Apache Hadoop Distributed File System (HDFS)

Team members and contributions

Name	Responsibilities
Trieu Huynh Ba Nguyen	1. Why did you select this particular case study? (0,5 points) 6. Revisit the properties of distributed systems discussed in the class, how are the properties considered in your case study? For instance, how it handles failures, transparency, etc. (1,5 points)
David Vigh	2. What was the situation before this distributed system was implemented? How were the tasks executed? (0,5 points) 4. What architectural style has been used in your case study? In your opinion, why is it a preferred option over other architectural styles? (1 point) 5. Are there any videos explaining your case study? If so, please mention the source. (0,5 points)
Arlis Puidet	3. How different components in your case study are interlinked? Include an architectural diagram. If downloaded from the Internet mention the source and explain the working. (1 point) 8. Does your case study include multiple data storage facilities? If so, why? (1 point)
Haoyu Zhang	7. The key requirements for any distributed system are scalability, reliability, performance, and openness. Provide three examples of where these requirements might be in conflict with each other, how does your case study deal with these potential conflicts? (2 points)

1. Why did you select this particular case study?

For this assignment, we selected the Apache Hadoop Distributed File System (HDFS), which stores data on cost-effective machines, delivering exceptionally high combined bandwidth throughout the cluster. The system was developed by the Apache Software Foundation as a part of the Nutch project in 2004 and has since become one of the most popular distributed file systems in the industry. HDFS is part of the wider Apache Hadoop open-source software collection.

We chose HDFS thanks to its typical characteristics of a well-designed distribution system: high performance, fault tolerance, and scalability. Through examining HDFS, we get to understand the mechanics of a distributed system, how it manages errors and accuracy, and how it processes a large quantity of data.

2. What was the situation before this distributed system was implemented? How were the tasks executed?

The history of human data storage can be traced back to the earliest civilizations, where information was recorded on various materials, including stone tablets, papyrus scrolls, and clay tablets. These early methods of data storage were primarily used for record-keeping, such as keeping track of taxes, trade, and other valuable information.

As technology evolved, so did the methods for data storage. For example, during the Middle Ages, monks would copy books by hand, creating illuminated manuscripts that could take years to complete. The advent of the printing press in the 15th century revolutionized the way information was stored and shared, allowing for the mass production of books and other printed materials.

The development of electronic computers in the 20th century marked a significant turning point in the history of human data storage. These machines enabled the storage and processing of vast amounts of data in a digital format, using magnetic tapes, disks, and other storage media. However, these methods are just single, local machine system. Accessing, modifying, and managing data in such forms are labour-intensive and time-consuming. As the amount of data being generated continued to grow, it became increasingly clear that new methods were needed to manage and store this data efficiently.

This led to the development of distributed systems, such as HDFS, that enabled the storage and processing of large volumes of data across multiple machines. By doing so, HDFS can store and process data more efficiently, achieve higher levels of fault tolerance, and supply faster access to data. Additionally, HDFS can scale to meet the requirements of data-intensive applications, allowing organizations to process data faster and more effectively.

HDFS achieves fault tolerance through data replication, where data is copied across multiple machines in the cluster. In case of a machine failure, the data can be retrieved from the replicated copy, ensuring data availability and consistency. Moreover, HDFS includes a NameNode and multiple DataNodes that enable it to manage metadata and data storage. The NameNode maintains the file system namespace and manages access to files, while the DataNodes store data and respond to read and write requests.

In terms of performance, HDFS can provide high aggregate bandwidth across the cluster by distributing data processing and storage across multiple machines. This design allows HDFS to process data in parallel, making it well-suited for handling large data sets on time.

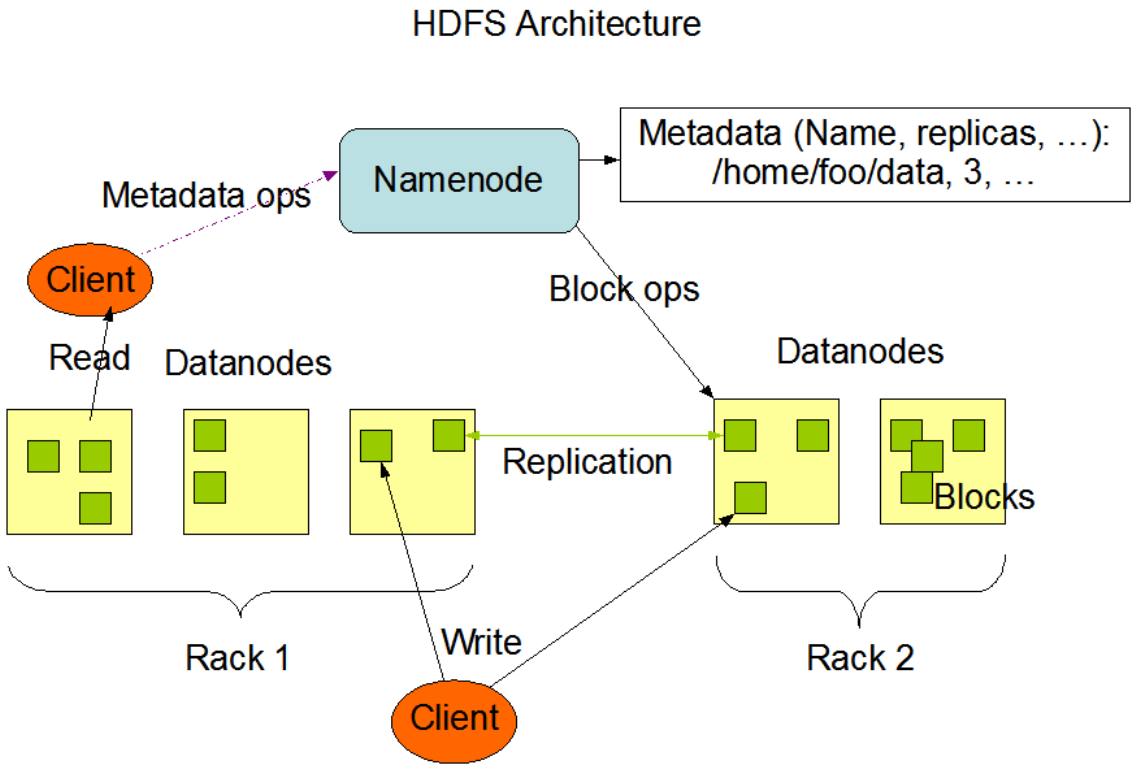
3. How different components in your case study are interlinked? Include an architectural diagram. If downloaded from the Internet mention the source and explain the working.

In HDFS, the file system namespace and the management of data storage are separated between two key components - NameNode and DataNodes, and the data itself is divided into blocks.

The NameNode is the central controller in HDFS, which manages the file system namespace and keeps track of the location of data blocks on the DataNodes. When a client requests data, the NameNode is first contacted to retrieve the metadata related to the requested file, including the location of the blocks that make up the file. The NameNode then sends this information back to the client, allowing it to directly contact the DataNodes and retrieve the necessary data.

The DataNodes are responsible for storing and retrieving data blocks. Each DataNode stores a subset of the data blocks and periodically reports back to the NameNode with the block's status, including the health and availability of the node. DataNodes are able to work together to ensure the availability and reliability of the data by replicating the data blocks across multiple nodes. This replication process is transparent to the client applications and is managed by the NameNode.

The following architectural diagram shows the relationship between NameNode, DataNodes, and Blocks in HDFS:



Source: Apache Hadoop [1]

4. What architectural style has been used in your case study? In your opinion, why is it a preferred option over other architectural styles?

HDFS follows the “Manager/Subordinate” architectural style, where the NameNode acts as the “manager”, and the DataNodes act as “subordinates”.

We believe HDFS opted for this style because it provides centralized control and management of the file system, while the data storage and processing are distributed among multiple nodes. This style provides fault tolerance and scalability, as multiple DataNodes can be added to the cluster, and the NameNode can manage the metadata and the namespace efficiently.

5. Are there any videos explaining your case study? If so, please mention the source.

As HDFS is a widely used distributed system, there are plenty of videos showing its components and operations. We recommend two introductory videos to this topic, provided by YouTube channel “Simplilearn” [2] and “edureka!” [3].

Those videos are a part of a series detailing the concept and implementation of HDFS.

6. Revisit the properties of distributed systems discussed in the class, how are the properties considered in your case study? For instance, how it handles failures, transparency, etc.

Due to the nature of distributed systems, they need to be designed and constructed in a way that ensures efficient operation. HDFS is not an exception to this rule, and has many properties to handle this very nature. Some notable ones are:

- Performance: It provides high-performance data access across the distributed system through the use of parallelism, where data blocks can be read from multiple DataNodes at the same time, and through the use of data locality, where data processing tasks are scheduled on the same machine where the data resides, reducing network traffic.
- Fault Tolerance: It is fault-tolerant, and can continue to operate effectively even in the event of machine failures or other types of failures. This is possible through the use of replication, where data blocks are copied to multiple DataNodes to ensure that they are available even if one of the nodes fails.
- Scalability: It handles big volumes of data across a large number of machines. The use of NameNode and DataNodes in HDFS allows the system to scale horizontally, adding new machines to the cluster as needed to meet the demands of the data processing workload.
- Consistency: It ensures data consistency across the distributed system. The NameNode acts as the central controller for the file system namespace, ensuring that all clients see a harmonious view of the data. The replication of data blocks across multiple DataNodes also promises that the data is consistent across the system.
- Transparency: It is transparent, abstracting away the details of the underlying hardware and software to the user, making it easier to use. HDFS provides a

uniform view of the file system namespace, regardless of the physical location of the data.

7. The key requirements for any distributed system are scalability, reliability, performance, and openness. Provide three examples of where these requirements might be in conflict with each other, how does your case study deal with these potential conflicts?

As with most pieces of software, errors and conflicts are not totally avoidable for HDFS. However, the system was designed with many robust mechanisms to address these issues. Some examples of those conflicts and how HDFS tackles them are:

a) Scalability and Performance: As the number of DataNodes increases in the HDFS cluster, the system's scalability improves, but it might impact the performance of the system. To address this issue, HDFS adopts a distributed architecture to store and process data locally on the nodes. This approach optimizes performance by reducing data movement across the network while maintaining scalability by adding more nodes to the cluster as required.

b) Scalability and Reliability: Adding more nodes to the HDFS cluster enhances scalability, but it also increases the likelihood of node failures. HDFS ensures reliability by replicating data across multiple DataNodes, so even if some nodes fail, the data is still available. However, replicating data can impact scalability in return since it necessitates additional storage and network bandwidth. HDFS manages this conflict by providing configurable replication factors that can be adjusted based on the desired balance between scalability and reliability.

c) Performance and Transparency: HDFS provides high performance by processing and storing data locally on the nodes, but this can limit the system's transparency. As the data is distributed across several nodes, accessing and processing the data can be more complex than on a single machine. To resolve this conflict, HDFS provides APIs and interfaces that enable clients to interact with the system seamlessly. Furthermore, HDFS supports a wide range of applications and tools that can access and process data in different formats, which enhances transparency.

8. Does your case study include multiple data storage facilities? If so, why?

HDFS indeed does have multiple data storage facilities, which are reflected in the use of different DataNodes. The data in HDFS is divided into many blocks, and each DataNode stores one of these blocks. Each block is replicated on multiple DataNodes for redundancy, ensuring that data remains available even in the event of node failure. The replication factor is configurable, with a default value of three, which means that each block is stored on three different DataNodes.

The use of multiple data storage facilities is a key feature of HDFS that provides the following benefits:

- Tolerance: In case one node fails, HDFS can replicate the data stored on that node to another nodes, ensuring the availability of such data.

- Performance: HDFS can access data in parallel, using multiple nodes to manipulate data simultaneously.
- Scalability: When the amount of data in the cluster grows, more nodes can be added to the cluster, and the data can be distributed across the new nodes.

References

[1] Apache.org. (2019). *HDFS Architecture Guide*. [online] Available at: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[2] Simplilearn (2020). *What Is HDFS?* | *HDFS Architecture* | *HDFS Tutorial For Beginners* | *HDFS In Hadoop* | *Simplilearn*. [online] Available at: https://www.youtube.com/watch?v=nRX4_3qf3rc&t=466s&ab_channel=Simplilearn [Accessed 16 Feb. 2023].
[3] edureka! (2018). *What is HDFS* | *Hadoop Distributed File System (HDFS) Introduction* | *Hadoop Training* | *Edureka*. [online] Available at: https://www.youtube.com/watch?v=GJYEsEEfjvk&ab_channel=edureka%21 [Accessed 16 Feb. 2023].