EECS 4314 - Advanced Software Engineering

Apache Hadoop Conceptual Architecture

Kimberly Bonilla (218091306)
Pawanjot Randhawa (219732718)
Kennie Oraka (219163104)
Aishwarya Narayanan (219350248)

York University, Lassonde School of Engineering
October 2025

Abstract:

This report explores the conceptual architecture of Apache Hadoop, a Java-based open-source framework designed for distributed storage and large-scale data processing. Hadoop enables organizations to manage a large number of datasets by splitting the workload across clusters of hardware, and this provides both scalability and fault tolerance.
The report looks at the framework into four core subsystems, HDFS (Hadoop Distributed File System) for storage, YARN (Yet Another Resource Negotiator) for resource management, MapReduce for data processing, and Hadoop Common for shared libraries/utilities and configurations.

Each subsystem is further analyzed in terms of its architecture style, interactions, and overall responsibilities within the system. To connect the technical aspects of Hadoop with real-world applications, we have included two practical use cases that show how the system handles large-scale data processing. The use cases we used focus on identifying trending hashtags on social media and building a video recommendation system. Furthermore, the team reflects on how we used generative AI tools to aid in our research process and make sense of complex documentation which helped us to clarify and better understand architectural concepts. Today, Hadoop continues to evolve as the foundation for many big data infrastructures and the report highlights how each of Hadoop's subsystems work together to deliver reliability, scalability, and efficiency.

Table of Contents:

1) Introduction:

1.1) Hadoop Overview

Hadoop is a Java-based open source framework that manages the storage and processing of big data by utilizing distributed storage and parallel processing. It is widely utilized by major companies such as Facebook and Amazon because it effectively handles large datasets using a cluster of commodity hardware. This means that data is stored in replicas from across multiple machines, so if one computer in the cluster fails (which often happens), Hadoop keeps running without interruption. It's built to detect failures automatically and recover at the software level, ensuring the system remains reliable and available even when individual nodes go down [1].

1.2) Hadoop Conceptual Architecture

The overall architecture of Hadoop follows a layered architecture composed of the four main subsystems: YARN (Resource Management System), HDFS (Hadoop Distributed File System for Data Storage), MapReduce (for data processing) and Hadoop Common. The dependencies as shown in Figure 1 illustrate the interactions between the subsystems.

MapReduce relies on YARN for coordination, resource management, and job scheduling. YARN and MapReduce rely on HDFS to access the data when needed for YARN, and store the output data from a MapReduce job to improve performance. Finally, all the Subsystems rely on Common, which is a shared layer that provides libraries, utilities, configuration files for communication, and I/O operations [2]. The diagram shows that some subsystems rely more heavily on others, for instance, MapReduce depends on all components, while HDFS relies only on the Common library. Understanding each subsystem's role helped us to have a clear conceptual architecture of their dependencies.

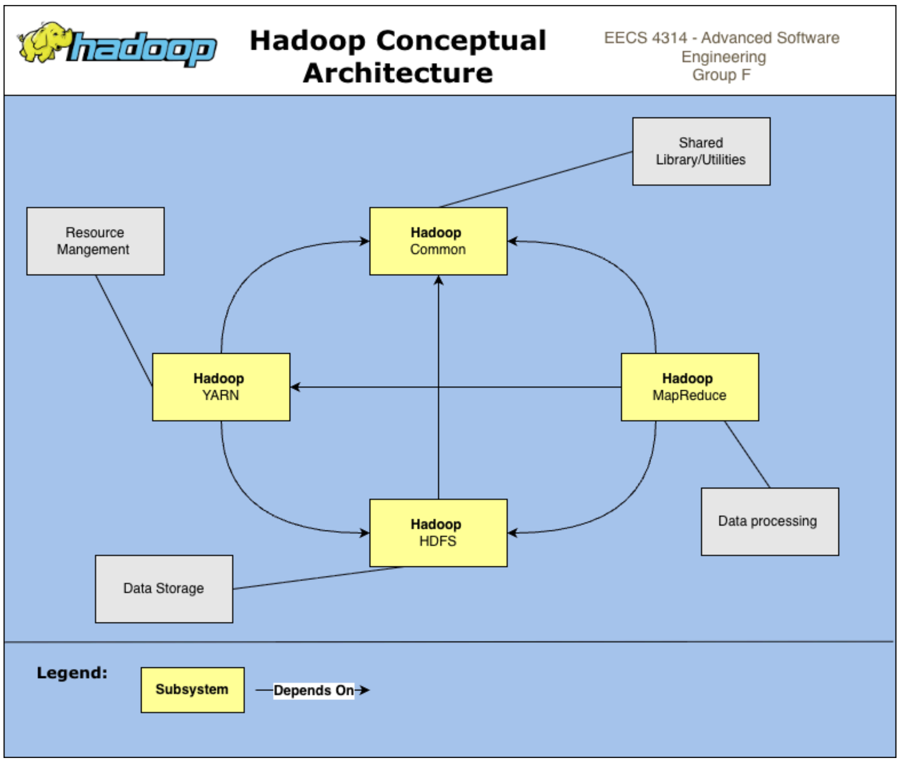

Figure 1: Hadoop Conceptual Architecture Diagram

2) Subsystem Architecture:

2.1) HDFS

2.1.1 Overview

The Hadoop Disturbed File system is what enables the large scale data storage and access of the system. It implements a master/slave architecture, in which there are two types of nodes. A single NameNode that acts as the master, and multiple Datanodes that act as the slaves. The NameNode manages and regulates the system, while the datanodes are responsible for the actual storing and processing of data. The data itself is stored by being split into blocks and stored across multiple datanodes, which is all tracked by the nameNode (Figure 2).[3]

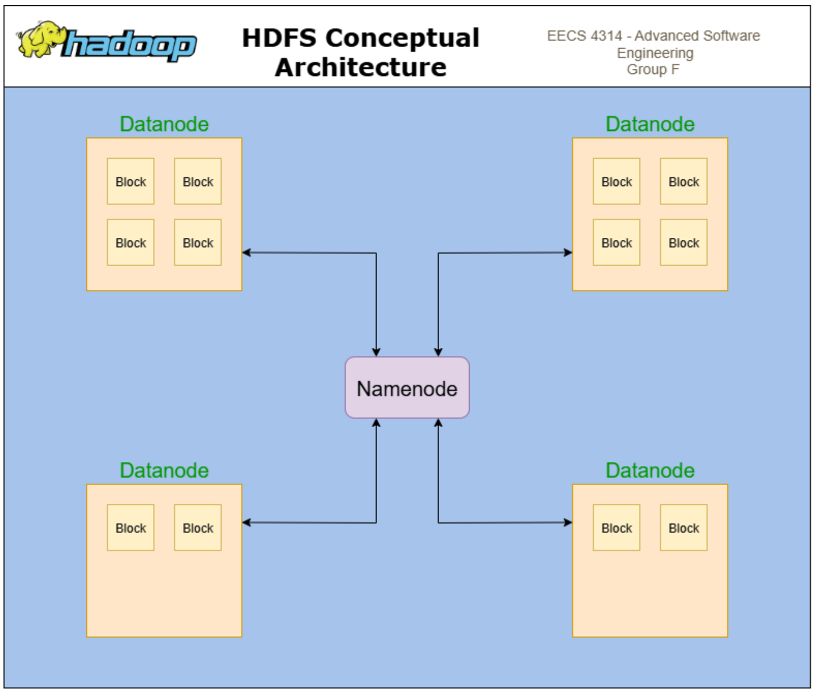
hadoop
HDFS Conceptual Architecture
EECS 4314 - Advanced Software Engineering
Group F
Datanode
Block
Namenode

Figure 2: HDFS Master/Slave Architecture

2.1.2 Blocks

Blocks are smaller, fixed size pieces of data. In the HDFS system, large files are split into these smaller blocks, allowing the system to store and process large amounts of data. For any particular file in the system, all blocks except the last one will be of the same size. By default, the size is set to 128 MB in most modern versions, however this value can be changed to better suit the application's requirements. After the blocks have been created, they are also duplicated three times by default, another setting which can be changed. The duplication is done in order to improve the system's reliability, so that a piece of data is not reliant on one node in the system. If a node were to fail, the system can automatically retrieve the block from another node and still manage to recreate the original file. After the blocks have been created, they are sent to be stored in the various datanodes, ensuring the copies exist on separate nodes. The mapping process is performed by the Namenode.[3]
An example of how this works can be seen in Figure 3:

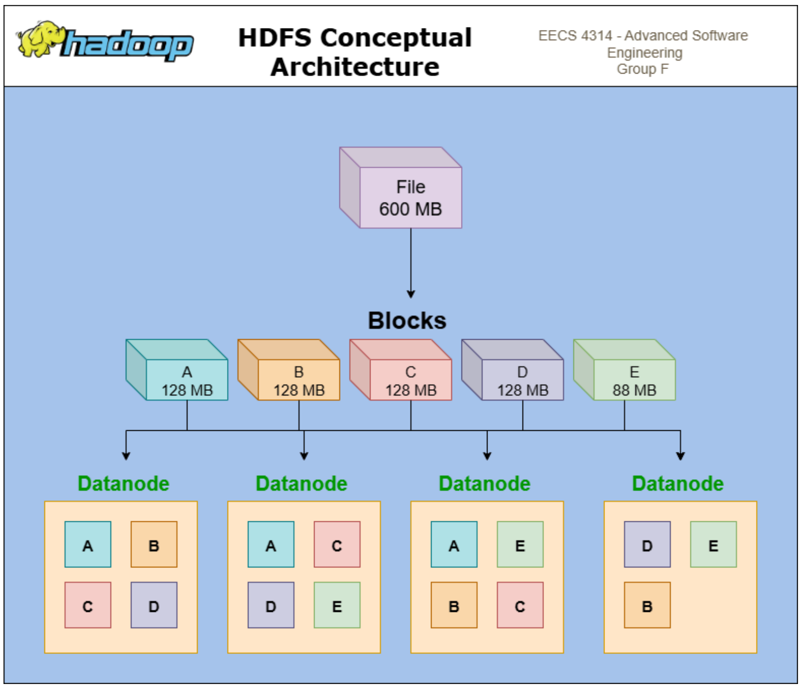

Figure 3: HDFS Block Replication and Recovery Process

2.1.3 Namenode-Datanodes Interactions

The Namenode is the system's master, and acts as the control center of the system. It manages the file system namespace, and includes file metadata such as file names, sizes, types, and authors. The Namenode also keeps a transaction log called an EditLog, which keeps track of changes that happen to this metadata. The actual file namespace itself and the mapping of all blocks is stored in a file called the FsImage. Both the EditLog and FsImage are used in order to recover and maintain the state of the HDFS system.

The Datanodes act as the worker or slave nodes, and are responsible for the actual storing and processing of data. They execute both read and write operations as directed by the users, and can also perform operations such as block creation, deletion, and replication when instructed. Datanodes store each block in a separate file on their local file system, and create an optimal number of directories and sub directories for the blocks to be stored in. Datnodes also periodically send out both a heartbeat and a block report to the Namenode. The heartbeat relays that the Datanode is functioning as expected, while the block report contains a list of all blocks on this Datanodes local file system.

In terms of actual usage, the system is set up so that the client only communicates with the Namenode. Actions the client wishes to perform such as reading or writing data are sent to the Namenode which will get the locations of the blocks the client wishes to interact with. It then returns this list of blocks to the client, after which the client will then connect to Datanodes containing those blocks, and instructs the nodes to perform the operations. The datanodes then send the block data directly to the client, where they are reassembled into the original file. [3]
Figure 4 below illustrates this flow, where if a client want to read a file, they connect with the name node which returns the nodes which contain the blocks, after which the client connects to those nodes and reads the data:

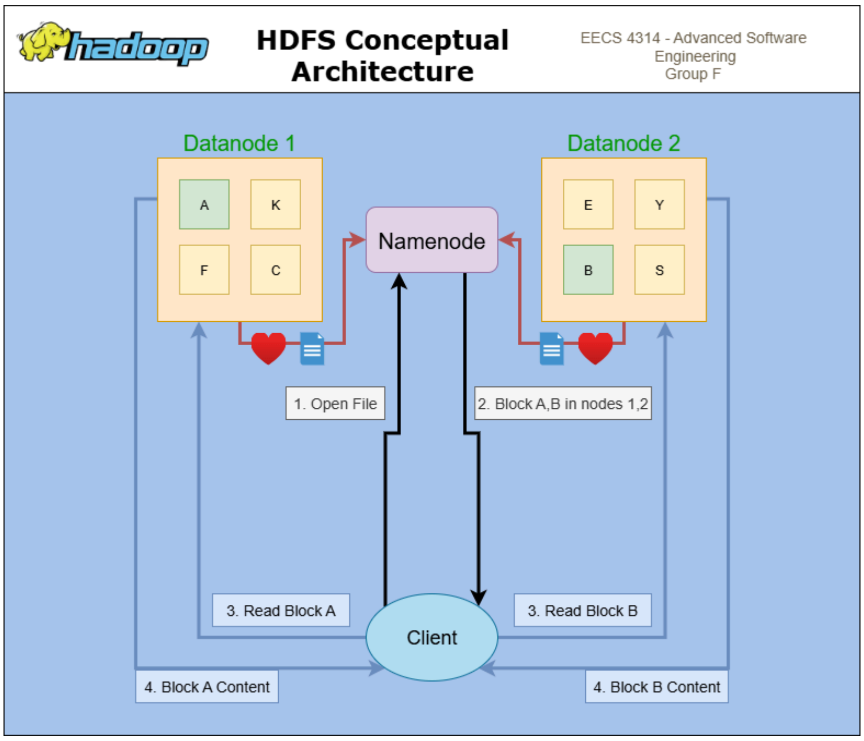

Figure 4: HDFS File Read Workflow

2.2) YARN

2.2.1 Overview

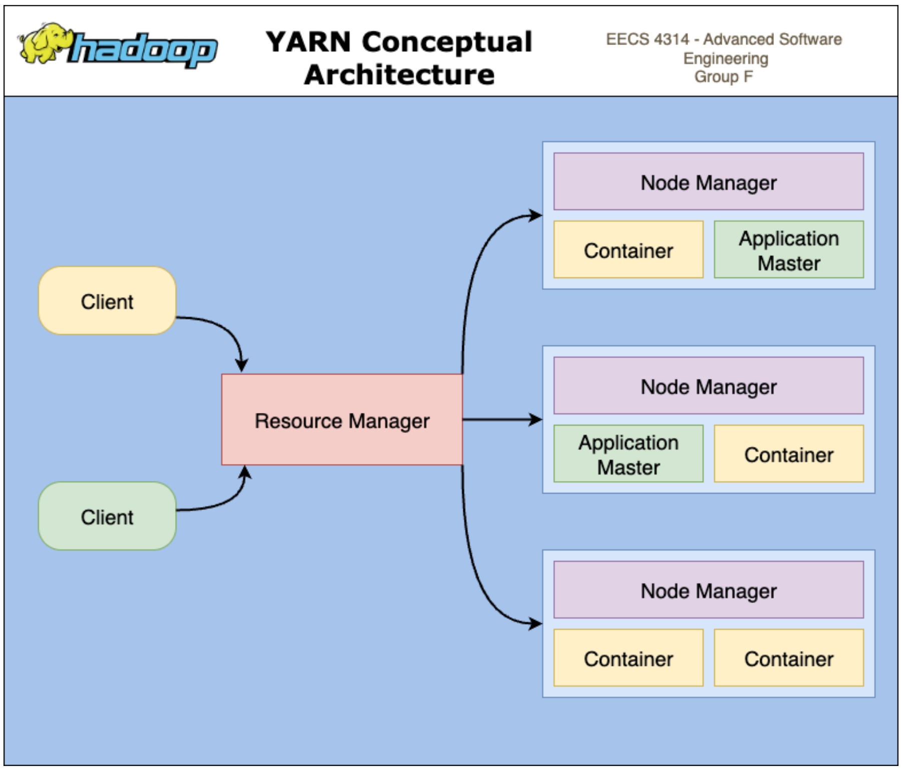

Figure 5: YARN Conceptual Architecture

The Yet Another Resource Negotiator (YARN) is a key part of Hadoop that manages and coordinates how resources are shared across different applications running in the cluster. Its main objective is to improve scalability and efficiency by separating resource management from the processing logic. The Resource Manager is the primary coordinator of this system. It receives job requests from clients, determines how to allocate cluster resources, and tracks which nodes are available. This design allows multiple applications to run concurrently, while ensuring that each uses only the resources it requires [7].

Each machine in the cluster hosts a Node Manager, which is responsible for managing the resources on that node. The Node Manager monitors the performance of the node and reports this information back to the Resource Manager. It is also responsible for launching and managing containers, which are the isolated environments where application tasks actually run. Containers are allocated fixed amounts of CPU, memory, and storage, ensuring that applications do not interfere with one another [9].

For every application submitted, a dedicated process called the Application Master is created. The Application Master runs inside one of the containers and coordinates all tasks related to that specific application. It communicates with the Resource Manager to request additional resources and interacts with Node Managers to launch and monitor task containers [7]. Figure 5 reflects this process: clients communicate with the Resource Manager, which then coordinates with multiple Node Managers that host Application Masters and containers. Overall, this architecture enables YARN to support multiple processing frameworks and manage cluster resources efficiently.

2.3) MapReduce

2.3.1 Overview

The MapReduce framework is designed to process large-scale data in parallel by dividing tasks into smaller ones, which are then executed across clusters. The framework splits the data into parts and processes these sections onto different worker nodes, and then aggregate the results to provide the final output. Scheduling and monitoring are also a feature of MapReduce to ensure stability and fix any errors/faults. In terms of the Hadoop ecosystem, MapReduce runs on the same node set as HDFS and uses YARN management, to ensure fault-tolerance and efficient performance. The parallelism combined with fault-tolerance make MapReduce efficient and reliable [4].
The MapReduce subsystem can be understood in terms of two main sub components, the workflow component and managing/execution of services which are explained in the next sections.

2.3.2 Managing MapReduce Services

Each MapReduce job comes with two main services: JobTracker (JT or the master) and the TaskTracker (TT or the workers). The master is responsible for allocating, monitoring and handling failures, while TaskTracker runs the assigned operations (map and reduce).

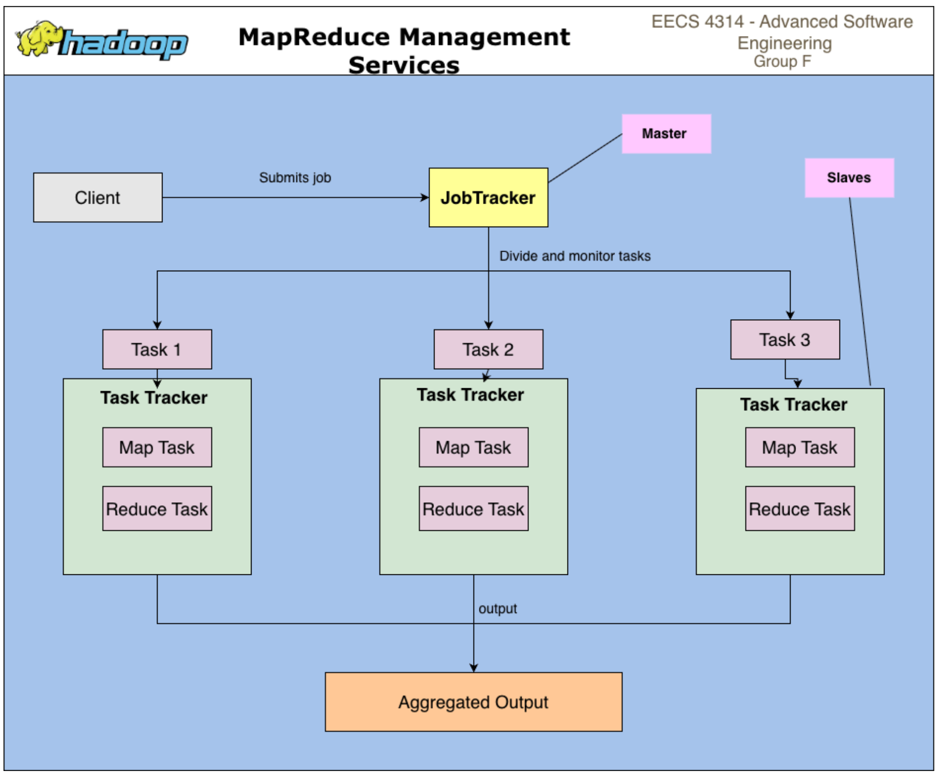

Figure 6: MapReduce Master/Slave Architecture

From figure 6 above, MapReduce framework follows a master-slave architecture, where JobTracker (master) coordinates multiple workers. In this example, the master divides the data into three tasks, queuing them in a FIFO manner, and allocating them to a correct slave (green boxes) that will handle various map and reduce phases. The workers (TaskTrackers) run multiple tasks in parallel, allowing the master to oversee, allocate and handle any failures. The results from the three tasks are then aggregated into a single output (orange box) [5]. The hierarchical model allows the master to have control over the workers while the workers have to report back to the master, thereby combining fault tolerance with efficient parallel processing.

2.3.3 MapReduce DataFlow

The Next subsystem is composed of two main operations: the map and reduce phase. It can be described as a sequence of phases that transform data between components. These phases include: input, shuffle and sort, and output. The data flow of MapReduce follows a pipe-and-filter, before analysing the reasoning, an overview of the dataflow is explained below:
Input and split: Input data stored in HDFS is taken to be processed. The data is split and each one is assigned to a mapper depending on the size. **Mapper phase:** Once the blocks have been assigned to mappers, the workers will read and apply the map function to the data. Key/value pairs are produced, ensuring only one copy is done to avoid redundancy. **Shuffle and sort:** Intermediate results from the Mapper phase will determine where the data is redistributed. Data is grouped by key, sorted, shuffled and given to a specific reducer. **Reduce phase:** The reducer processes grouped key-value pairs, aggregates the results and writes the output back to the HDFS [6].

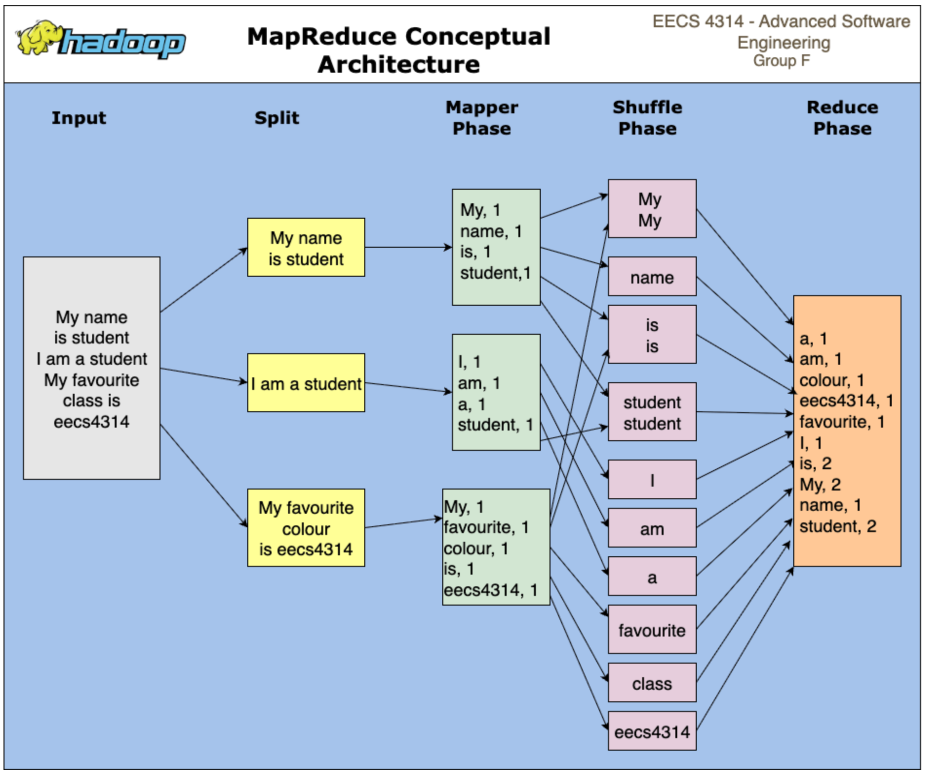
hadoop
MapReduce Conceptual Architecture
EECS 4314 - Advanced Software Engineering
Group F
Input
Split
Mapper Phase
Shuffle Phase
Reduce Phase
My name is student I am a student My favourite class is eecs4314
My name is student
I am a student
My favourite colour is eecs4314
My, 1 name, 1 is, 1 student,1
I, 1 am, 1 a, 1 student, 1
My, 1 favourite, 1 colour, 1 is, 1 eecs4314, 1
My My
name
is is
student student
I
am
a
favourite
class
eecs4314
a, 1 am, 1 colour, 1 eecs4314, 1 favourite, 1 I, 1 is, 2 My, 2 name, 1 student, 2

Figure 7: MapReduce Pipe-and-Filter Model

The Conceptual architecture for the dataflow of MapReduce, as illustrated in figure 7 follows the pipe-and-filter. It is composed of a series of filters; input and split, map, shuffle and sort, reducer, and output. The components (filters) do not share information of their current state to other filters. The connectors ensure data is being passed to the next filter. In the example above, the input “My name is student, I am a student My favourite class is eecs4314” is split in three and transferred to the mapper phase, each split is processed independently. The same logic is applied to the other filters where the Reduce phase does not know about the map phase, an advantage to this is scalability as more data can be processed at once.

2.4) Hadoop Common

2.4.1 Overview

Hadoop Common serves as the foundation for the entire Hadoop ecosystem. It provides the shared Java libraries, utilities, and APIs that HDFS, YARN, and MapReduce all depend on. These include configuration management, logging, security, I/O operations, and network communication. By offering these shared components, Hadoop Common ensures interoperability and consistency across all subsystems, and this allows them to work together within the framework [1].

2.4.2 Architecture Style

It follows a layered architectural style, where each layer provides services to the one above it and depends on the one below. As illustrated in Figure 7, the architecture starts from the hardware layer at the base, integrating with the operating system through components such as the Unix Shell and Async Profiler. Above that, the Security and Access Layer manages secure communication, authentication, and key management services. The Cluster & Utilities Layer provides tools that support distributed operations and diagnostics. Finally, the top Applications and Frameworks Layer connects to higher-level systems which includes HDFS, YARN, and MapReduce, which in turn, relies on Hadoop Common’s lower layers for consistent performance and communication.

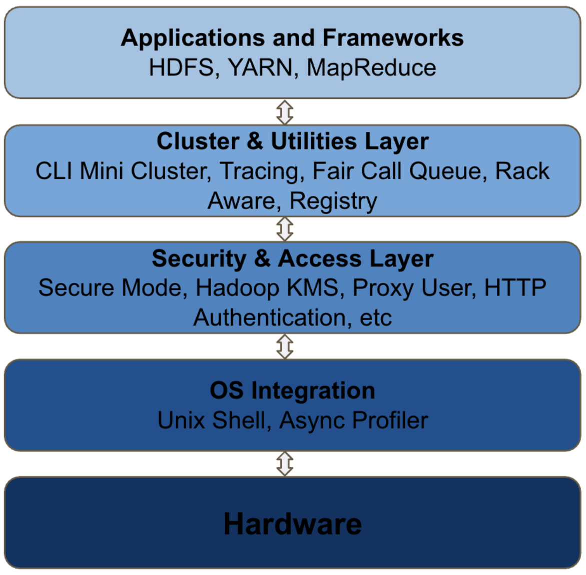
Applications and Frameworks
HDFS, YARN, MapReduce
Cluster & Utilities Layer
CLI Mini Cluster, Tracing, Fair Call Queue, Rack Aware, Registry
Security & Access Layer
Secure Mode, Hadoop KMS, Proxy User, HTTP Authentication, etc
OS Integration
Unix Shell, Async Profiler
Hardware

Figure 7: Layered Architecture of Hadoop Common

3) Use Cases:

3.1 Use Case 1: Trending Hashtags Scenario

Social media platforms have to identify trending hashtags from thousands of user posts. Over time, massive amounts of posts, comments and interactions are continuously added, making it difficult to process all of the data. Platforms have to extract hashtags and count the number of occurrences. The solution is to use a MapReduce workflow. This example clearly shows how MapReduce can be used to process and extract the trending hashtags, which explains the architecture from sections 2.4.

From the diagram below (Figure 8), the client starts off by submitting the job to the JobTracker (master) finding trending hashtags. Next, the master will request the input data from which the HDFS has already split up into blocks. The master then sends the tasks to the appropriate worker (TaskTracker's). Each TaskTracker will follow a series of maps and reduce phases, in the example this is only one task (extracting hashtags). In the map phase, hashtags are read and key-value pairs are generated. The list is then shuffled and sorted and sent to the reduce phase. The reduce phase will aggregate counts for each hashtags for example (#Hadoop, 12) (#MapReduce, 8) etc. The results of the counts will be written back to the HDFS as the output.

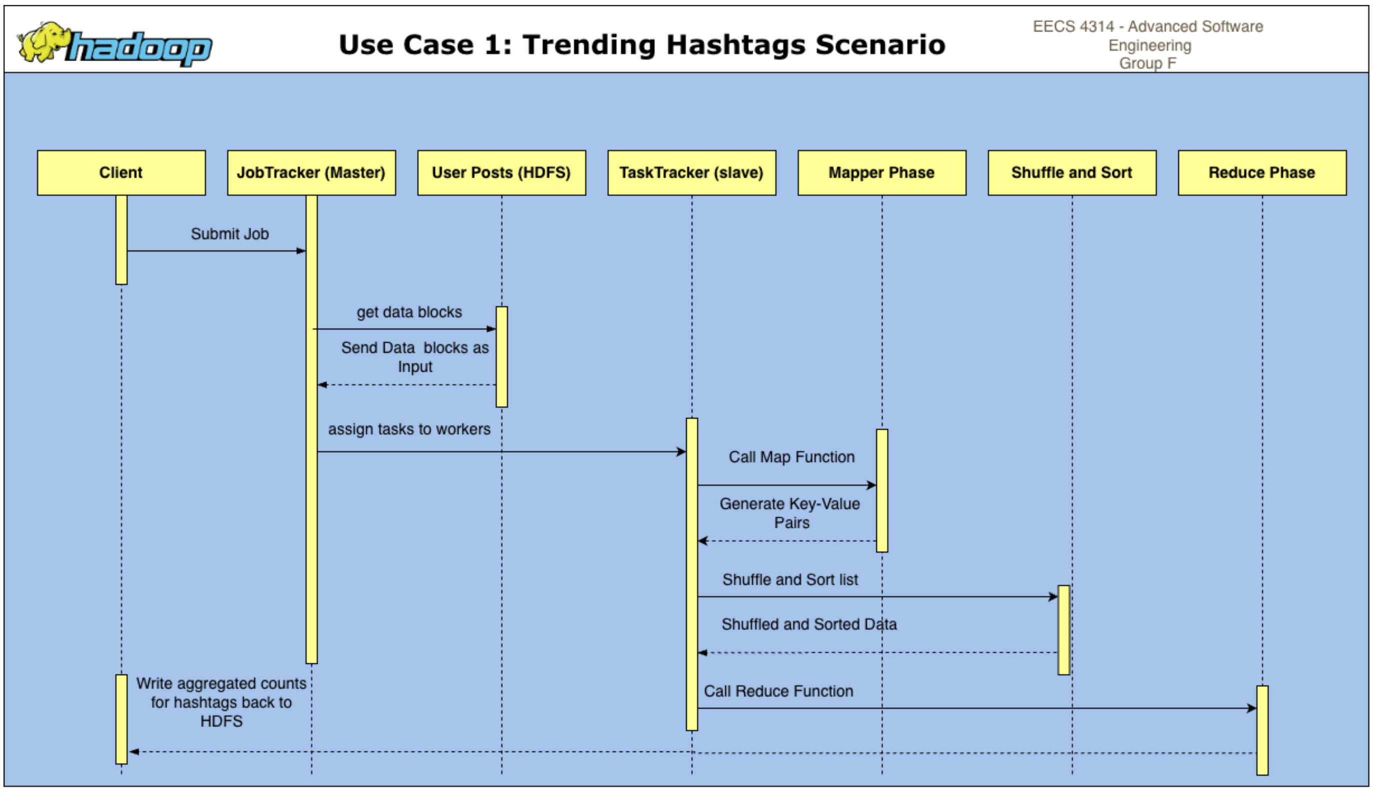

Figure 8: Trending Hashtags Scenario Sequence Diagram

3.2 Use Case 2: Video Recommendation System

Video Streaming companies likely want to recommend content that their users are more likely to engage with, as this would result in greater profits for them. To do this, they can use the huge volumes of watch and search history logs that are generated by their users. They can filter through this data to recommend content that falls into each user's preferred genre. To do this, companies could use the MapReduce workflow discussed in this report.

To start, companies can send user log data into the HDFS system for storage. They could then use the mapper in order to generate key value pairs like (userID,genre). After the data has been mapped, it would be shuffled and sorted, after which the reduce phase can begin. Here the system would aggregate counts of each userID's most viewed genre, then send these results back into the HDFS for use in their recommendation system.

The results would allow the company to see each userID's highest genre count. As an example, say a user Joe primarily viewed sports videos, and didn't care much about news. The system's results would show us results such as (joe,sports,248) and (joe,news,21). Using this data, the company could recommend more sports videos to joe to keep him more engaged, increasing their engagement rates. Hadoop in particular works great here due to the large amount of data we are filtering, as using hadoops parallel processing power we can filter through this data very efficiently. A simplified view of the process we discussed can be seen in Figure 9:

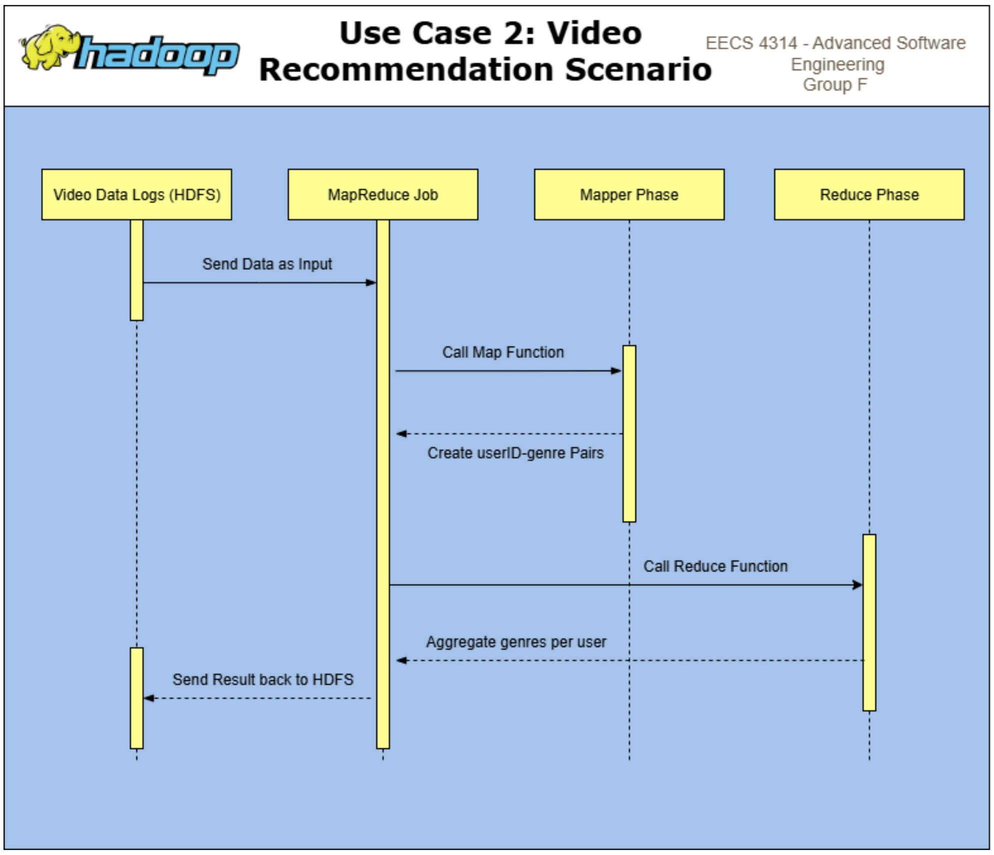

Figure 9: Video Recommendation Scenario Sequence Diagram

4) External Interfaces

Apache Hadoop exchanges information between users, applications, and distributed components through various external interfaces. Users often submit jobs through command-line tools, client programs, or graphical user interfaces (GUIs), providing information such as job configurations, input data locations, and resource requirements [4]. These submissions are transmitted over the network to the Hadoop cluster. The Resource Manager in YARN interprets this information to allocate resources and schedule tasks across the cluster. HDFS allows tasks to access large datasets stored as files across multiple nodes and to write intermediate and final outputs back to the system. Node Managers and DataNodes continuously send status updates, providing information on resource availability and task progress back to the Resource Manager and NameNode [10].

Hadoop also supports interactions with external systems beyond direct job execution. Monitoring and logging tools can access performance metrics, cluster health information, and execution logs through network messages or files [10]. Data may also be read from or written to external databases or storage systems as part of processing pipelines, allowing Hadoop jobs to integrate with other enterprise systems. During MapReduce processing, tasks read input files, exchange intermediate data over the network, and write final outputs back to HDFS or external storage [4]. These external interfaces allow Hadoop to provide a flexible and scalable platform for distributed data processing.

5) Developer Responsibilities and Implications

In Apache Hadoop, the division of responsibilities among developers influences productivity and system quality. Each subsystem (HDFS, YARN, and MapReduce) is managed by dedicated developer teams, allowing developers to focus on the reliability and performance of their specific module. For example, HDFS developers concentrate on data storage and replication, YARN developers handle resource management and job scheduling, and MapReduce developers focus on task execution and data processing [11]. This clear ownership enables parallel development, as teams can work simultaneously on different subsystems, speeding up feature implementation and system evolution. However, the interdependence of these subsystems requires careful coordination. Developers must manage APIs, data flows, and integration points to prevent conflicts and ensure smooth operation across the cluster [12]. Additionally, developers are responsible for the testing, maintenance, and quality of their subsystems. Overall, this structure helps maintain the robustness of the entire Hadoop ecosystem.

6) Lessons Learned:

For this assignment, generative AI was used to understand the entire structure of Hadoop. When doing initial research, we found that the Apache documentation was quite detailed and focused a lot on the technical aspects of each system. While this is great information to have, it takes longer to understand and digest how each part works together. This is where AI was very helpful. Generative AI was really good at breaking down and explaining the more complex parts of hadoop. This was very helpful as it allowed for a general understanding of how each part works by itself and as a whole, making it much easier to understand the technical parts the documentation was talking about.

In terms of drawbacks, Generative AI was sometimes not the best at the technical parts, as when asking specific questions it seemed to contradict previous things it had said or just simply make things up. Overall we found that it was better to use AI in order to get a basic understanding of the system, then use the documentation to deeply understand each part. The AI helped the most when making connections between the various parts, as it was really good at explaining the connections between the parts of the system. The documentation was better at explaining how each part is designed and implemented, going into depth on how each sub-system is designed and implemented. Using both resources definitely seems to be the best way to learn, as it's much faster than trying to piece everything together yourself.

7) Derivation Process and Alternatives

For this project, our derivation process involved cross-referencing the descriptions of each Hadoop subsystem with the examples and architectural styles from class to ensure consistency. We mapped the components to their corresponding architectural styles to better understand how each subsystem contributes to the overall framework.

In exploring alternatives, we compared Hadoop's MapReduce approach with Apache Spark, which builds on many of the same distributed processing principles but takes them further with in-memory computation.
MapReduce laid the foundation for large-scale parallel data processing by introducing a powerful model that splits work by writing intermediate results to the disk after each step. This makes it more ideal for batch-processing jobs where accuracy and data retention are more critical than speed. Spark, on the other hand, extends these ideas with a more memory-centric approach that keeps data in RAM between steps instead of writing intermediate results to disk. Using a Directed Acyclic Graph (DAG) execution model, Spark reduces disk I/O and can achieve processing speeds up to 100 times faster than MapReduce for smaller or iterative workloads [8].
Together, the two frameworks represent different trade-offs where MapReduce scalability, while Spark focuses on speed.

8) Conclusions:

This report analyzed the conceptual architecture of Apache Hadoop, showing how its core subsystems (HDFS, YARN, MapReduce, Hadoop Common) work together to provide scalable and efficient large-scale data processing. The use cases demonstrated Hadoop's ability to address real-world challenges, such as analyzing trending hashtags on social media and generating personalized video recommendations. Looking forward, Hadoop can continue to evolve by improving cloud integration, faster real-time data processing, and AI-driven resource management. Additionally, enhancements to monitoring tools and developer support will strengthen its role as a robust platform for managing complex data workloads.

9) References:

[1] Apache Software Foundation, “Apache Hadoop,” *Apache Hadoop Project*, [Online]. Available: https://hadoop.apache.org/

[2] Databricks, “Hadoop Ecosystem,” *Databricks Glossary*. [Online]. Available: https://www.databricks.com/glossary/hadoop-ecosyste

[3] Apache Software Foundation, “HDFS Architecture (HDFS Design),” *Hadoop HDFS Design Documentation*, [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[4] Apache Software Foundation, “MapReduce Tutorial,” *Hadoop MapReduce Client Core Documentation*, 2024. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[5] A. Verma, A. H. Mansuri, and N. Jain, “Big data management processing with Hadoop MapReduce and Spark technology: A comparison,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, Indore, India, 2016, pp. 1–4. doi: 10.1109/CDAN.2016.7570891

[6] Databricks, “MapReduce,” *Databricks Glossary*, 2024. [Online]. Available: https://www.databricks.com/glossary/mapreduce

[7] Apache Software Foundation, “Apache Hadoop YARN,” *Apache Hadoop*, version 3.4.2, 2025. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html

[8] IBM Cloud Education Team, “Hadoop vs. Spark: What’s the Difference?” *IBM Think*, 2025. [Online]. Available: https://www.ibm.com/think/insights/hadoop-vs-spark

[9] GeeksforGeeks, “Hadoop YARN Architecture,” *GeeksforGeeks Big Data Tutorial,* June 24, 2025. [Online]. Available: https://www.geeksforgeeks.org/big-data/hadoop-yarn-architecture/

[10] Apache Software Foundation, “Interface Classification,” *Hadoop Common Documentaion*, [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/InterfaceClassification.html

[11] ProjectPro, “Hadoop Developer Job Responsibilities Explained,” *ProjectPro Articles,* October 28, 2024. [Online]. Available: https://www.projectpro.io/article/hadoop-developer-job-responsibilities-explained/305

[12] GeeksforGeeks, “Hadoop Ecosystem,” *GeeksforGeeks,* August 22, 2025. [Online]. Available: https://www.geeksforgeeks.org/dbms/hadoop-ecosystem

10) Appendix

i) Data Dictionary:

Term	Definition
HDFS (Hadoop Distributed File System)	Responsible for distributed data storage across multiple nodes.
NameNode	Master node in HDFS that manages metadata, file structure, and regulates the system
DataNode	Worker node that stores data blocks and performs read/write operations (data processing).
Block	A fixed-size unit of data (typically 128 MB) into which files are divided for storage.
YARN (Yet Another Resource Negotiator)	Resource management subsystem responsible for scheduling and allocating jobs.
MapReduce	Data processing subsystem that divides workloads into map and reduce phases for parallel computation.
JobTracker / TaskTracker	Components of early Hadoop versions that managed MapReduce job coordination and execution.
Hadoop Common	Shared library that provides configuration, logging, and I/O utilities for all Hadoop components.

ii) Naming Conventions:

The naming conventions used in our described architecture follow the normal terminology and abbreviations within the Hadoop framework.

Subsystem Names: Each major component; HDFS (Hadoop Distributed File System), YARN (Yet Another Resource Negotiator), and MR (MapReduce).