Recap:

Hadoop

Daemons :

DFS →data storage framework

1. Namenode

1. Datanode

1. Sec namenode

YARN ⇒ data processing framework

4 . resource manager

5. Node manager

HDFS ?

1.Sharding

WORM → write once Read many

2. Distributed

3. Replicator factor ---> setrep 50 dir /file

=======================================

HDFS shell commands

hadoop fs -mkdir

-copyFromLocal -put

--can we write multiple files into hdfs

/retail

Sales.txt

Sales1.txt

Sales2.txt

-rm

-rm -R

-rmdir -empty directory

-copyToLocal -get

-ls

-ls r / day2

-cp

-mv

-cat

-head -tail

-chmod 777

777 - all

400 ---REad only

444

Rwx rwx rwx

Read 4

Write 2

Execute 1

========================================

Select loc, avg(sal) from employee group by job ;

loc →sal

Hyd 5000

Hyd 7000

Beng 8999

Hyd 55000

Employee 10m ---> Oracle ---> server

2m 2m 2m 2m 2m

D1 C d4 c d8 c d16 c d29c

M1 code M2 DB

Human ←----- food

Tiger -------> food

=====================================

Mapreduce is a programming paradigm to process the distributed data(HDFS) in a parallel way

V1 → HDFS + Mapeduce

V2 → HDFS + YARN

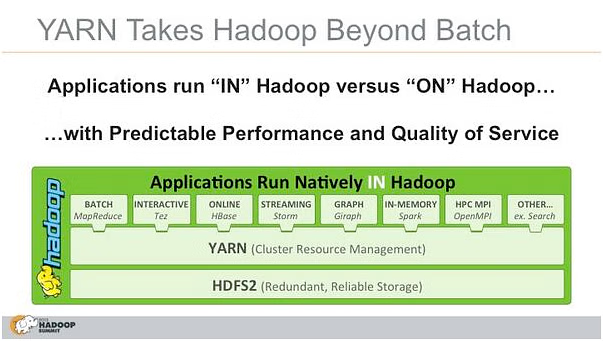
YARN →MR → Batch processing

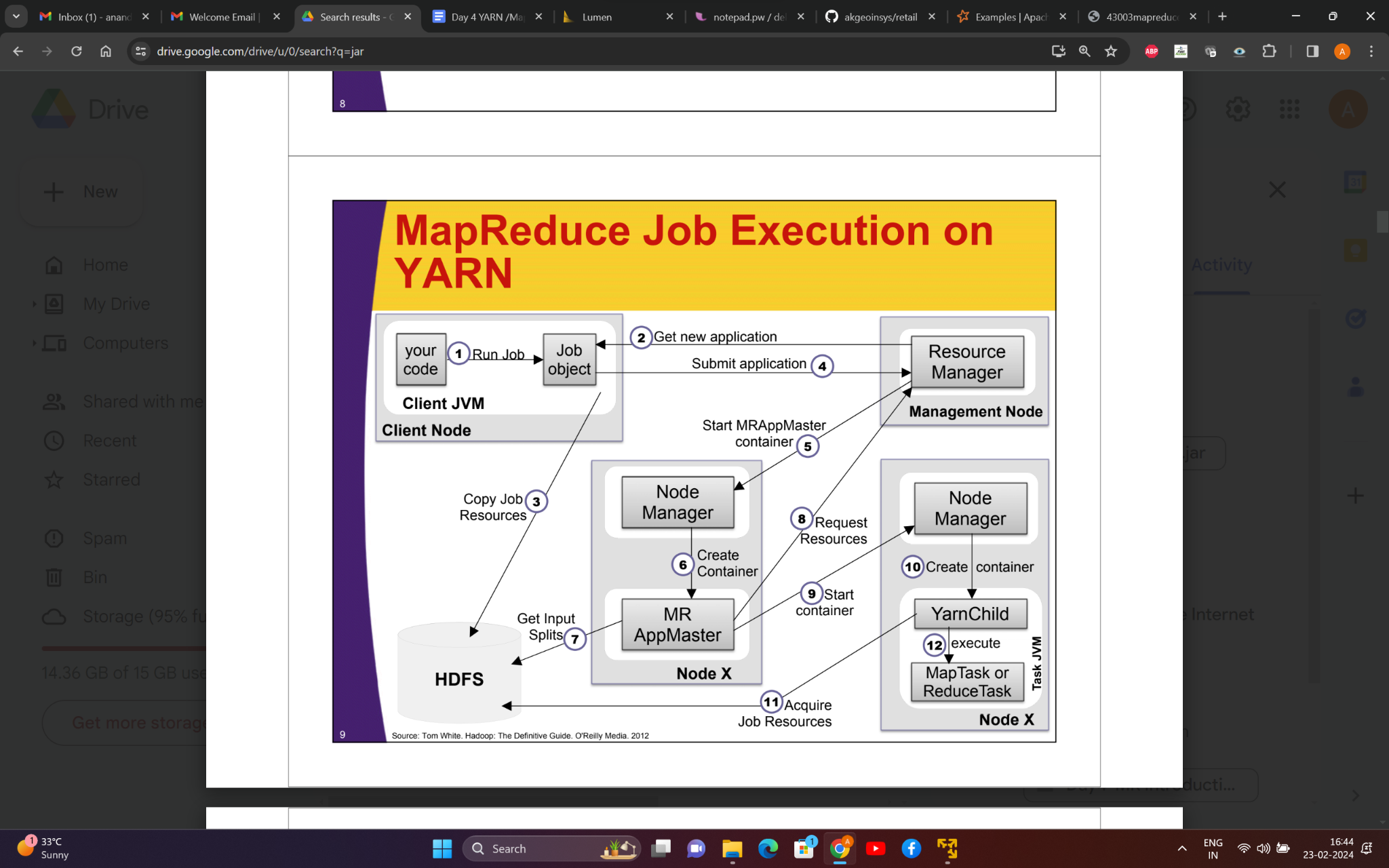
Mahout → Machine learning

Giraph → Graph processing

Spark → in-memory processing

Storm -> Streaming processing





1. Processing Model:

Hadoop: Primarily designed for batch processing. It uses MapReduce for processing data in two steps: Map phase and Reduce phase.

Spark: Supports both batch and real-time processing. It introduces the Resilient Distributed Dataset (RDD) abstraction, which allows for in-memory processing and iterative algorithms.

2. Speed:

Hadoop: Typically slower than Spark due to its reliance on disk storage for intermediate data between map and reduce stages.

Spark: In-memory processing significantly speeds up iterative algorithms, making it faster than Hadoop for certain workloads.

3. Ease of Use:

Hadoop: More complex and requires developers to write verbose MapReduce code.

Spark: Offers high-level APIs in Scala, Java, Python, and R, making it more user-friendly. It also supports SQL queries through Spark SQL.

4. Data Processing:

Hadoop: Better suited for large-scale batch processing and handling massive datasets.

Spark: More versatile, capable of handling batch processing, interactive queries, streaming, and machine learning workloads.

5. Fault Tolerance:

Hadoop: Achieves fault tolerance through data replication. If a node fails, the task can be rerun on another node with a copy of the data.

Spark: Uses lineage information to reconstruct lost data, allowing for fault recovery. It also supports data replication for additional fault tolerance.

6. Data Storage:

Hadoop: Uses the Hadoop Distributed File System (HDFS) for distributed storage.

Spark: Can use HDFS or other storage systems like HBase, S3, and more.

7. Integration:

Hadoop: Has a broader ecosystem with various tools like Hive, Pig, and others.

Spark: Growing ecosystem with integration points for machine learning (MLlib), graph processing (GraphX), and SQL-based queries (Spark SQL).

8. Resource Management:

Hadoop: Uses MapReduce as the default resource management system.

Spark: Can work with various cluster managers like Apache Mesos, Hadoop YARN, or standalone.

9. Use Cases:

Hadoop: Well-suited for batch processing and storing large amounts of data.

Spark: Suitable for real-time data processing, iterative algorithms, machine learning, and interactive queries.

**Spark**

Apache Spark is an open-source, distributed computing system that provides a fast and general-purpose cluster-computing framework for big data processing. Here are detailed features of Apache Spark:

**In-Memory Processing:**

Spark performs in-memory processing, allowing it to cache intermediate data in memory, which significantly speeds up iterative algorithms and repetitive queries.

Resilient Distributed Datasets (RDD):

RDD is the fundamental data structure in Spark, representing a fault-tolerant collection of objects that can be processed in parallel. It allows for parallel transformations and supports fault recovery.

DAG (Directed Acyclic Graph) Execution Engine:

Spark uses a directed acyclic graph for job execution, optimizing task execution and enabling iterative processing.

Multiple Language Support:

Spark provides APIs in Scala, Java, Python, and R, making it accessible to a broader audience of developers.

Spark SQL:

Spark SQL provides a programming interface for data manipulation using SQL queries. It supports both structured and semi-structured data.

DataFrames:

Spark introduces DataFrames, which are distributed collections of data organized into named columns. DataFrames provide a higher-level API and optimization opportunities over RDDs.

**Structured Streaming:**

Spark supports structured streaming for real-time data processing. It enables continuous processing of live data streams with DataFrame and SQL APIs.

**Machine Learning Library (MLlib):**

MLlib is Spark's machine learning library, offering a set of high-level APIs for machine learning algorithms. It includes tools for classification, regression, clustering, and collaborative filtering.

**Graph Processing (GraphX):**

GraphX is Spark's graph processing library, providing an API for graph computation and analysis. It allows users to express graph computation workflows efficiently.

**Spark Streaming:**

Spark Streaming is a micro-batch processing engine for stream data, enabling the processing of live data streams in near real-time.

**Integration with Hadoop Ecosystem:**

Spark can run on Hadoop Distributed File System (HDFS) and is compatible with Hadoop's YARN cluster manager. It can also integrate with HBase, Hive, and other Hadoop ecosystem components.

Cluster Managers:

Spark can run on various cluster managers, including Apache Mesos, Hadoop YARN, and its standalone built-in cluster manager.

Fault Tolerance:

Spark achieves fault tolerance through lineage information, which allows lost data to be reconstructed in case of node failures. It also supports data replication for additional fault tolerance.

Ease of Use:

Spark provides high-level APIs in different languages, making it easier to use for developers. It also includes built-in libraries for common tasks, reducing the need for external tools.

Community and Ecosystem:

Spark has a vibrant and active open-source community, contributing to its ecosystem. It integrates with various tools and technologies, expanding its capabilities