2) in Figure 1-6. Arriving at this site, the user may decide to click to learn more about the process of becoming certified in data science. The user chooses a link toward the top of the page on Certifications, bringing the user to a new URL: https://education.emc.com/guest/certification/framework/stf/data_science.aspx, which is (3) in Figure 1-6.

Visiting these three websites adds three URLs to the log files monitoring the user's computer or network use. These three URLs are:

<https://www.google.com/#q=EMC+data+science>

https://education.emc.com/guest/campaign/data_science.aspx

https://education.emc.com/guest/certification/framework/stf/data_science.aspx



FIGURE 1-6 Example of EMC Data Science search results

Unstructured data

- Data that has no inherent structure, which may include text documents, PDFs, images, and video.

➤ Machine Generated

- Satellite images
- Scientific data
- Photographs and video
- Radar or sonar data

➤ Human Generated

- Word, PDF, Text
- Social media data (Facebook, Twitter, LinkedIn)
- Mobile data (text messages)
- website contents (blogs, Instagram)



Get smaller to understand how ants work and what they are capable of.



Use this knowledge to control thousands of ants and do amazing things!

Hadoop Components

What is Hadoop

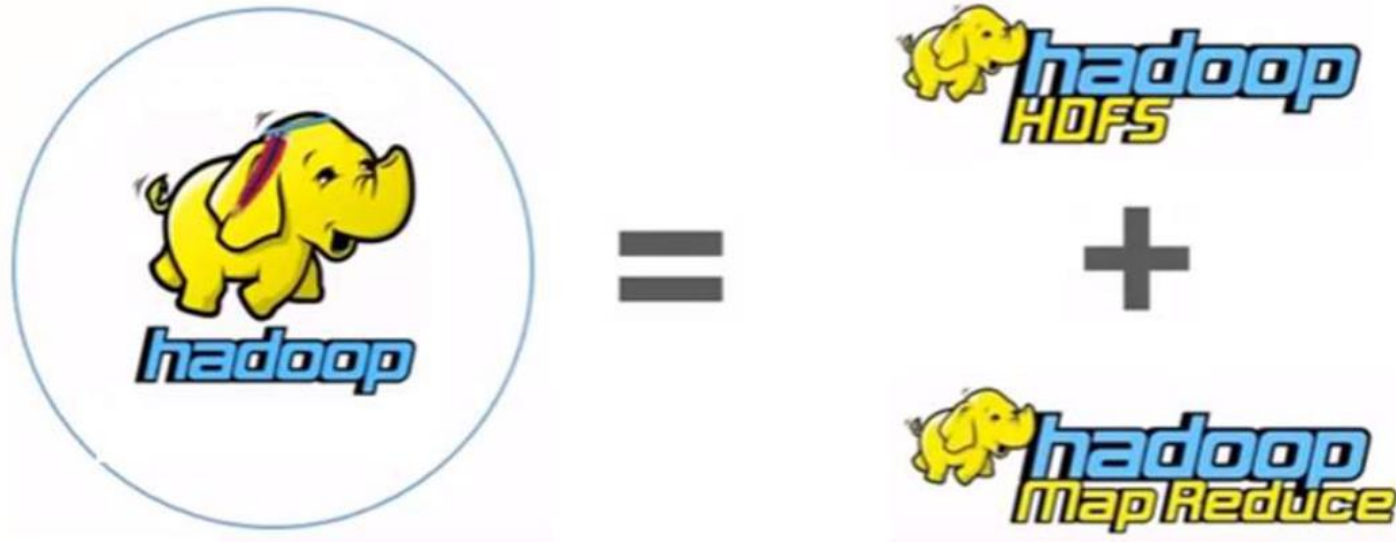
- Hadoop is a software framework for *distributed processing* of *large datasets* across *large clusters* of computers
- • *Large datasets* → Terabytes or petabytes of data
- • *Large clusters* → hundreds or thousands of nodes
- • Hadoop is open-source implementation for Google *MapReduce*
- • Hadoop is based on a simple *programming model* called *MapReduce*
- • Hadoop is based on a simple *data model*, *any data will fit*

When Hadoop?

- When you need to process large volumes of unstructured data.
- When parallel processing is feasible for your task.
- When batch job execution is permissible.
- When you have access to abundant cost-effective hardware.

Hadoop Core Components

Hadoop Distributed File System (HDFS) + MapReduce



Hadoop Components

- **Hadoop consists of two core components**
 - The Hadoop Distributed File System (HDFS)
 - MapReduce Software Framework
- **There are many other projects based around core Hadoop**
 - Often referred to as the 'Hadoop Ecosystem'
 - Pig, Hive, HBase, Flume, Oozie, Sqoop, etc
 - Many are discussed later in the course

Hadoop Components

- **A set of machines running HDFS and MapReduce is known as a Hadoop Cluster**
 - Individual machines are known as nodes
 - A cluster can have as few as one node, as many as several thousands
 - More nodes = better performance!

Hadoop Components: HDFS

- **HDFS, the Hadoop Distributed File System, is responsible for storing data on the cluster**
- **Data files are split into blocks and distributed across multiple nodes in the cluster**
- **Each block is replicated multiple times**
 - Default is to replicate each block three times
 - Replicas are stored on different nodes
 - This ensures both reliability and availability

Hadoop Components: MapReduce

- **MapReduce is the system used to process data in the Hadoop cluster**
- **Consists of two phases: Map, and then Reduce**
- **Each Map task operates on a discrete portion of the overall dataset**
 - Typically one HDFS data block
- **After all Maps are complete, the MapReduce system distributes the intermediate data to nodes which perform the Reduce phase**
 - Much more on this later!

File system in operating system

A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.

[Source- click here](#)

HDFS Basic Concepts

- **HDFS is a filesystem written in Java**
 - Based on Google's GFS
- **Sits on top of a native filesystem**
 - ext3, xfs etc
- **Provides redundant storage for massive amounts of data**
 - Using cheap, unreliable computers

HDFS Basic Concepts (cont'd)

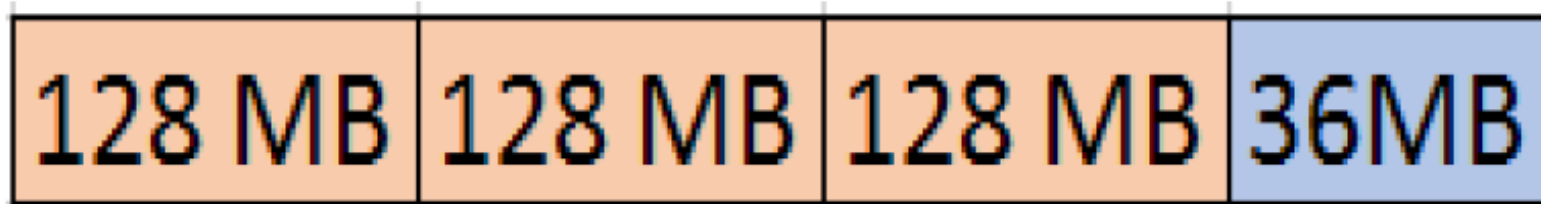
- **HDFS performs best with a 'modest' number of large files**
 - Millions, rather than billions, of files
 - Each file typically 100Mb or more
- **Files in HDFS are 'write once'**
 - No random writes to files are allowed
- **HDFS is optimized for large, streaming reads of files**
 - Rather than random reads

How Files Are Stored

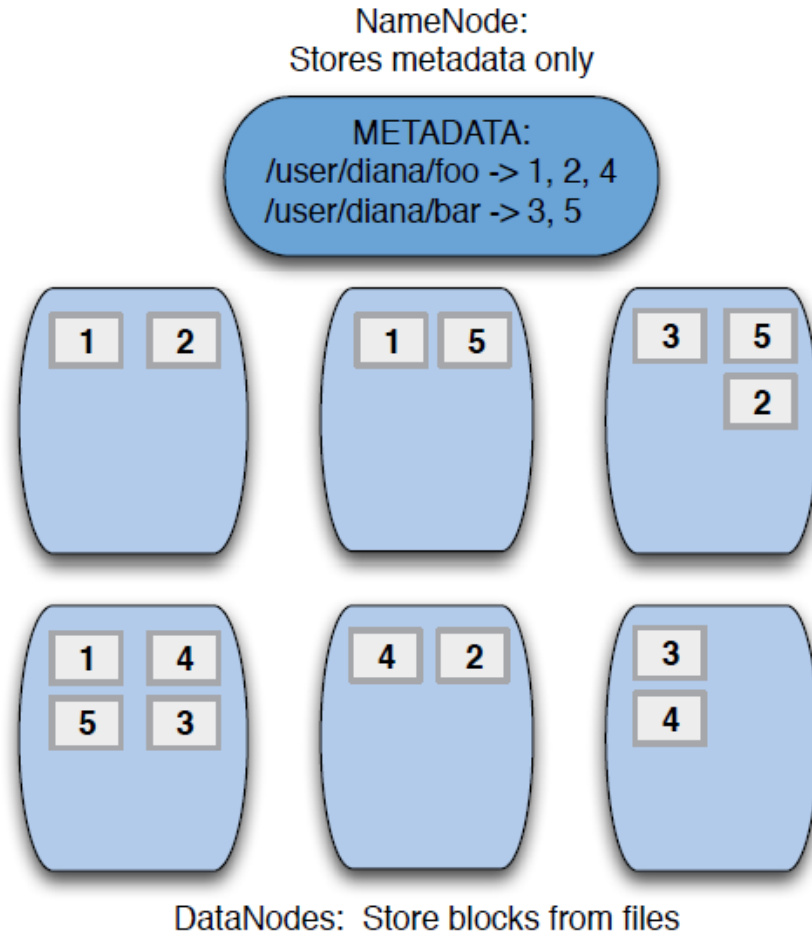
- **Files are split into blocks**
- **Data is distributed across many machines at load time**
 - Different blocks from the same file will be stored on different machines
 - This provides for efficient MapReduce processing (see later)
- **Blocks are replicated across multiple machines, known as DataNodes**
 - Default replication is three-fold
 - i.e., each block exists on three different machines
- **A master node called the NameNode keeps track of which blocks make up a file, and where those blocks are located**
 - Known as the metadata

File Storage in HDFS

- Split into multiple blocks/chunks and stored into different machines.
- Blocks – 64MB size (default), 128MB (recommended).
- Replication – fault tolerance and availability, it is configurable and it can be modified.
- No storage space wasted. E.g. 420MB file stored as

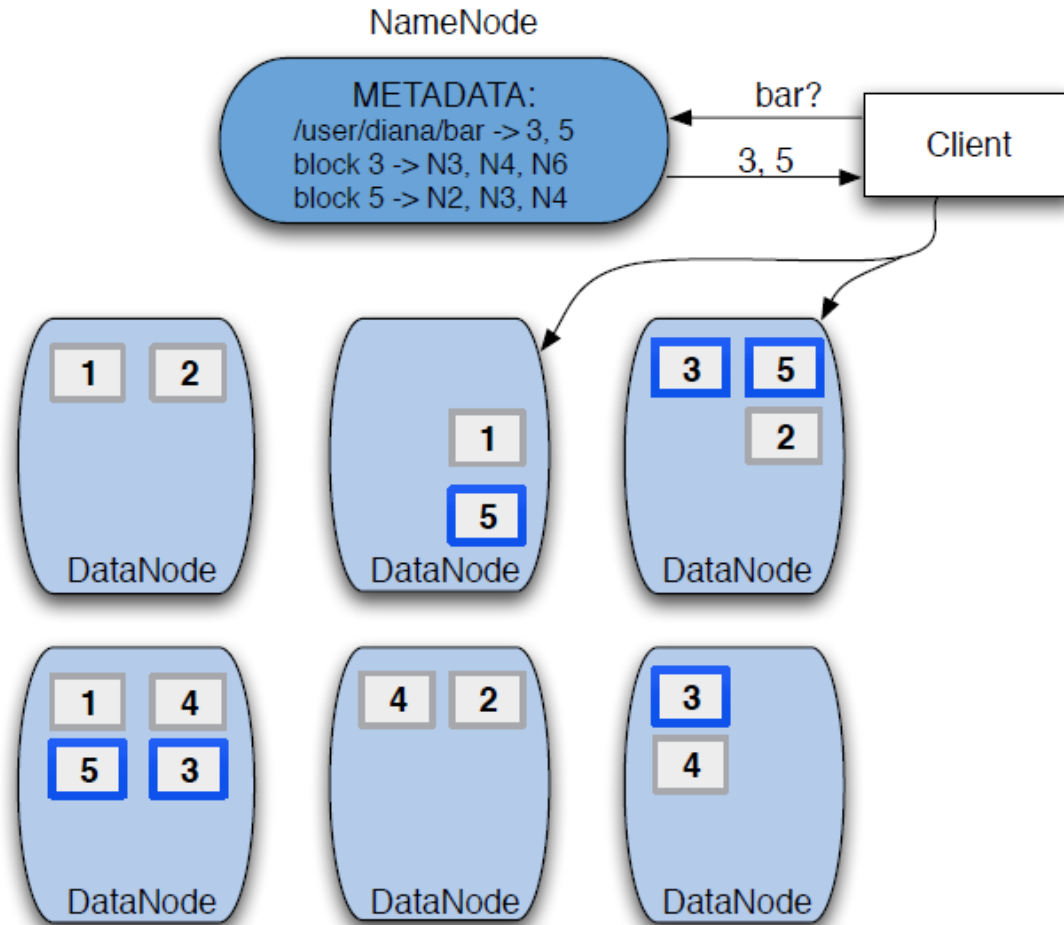


How Files Are Stored: Example



- **NameNode holds metadata for the data files**
- **DataNodes hold the actual blocks**
 - Each block is replicated three times on the cluster

HDFS: Points To Note



- **When a client application wants to read a file:**
 - It communicates with the NameNode to determine which blocks make up the file, and which DataNodes those blocks reside on
 - It then communicates directly with the DataNodes to read the data

Core Hadoop Concepts

- **Applications are written in high-level code**
 - Developers do not worry about network programming, temporal dependencies etc
- **Nodes talk to each other as little as possible**
 - Developers should not write code which communicates between nodes
 - ‘Shared nothing’ architecture
- **Data is spread among machines in advance**
 - Computation happens where the data is stored, wherever possible
 - Data is replicated multiple times on the system for increased availability and reliability

Hadoop: Very High-Level Overview

- **When data is loaded into the system, it is split into ‘blocks’**
 - Typically 64MB or 128MB
- **Map tasks (the first part of the MapReduce system) work on relatively small portions of data**
 - Typically a single block
- **A master program allocates work to nodes such that a Map task will work on a block of data stored locally on that node**
 - Many nodes work in parallel, each on their own part of the overall dataset

Fault Tolerance

- If a node fails, the master will detect that failure and re-assign the work to a different node on the system
- Restarting a task does not require communication with nodes working on other portions of the data
- If a failed node restarts, it is automatically added back to the system and assigned new tasks
- If a node appears to be running slowly, the master can redundantly execute another instance of the same task
 - Results from the first to finish will be used

MapReduce

- **MapReduce is a method for distributing a task across multiple nodes**
- **Each node processes data stored on that node**
 - Where possible
- **Consists of two phases:**
 - Map
 - Reduce

Features of MapReduce

- **Automatic parallelization and distribution**
- **Fault-tolerance**
- **Status and monitoring tools**
- **A clean abstraction for programmers**
 - MapReduce programs are usually written in Java
- **MapReduce abstracts all the ‘housekeeping’ away from the developer**
 - Developer can concentrate simply on writing the Map and Reduce functions

MapReduce: The Mapper

- **Hadoop attempts to ensure that Mappers run on nodes which hold their portion of the data locally, to avoid network traffic**
 - Multiple Mappers run in parallel, each processing a portion of the input data
- **The Mapper reads data in the form of key/value pairs**
- **It outputs zero or more key/value pairs**

```
map(in_key, in_value) ->  
    (inter_key, inter_value) list
```


MapReduce: The Mapper (cont'd)

- **The Mapper may use or completely ignore the input key**
 - For example, a standard pattern is to read a line of a file at a time
 - The key is the byte offset into the file at which the line starts
 - The value is the contents of the line itself
 - Typically the key is considered irrelevant
- **If it writes anything at all out, the output must be in the form of key/value pairs**

MapReduce Example: Word Count

- Count the number of occurrences of each word in a large amount of input data

```
Map(input_key, input_value)
  foreach word w in input_value:
    emit(w, 1)
```

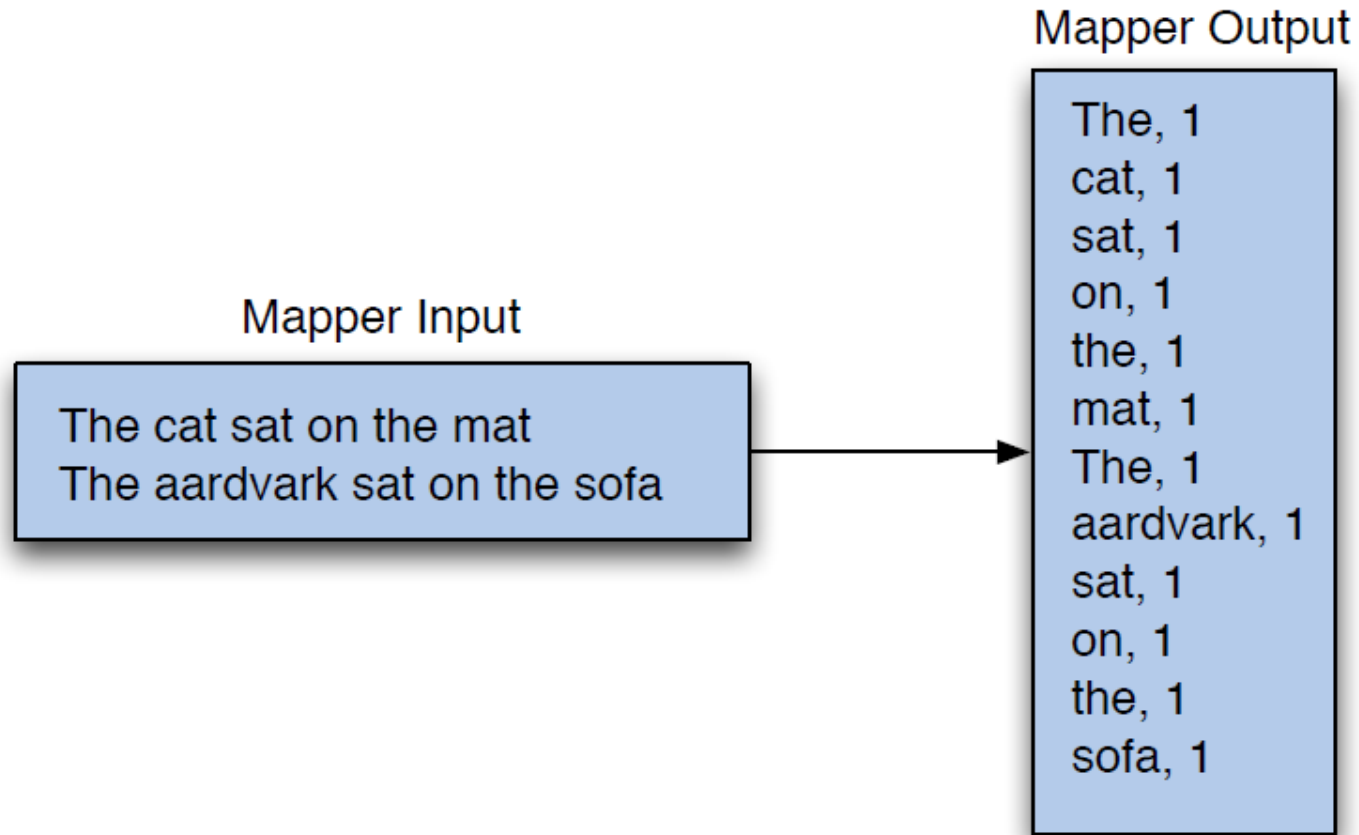
- Input to the Mapper

```
(3414, 'the cat sat on the mat')
(3437, 'the aardvark sat on the sofa')
```

- Output from the Mapper

```
('the', 1), ('cat', 1), ('sat', 1), ('on', 1),
('the', 1), ('mat', 1), ('the', 1), ('aardvark', 1),
('sat', 1), ('on', 1), ('the', 1), ('sofa', 1)
```

Map Phase



MapReduce: The Reducer

- **After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list**
- **This list is given to a Reducer**
 - There may be a single Reducer, or multiple Reducers
 - All values associated with a particular intermediate key are guaranteed to go to the same Reducer
 - The intermediate keys, and their value lists, are passed to the Reducer in sorted key order
 - This step is known as the 'shuffle and sort'
- **The Reducer outputs zero or more final key/value pairs**
 - These are written to HDFS
 - In practice, the Reducer usually emits a single key/value pair for each input key

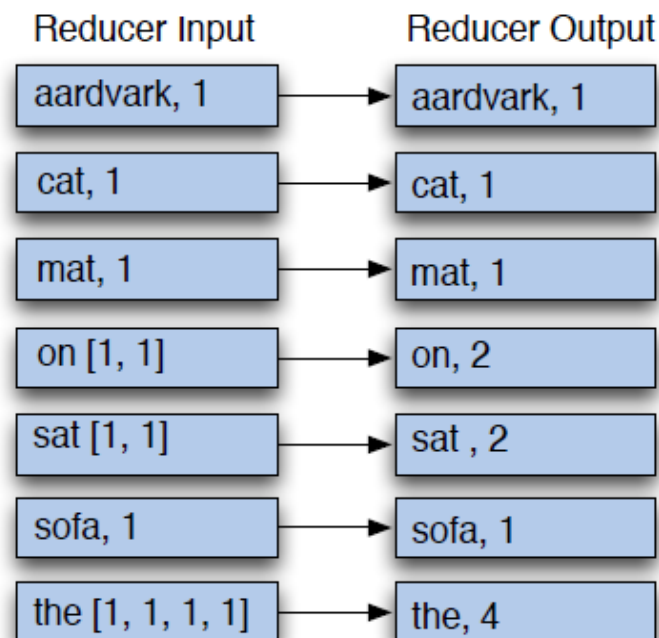
Example Reducer: Sum Reducer

- Add up all the values associated with each intermediate key:

```
reduce(output_key,  
        intermediate_vals)  
    set count = 0  
    foreach v in intermediate_vals:  
        count += v  
    emit(output_key, count)
```

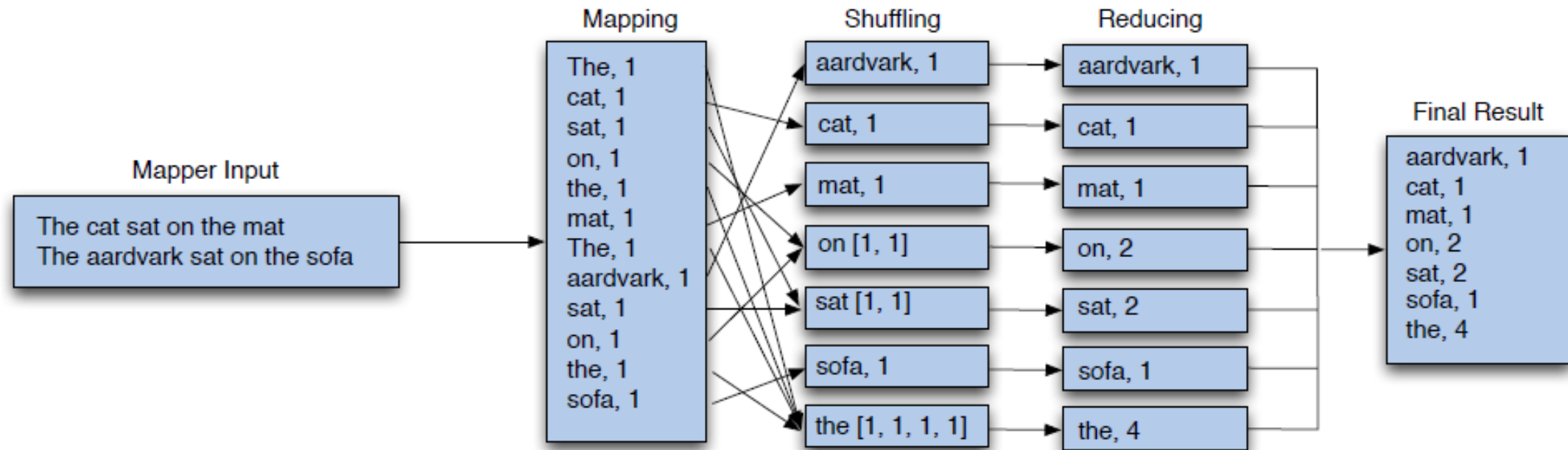
- Reducer output:

```
('aardvark', 1)  
( 'cat', 1)  
( 'mat', 1)  
( 'on', 2)  
( 'sat', 2)  
( 'sofa', 1)  
( 'the', 4)
```



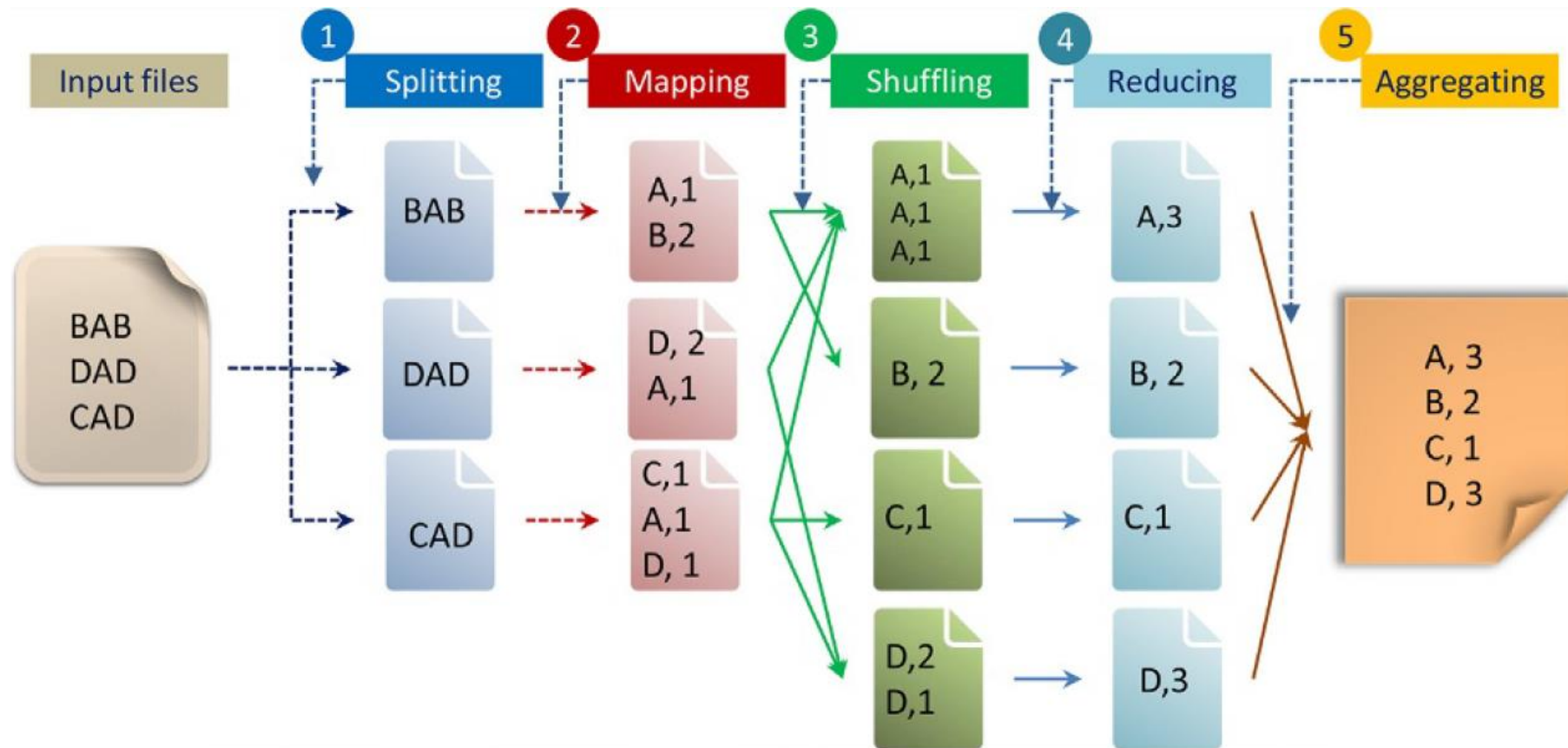
Putting It All Together

The overall word count process



Five steps MapReduce programming model

Step 1: Splitting, Step 2: Mapping (distribution), Step 3: Shuffling and sorting, Step 4: Reducing (parallelizing), and Step 5: Aggregating.



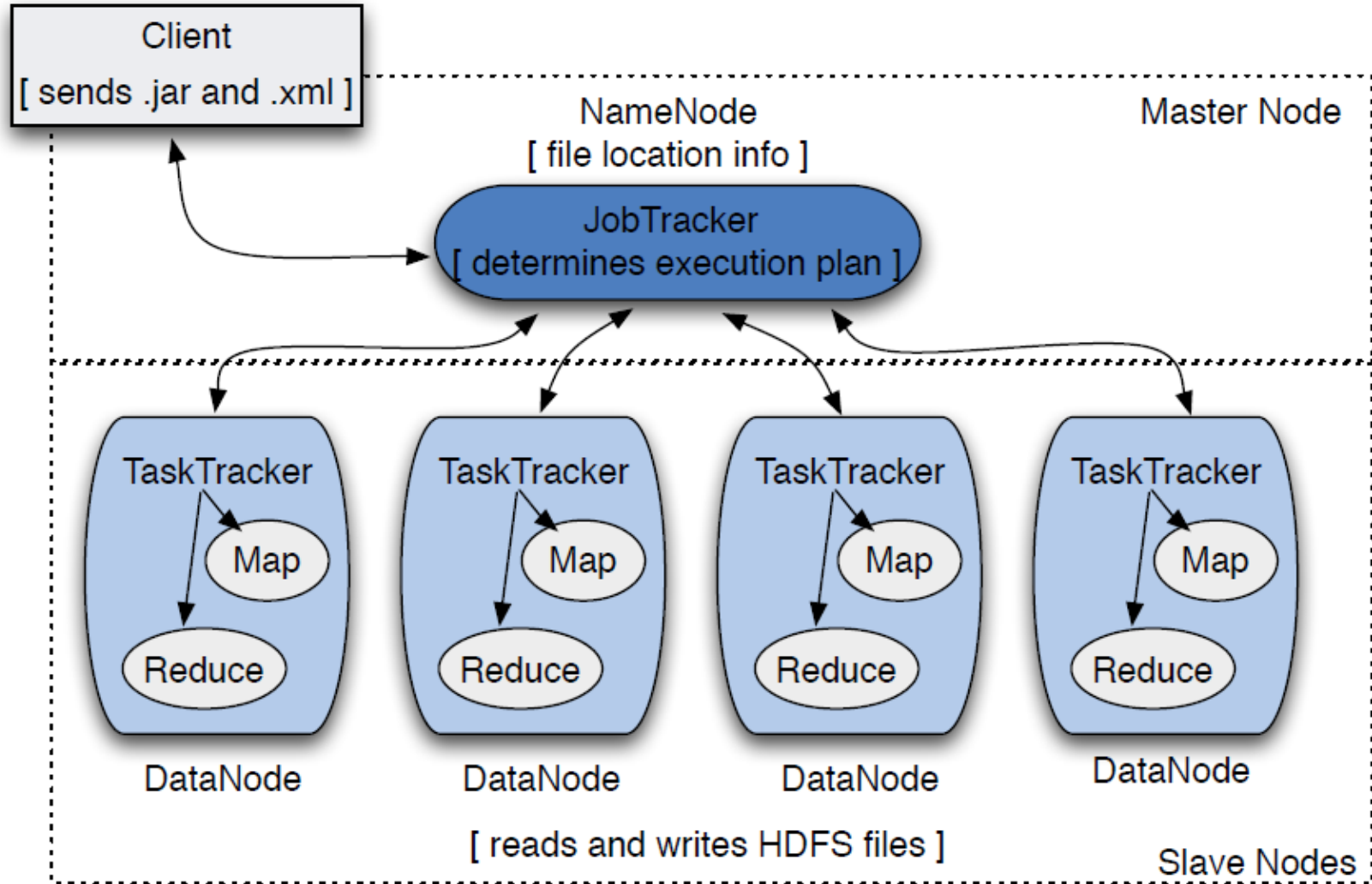
Why Do We Care About Counting Words?

- **Word count is challenging over massive amounts of data**
 - Using a single compute node would be too time-consuming
 - Using distributed nodes require moving data
 - Number of unique words can easily exceed the RAM
 - Would need a hash table on disk
 - Would need to partition the results (sort and shuffle)
- **Fundamentals of statistics often are simple aggregate functions**
- **Most aggregation functions have distributive nature**
 - e.g., max, min, sum, count
- **MapReduce breaks complex tasks down into smaller elements which can be executed in parallel**

Hadoop Cluster

- **Hadoop is comprised of five separate daemons:**
- **NameNode**
 - Holds the metadata for HDFS
- **Secondary NameNode**
 - Performs housekeeping functions for the NameNode
 - Is not a backup or hot standby for the NameNode!
- **DataNode**
 - Stores actual HDFS data blocks
- **JobTracker**
 - Manages MapReduce jobs, distributes individual tasks to...
- **TaskTracker**
 - Responsible for instantiating and monitoring individual Map and Reduce tasks

Basic Cluster Configuration



Area to apply

1. Modeling true risk
2. Customer churn analysis
3. Recommendation engine
4. PoS transaction analysis
5. Analyzing network data to predict failure
6. Threat analysis
7. Search quality
8. Data “sandbox”

Point of Sale Transaction Analysis

Challenge:

- **Analyzing Point of Sale (PoS) data to target promotions and manage operations**
 - Sources are complex and data volumes grow across chains of stores and other sources

Solution with Hadoop:

- **Batch processing framework**
 - Allow execution in parallel over large datasets
- **Pattern recognition**
 - Optimizing over multiple data sources
 - Utilizing information to predict demand

Typical Industry:

- Retail



Hadoop: Assumptions

- Hadoop is designed for large clusters, assuming hardware failures are common.
- Batch processing with high throughput is emphasized over low latency.
- HDFS handles gigabytes to terabytes of data per typical file.
- Hadoop offers high data bandwidth, scaling to hundreds of nodes and supporting millions of files.
- Hadoop applications follow a write-once-read-many access model, prioritizing data movement efficiency.

Summary

- **Apache Hadoop is a fast-growing data framework**
- **What problems exist with 'traditional' large-scale computing system**
- **Features the Hadoop Distributed File System (HDFS)**
- **Concepts behind MapReduce & Hadoop cluster operates**

Reference

Cloudera – Introduction to Hadoop

https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html