

Introduction to Hadoop and Ecosystems

MSBA 6331 Prof Liu

1

Goals

- To give you a quick overview of Hadoop and ecosystem.
- In this section, you will learn
 - The motivation behind Hadoop
 - What Hadoop is and its core components
 - Characteristics of HDFS
 - How MapReduce supports large scale data processing
 - Examples of translating computing tasks into MapReduce steps
 - Key members of the Hadoop ecosystem
 - Understand the relationship between Hadoop and data lake
 - Understand the impact of cloud computing on big data

2

2

Introduction to Hadoop and Ecosystems

HADOOP OVERVIEW

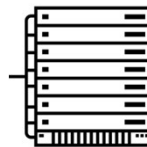
3

3

What's Hadoop?



- A set of open-source programs and procedures created in 2005
- Used for processing large amounts of data
- Servers running applications on clusters
- Handles parallel jobs or processes



A Hadoop cluster is a collection of computers working together to perform tasks

4

4

Why Study Hadoop & MapReduce?

- MapReduce as a programming framework is generally applicable beyond Hadoop
 - It is a good foundation for understanding newer big technologies (e.g. Hive & Spark)
- There are still legacy on-premise enterprise data solutions that is based on Hadoop
- A good starting point for understanding big data ecosystem

5

5

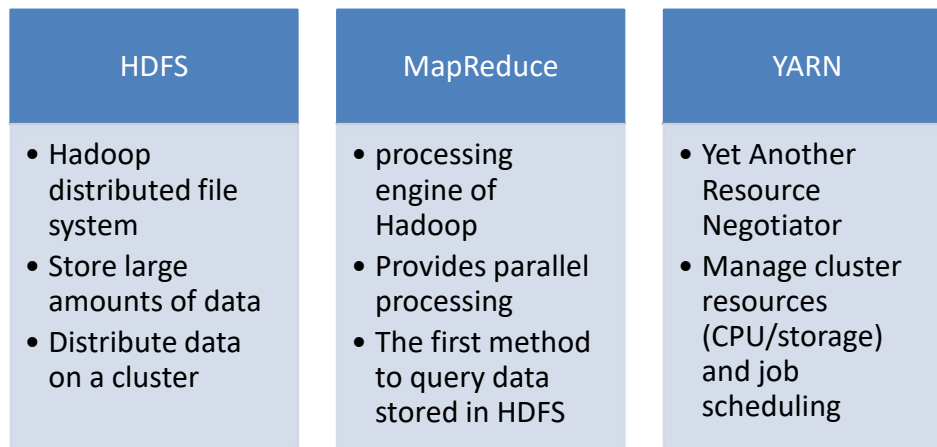
Why Motivated Hadoop?

- We have too much data to process by a single computer
 - It may take a single computer a long time to read and process 100 GB of data, if possible at all
 - Need parallel processing
- Traditional databases are too expensive
 - Can we store large quantities of data using cheap hardware?
- Traditional tools not good with semi- and un-structured data (e.g., logs and images)

6

6

Core Components Hadoop



7

7

HDFS: Hadoop Distributed File System

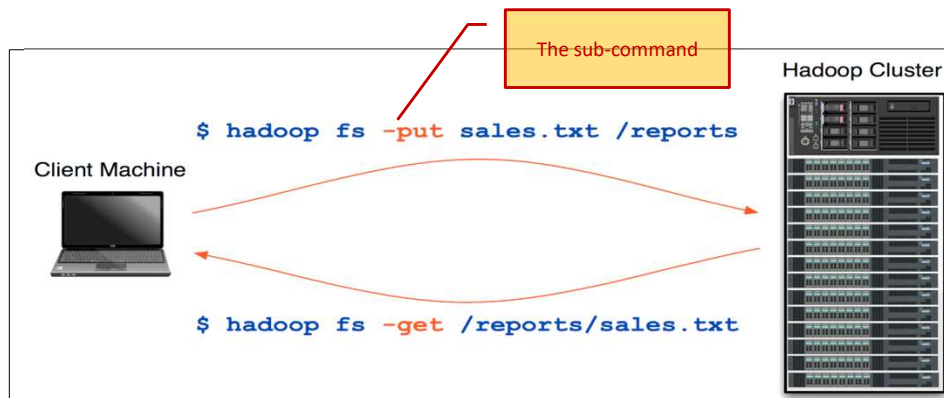
- Provides **inexpensive** and **reliable** storage for **massive** amounts of data
 - Optimized for a relatively small number of large files
 - Each file likely to exceed 100 MB, multi-gigabyte files are common
 - Store file in hierarchical directory structure
 - e.g., /sales/reports/asia.txt
 - Cannot modify files once written (i.e., **immutable**)
 - Need to make changes? remove and recreate
 - Break large files into smaller blocks, and store **multiple copies** of each block across the cluster.
 - This redundancy helps with reliability.

8

8

Copying Local Data To And From HDFS*

- Remember that HDFS is separated from your local filesystem
 - Use `hadoop fs -put` to copy local files to HDFS
 - Use `hadoop fs -get` to copy HDFS files to local files



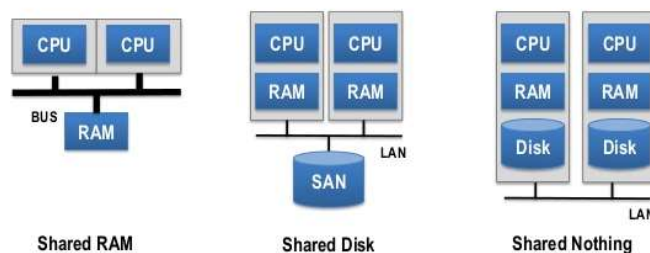
* indicates optional materials.

9

9

MapReduce

- MapReduce is not a language, it's a programming model
- Consists of two functions: `map` and `reduce`
- Follows **shared-nothing** architecture
 - parallel tasks do not share memory or DISK



10

10

Map function

- MapReduce works with key-value pairs
- The **map** function, **written by user**, takes an input pair and produces a set of *intermediate* key/value pairs.
 - $(k1, v1) \rightarrow \text{list}(k2, v2)$, e.g.:
 - $(, "fox\ jumps\ over") \rightarrow [("fox", 1), ("jumps", 1), ("over", 1)]$
 - $(, "fox\ jumps\ over") \rightarrow (, "FOX\ JUMPS\ OVER")$
 - Typically used to “break down” data
 - Filter, transform, parse data, break up a sentence, etc
 - Multiple workers can invoke the map function in parallel, each handling its own portion of the data.

11

11

Reduce function

- The **reduce** function, also written by the user, accepts an intermediate key and a set of values for that key. It merges these values together to form a possibly smaller set of values (0, 1, or many)
 - $(k2, \text{list}(v2)) \rightarrow \text{list}$
 - $(\text{"jumps"}, \text{list}(1, 1, 2, 1, 1)) \rightarrow (\text{"jumps"}, 6)$
 - Typically used to aggregate data from the map function
 - Multiple workers may invoke the reduce function in parallel, each responsible for a specific key range.
- A reduce step is optional, You can run something called a “map-only” job
- Between map and reduce, there is typically a hidden phase known as the “Shuffle and Sort” which organizes map outputs for delivery to the reducer.

12

12

A MapReduce Example: Word Count

- The following slides will explain an entire MapReduce job
 - **Input:** a text document (a list of sentences)
 - **Output:** word counts

```
SQL DW SQL
SQL SSIS SSRS
SQL SSAS SSRS
DW BI SQL
```



```
BI, 1
DW, 2
SQL, 5
SSAS, 1
SSIS, 1
SSRS, 2
```

13

13

Traditional Imperative Programming

```
word_counts = {}
file_path = 'input.txt'
with open(file_path, 'r') as file:
    for line in file:
        words = line.split()
        for word in words:
            word_counts[word] = word_counts.get(word, 0) + 1
# output word_counts to a csv file
```

Generated by ChatGPT

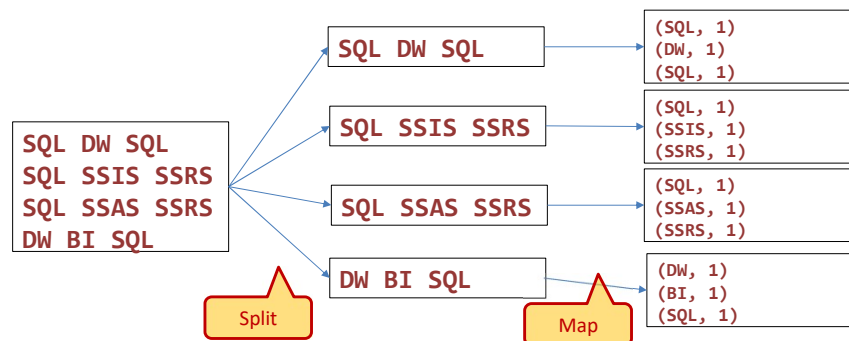
What could be the problem(s)
when you have a very big file?

14

14

A MapReduce Example – Split and Map

- Hadoop divides data into many splits
 - The number of map workers is determined by the amount of input data
 - Each map worker receives a portion of the overall input to process
 - In this case, there are 4 map workers, each processing one split
 - in practice, each worker can be assigned to multiple split, and each split can have multiple records

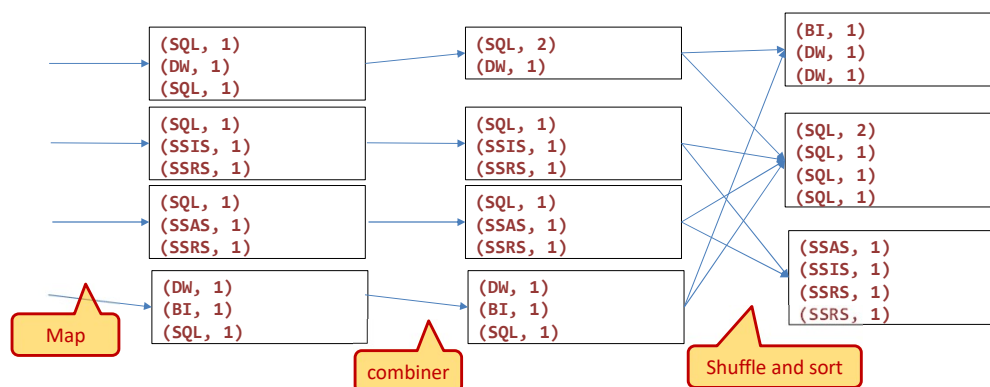


15

15

A MapReduce Example – Shuffle & Sort

- Map outputs are shuffled and sorted by key
 - Sometimes a combiner runs before shuffle and sort to reduce the amount of data transferred over the network
 - There may not be a combiner for certain aggregate operations
 - E.g. there are combiners for sum and max. but there is no combiner for average.
 - Shuffle and sort is implicit and carried out by the MapReduce framework.
 - Its results are fed into reducers via an iterator (we have three in this case)

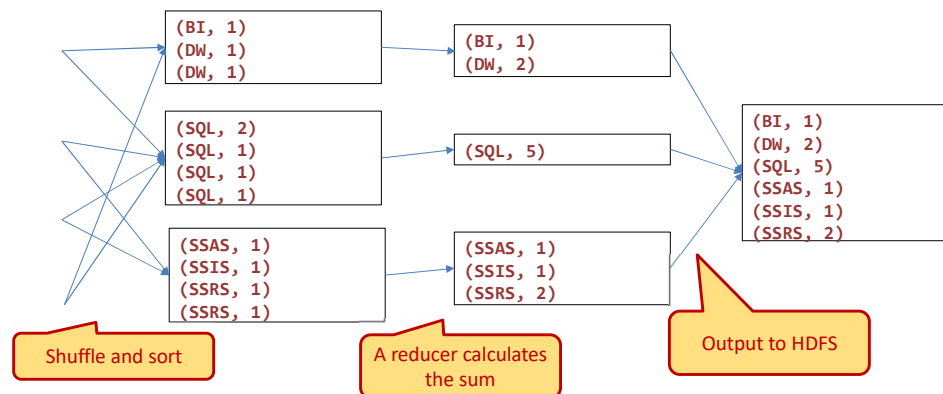


16

16

A MapReduce Example – Reduce

- Reducer input comes from the shuffle and sort process
 - All values for the same key are feed into a reducer
 - The reducer worker aggregates the values and emit a key-value pair
 - Each reducer's output is written to a job specific directory in HDFS



17

17

Benefits of MapReduce

- Simplicity (via fault tolerance)
 - Particularly when compared with other distributed programming models
- Flexibility
 - Works with **more data types** than RDBMS (key, value pairs can accommodate different kinds of structured and unstructured data)
- Scalability
 - **Horizontal scaling**: scale by adding more machines to the pool of resources.

Vertical scaling: scale by adding more powerful CPU/RAM to an existing machine

18

18

Exercise: MapReduce

- Work out the following example using pence & paper.
- **Input:** a file consists of order ID, employee name, and sales amount, separated by spaces

```
0 Alice 5
1 Bob 4
2 Alice 3
3 Alice 9
4 Diana 1
5 Bob 9
```



Paper & pencil approach to MapReduce

- **Desired output:** sum of sales by employees.
 - Following the word count example to draw the steps of a MapReduce workflow (assuming two map nodes & two reduce nodes)

19

Limitations of Hadoop

- Cannot handle **transactions***
- Cannot handle dependencies in computation.
 - E.g., record 2 must be processed after record 1
 - E.g., iterative computation in machine learning.
- High latency, not good for real-time applications
 - More suitable for batch processing
- Not built for processing lots of small files

- MapReduce is largely superseded by Spark and Tez engines.
- HDFS faces strong competition from cloud storage (e.g., Amazon S3)

20

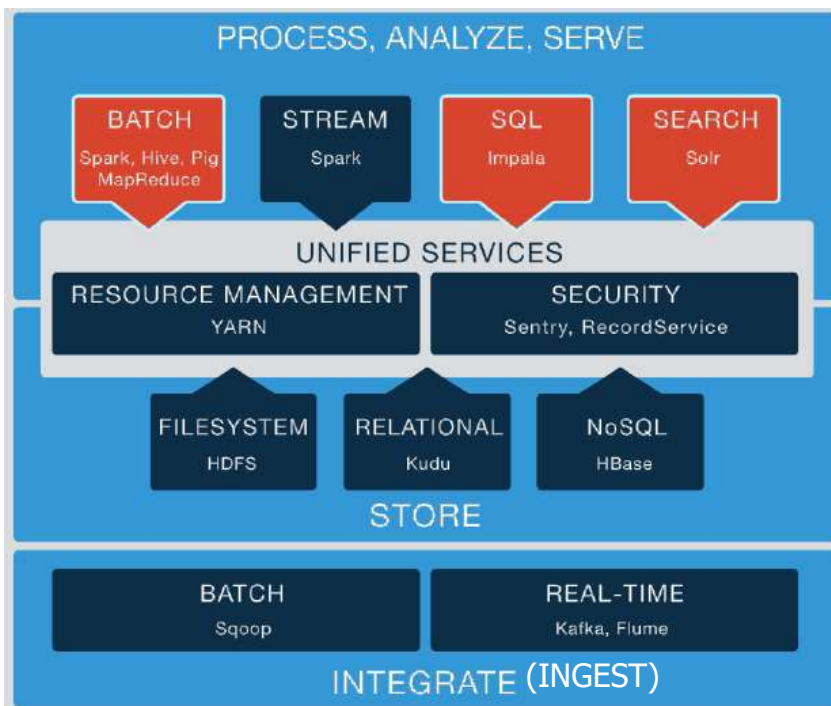
20

Introduction to Hadoop and Ecosystems

HADOOP ECOSYSTEM

21

21



Most of the Apache ecosystem is open source software.

22

22

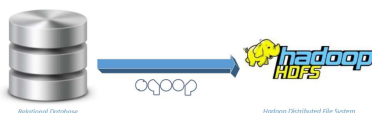
Data Storage



- **Hadoop Distributed File System (HDFS)**
 - HDFS is the storage layer for Hadoop
 - Provides inexpensive reliable storage for massive amounts of data on commodity hardware
 - Data is distributed when stored
- **Apache HBase**
 - A NoSQL distributed database built on top of HDFS
 - Scales to support very large amounts of data
 - High throughput
 - Facebook's messaging platform was based on Hbase
 - Maps **row key**, **col key**, **timestamp** to arbitrary byte array.
 - No high-level query language, no support for SQL, API access only.

23

Data Ingestion Tools



- **HDFS**
 - Direct file transfer
- **Apache Sqoop**
 - High speed import for HDFS from RDBMS (and vice versa)
 - Supports many RDBMS:
 - E.g. Oracle, Teradata, MySQL, MongoDB, Netezza

24

Data Integration Tools (2)



- **Apache Kafka for streaming data**
 - A high throughput, scalable **messaging** system
 - Distributed, reliable publish-subscribe system

<https://kafka.apache.org/powered-by>



- **Apache Flume for streaming data**
 - Distributed service for ingesting streaming data
 - Ideally suited for **event data** from multiple systems

25

Batch Processing Tools (1 of 3)



- **Apache Pig** builds on Hadoop to offer high-level data processing
 - An alternative to write low-level MapReduce code
 - Useful for ETL (extract, transformation, and loading)
 - Used by developers and analysts
- **Pig Interpreter**
 - Turns Pig Latin scripts into MapReduce (or Spark) Jobs
 - Submits those jobs to a Hadoop cluster

```
people = LOAD '/user/training/customers' AS (cust_id, name);
orders = LOAD '/user/training/orders' AS (ord_id, cust_id, cost);
groups = GROUP orders BY cust_id;
totals = FOREACH groups GENERATE group, SUM(orders.cost) AS t;
result = JOIN totals BY group, people BY cust_id;
DUMP result;
```

26

Batch Processing Tools (2 of 3)



- **Apache Hive** is the data warehouse application for Hadoop.
 - Uses a SQL-like language called HiveQL
 - Useful for managing and querying large tables in Hadoop
 - Translate SQL to MapReduce (Spark, Tez) jobs

```
SELECT  zipcode, SUM(cost) AS total
FROM    customers
JOIN    orders
      ON customers.cust_id = orders.cust_id
WHERE   zipcode LIKE '63'
GROUP BY zipcode
ORDER BY total DESC;
```

27

Batch Processing Tools (3 of 3)



- **Spark:** Spark is a fast large-scale in-memory processing engine
 - Interface with YARN clusters and HDFS, and many other storage options
 - Highly popular, has overtaken Hadoop
 - Has core spark and four components
 - SQL, Streaming, MLlib, GraphX
- **Spark SQL** – for SQL operations; not a SQL engine but flexible for SQL based data munging.

28

Interactive Query Tools (1 of 2)



- **Apache Impala** is a high-performance, interactive SQL engine
 - Very low latency – measured in milliseconds
 - Run on Hadoop clusters
 - Supports a dialect of SQL (Impala SQL)
 - Query data using a special massively parallel processing (MPP) engine

```
SELECT  zipcode, SUM(cost) AS total
FROM    customers
JOIN    orders
  ON    customers.cust_id = orders.cust_id
WHERE   zipcode LIKE '63'
GROUP BY zipcode
ORDER BY total DESC;
```

29

Interactive Query Tools (2 of 2)



**Inspired by
Google's Dremel**



Apache Drill is a open-source distributed, low-latency SQL engine that can query a variety of structured and semi-structured data sources.

- Query semi-structured data formats and sources, e.g., HDFS, HBase, MongoDB, Amazon S3, RDBMS, JSON/Parquet files
- Uses Standard ANSI SQL
- scale to thousands of nodes and petabytes of data at interactive speeds that BI/Analytics environments require

```
SELECT * FROM dfs.root.`/web/logs`;

SELECT country, count(*)
FROM mongodb.web.users
GROUP BY country;

SELECT timestamp
FROM s3.root.`clicks.json`
WHERE user_id = 'jdoe';
```

30

Machine Learning



- **Apache Spark MLlib**: a Machine Learning component of spark
- **H2O**: in-memory, distributed, fast, and scalable machine learning platform; can be used from R, Python, Scala.
 - Produced by the company H2O.ai (but software is open source).
 - **H2O/Sparkling Water**: Integrate H2O models into the Spark computing framework

31




Streaming





- **Apache Storm** is a distributed and fault-tolerance real-time computation platform
 - Guaranteed data processing
 - Fault-tolerance, scalability, high-performance
- **Spark Streaming**: the streaming component of Apache Spark.
- **Apache Flink**: open source distributed streaming processing framework that supports both event-time processing and batch analytics

32

The Hadoop Ecosystem

Data ingestion/integration:    

Storage:   

Batch processing:   

Interactive Processing: Impala  

Machine Learning:  

Streaming processing:  STORM  

33

Introduction to Hadoop and Ecosystems

HADOOP ADOPTION AND TRENDS

34

34

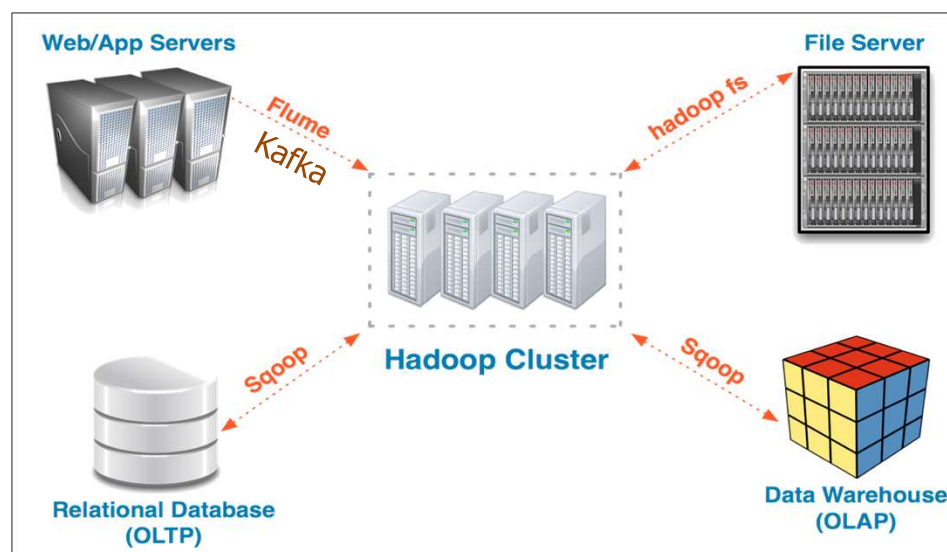
Hadoop & Data Lake

- **A Data Lake approach**
 - Traditional **Data Mart /Data Warehouse** is a "store of bottled water"
 - Data lake is a "large body of water in a more nature state" (James Dixon, CTO of Pentaho) – store both structured and unstructured data, often in more raw form.
 - Contents of the data lake stream in from a source to fill the lake, and various users can examine, dive in, or take samples.
- **Hadoop is a platform for building data lakes**
 - It provides for storing a mixture of structured and unstructured data, side by side.
 - It could be used to store far more detail about the transaction than you would in an ordinary RDBMS (due to costs limitations)

Source: data-science-central.com
 Read more: Hester, L. 2016. "Maximizing Data Value with a Data Lake," Data-Science-Central.Blog.

35

Hadoop and Data Center



36

36

Hadoop Trends

- 2019 Gartner Survey: 19% has Hadoop in use
- But the deployment experience has not been great; require large amount of management/implementation resources.
- **Cloud's** flexible expandability and management may better fulfill use cases addressed today by Hadoop **on-premises**.

37

37

Big Data and Cloud Computing

- Cloud Service Providers are also offering competing big data solutions in accelerated speeds:
 - Amazon Web Services (AWS): EMR, Redshift
 - Microsoft Azure: HDInsight, Data Lake Analytics
 - Google Cloud Computing (GCP): BigQuery, Cloud Dataflow
 - **Databricks**: Hosted Spark
 - **Snowflake**: Cloud-based Datawarehousing
- The use of HDFS is declining as data moves to the cloud
 - Cloud storage: Amazon S3, Azure Data Lake Storage (ADLS), Google Cloud Storage

38

38