The University of Western Ontario
Department of Computer Science

Apache Hadoop and Hive

Group Members:
Redacted

CS4411 - Databases II (Advanced Databases)

Final Project Report

Dec 9, 2023

Contents

1 Objectives

The objective of this report is to discover why Apache Hadoop & Hive are popular software in Database Management Systems. We will explore Hadoop's strengths in being able to quickly store and process large amounts of data. We will then analyze Apache Hive's SQL interface and see the benefits that it provides in conjunction with Apache Hadoop.

2 Architectural Analysis

2.1 Apache Hadoop

2.1.1 Design Principles

The design principle of hadoop is to allow large computations and data storage to be distributed across a cluster of machines, each contribute a part of the processing work, so that large amount of data can be processed with standard servers or computers. To achieve this, hadoop must also be reliable and easily scaled.

2.1.2 Components and Layers

There are 3 main components in hadoop:

HDFS(Hadoop Distributed File System): Responsible for the distribution of storage of data across nodes in the cluster. It consist of NameNode, which manage namespace and blocks, and DataNode, which contains the block replica.

YARN (Yet Another Resource Negotiator): Manages and schedules resources for applications in the cluster.

MapReduce: Handles the processing of data in parallel across the nodes in the cluster, utilizing the Map and Reduce tasks. It consist of JobTracker and TaskTracker, which are responsible for job scheduling and resource management, and task execution respectively.

2.1.3 Data Flow Mechanisms

Data is written into HDFS, and it is replicated across multiple nodes for parallel processing and fault tolerance. Then MapReduce jobs are executed, processing data where it resides in HDFS. Finally, processed results are written back to HDFS, and users can retrieve the results from there.

2.1.4 Scalability

Hadoop allows data to be broken down into small pieces which are processed independently, bigger data translates to more pieces. This gives hadoop a horizontal scalability, which means increasing processing and storage requirement can be met by increasing more nodes into the cluster. Therefore it is only limited by the size of clusters rather than the processing power of a single machine. Since it is easier to acquire more machines than to acquire a more powerful machine, this is one of the greatest strength of hadoop.

2.1.5 Fault Tolerance

Both HDFS and MapReduce have a failsafe mechanism. HDFS replicates data across nodes, providing fault tolerance in case of node failures, whereas MapReduce tasks are rerun on other nodes if a node fails during processing.

2.2 Apache Hive

2.2.1 Design Principles

The design principle for Hive is to be used along with Hadoop, and make it more accessible to users who are familiar with relational database. In order to do this, Hive provides a high-level query language named HiveQL, which is similar to SQL. This also achieves abstraction, enabling users to query data without knowledge about MapReduce code.

2.2.2 Components and Layers

UI – The user interface for users to submit queries and other operations to the system. Driver – The component which receives the queries and fetches results. Compiler – The component that parses the query from the driver, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore. Metastore – The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, and the corresponding HDFS files where the data is stored. Execution Engine – The component which executes the execution plan created by the compiler.

2.2.3 Data Flow Mechanisms

First, The user interacts with the UI to execute a query. Then the Driver creates a session for the query and sends it to the compiler for an execution plan. The compiler then gathers metadata about tables from the metastore. Metadata will then check query expressions and filter data based on conditions. After this, the compiler generates a plan, a series of stages with map/reduce jobs, metadata operations, or HDFS operations. Afterwards the execution engine submits these stages for processing. In each task (mapper/reducer), data is read from HDFS files using table-specific deserializers. Finally the output is processed through operator trees and written to temporary HDFS files using serializers.

2.2.4 Scalability

This has the same horizontal scalability as Hadoop for data. And for its development it has grown to support different Execution Engines such as Tez, or Spark.

2.2.5 Fault Tolerance

Same as Hadoop, Metadata are redundant, and the queries can be retried on another node if a task fails during execution.

3 Functionalities and Features

3.1 Data Storage Methods

3.1.1 Hadoop Distributed File System

The primary storage system used by Hadoop and Hive. It transfers data between nodes quickly to handle large amounts of data.

3.2 Query Optimization Techniques

3.2.1 Partitioning & Bucketing

To help with query performance and optimization, Hive uses partitioning and bucketing. By partitioning, it splits tables into more digestible pieces based on their columns. This helps query speed as it requires less data to be searched. While with bucketing, data is separated into buckets placed into different files in the Hadoop Distributed File System (HDFS; Hadoop's primary storage system). This allows more efficient querying of large sets of data, such as ones with JOIN operations and aggregations.

3.2.2 Vectorization - Vectorized Query Execution

Apache Hive's vectorized query execution is used to cut CPU usage considerably where scan, aggregates, and join operations are used. This is done by processing a block consisting of 1024 rows at a time rather than one row at a time. Each block's column is stored as a vector and then operations such as arithmetic and comparison operations are by iterating through a loop quicker than normal, as there will be no function called during the loop. This method of query optimization uses the CPU pipeline and cache memory to help speed up the operation. Vectorization leads to cutting down slow queries by using the CPU architecture more. Using "**set hive.vectorized.execution.enabled = true;**", will enable this feature to be used with Apache Hive.

3.2.3 Cost-Based Optimization (CBO)

Hive will decide how to add additional changes to help query cost, by change order of joins, recording table and/or column statistics, and creating a more optimized query plan. By using "**hive.cbo.enable=true**", you can enable hive to gather statistics.

hive.stats.autogather	gather table statistics
hive.stats.fetch.column.stats	gather column statistics
hive.compute.query.using.stats	generate query plan

3.2.4 Compression

Apache Hive supports different compression codecs to help reduce size of data in the HDFS. This helps with lowering data read and write times.

3.2.5 Parallel Execution

In Hive, the execution engine's job is to execute the queries. You can get Hive to translate the queries into formats that are readable by different execution engines, the different engines can process thae query differently. In Apache Tez, it can process data in memory without writing the results to disk, to help accelerate complex tasks.

3.2.6 Security Measures

- **Authentication and Access Control:** Needs to validate users identities and can allow different levels of access to certain users using the Kerberos technique. Guarantees that only authorized users can access the database.
- **Data encryption:** Data is protected via encryption by transforming the data into an unreadable format. Unauthorized access will not be able to understand the data in the database.
- **Audits and Monitoring:** Monitors the Hadoop ecosystem to detect unusual behaviour or potential security breaches. Logging and auditing tools can allow for quick discovery of security incidents

4 Methodology - Case Study

4.0.1 Starting Apache Hadoop & Hive

Installation Requirement:

1. Apache Hadoop 3.3.6

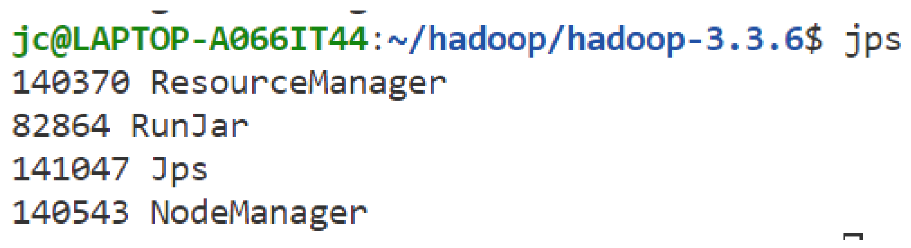
2. Java JDK (8 or 11)
3. Apache Hive 3.1.3

Install all the required softwares...

- **HDFS (Hadoop Distributed File System):** This is the storage component of Hadoop, which allows for the storage of large amounts of data across multiple machines. It is designed to work with commodity hardware, which makes it cost-effective.
- **YARN (Yet Another Resource Negotiator):** This is the resource management component of Hadoop, which manages the allocation of resources (such as CPU and memory) for processing the data stored in HDFS.

Starting Hadoop

- **start-dfs.sh:** starts namenode, datanode functions for HDFS
- **start-yarn.sh:** starts node manager and resource manger
- **start-all.sh:** alternatively, if you want to start everything at once
- use .**cmd** for Windows and **.sh** for Linux
- **jps:** check running daemons
- **hive:** starts hive



```
jc@LAPTOP-A066IT44:~/hadoop/hadoop-3.3.6$ jps
140370 ResourceManager
82864 RunJar
141047 Jps
140543 NodeManager
```

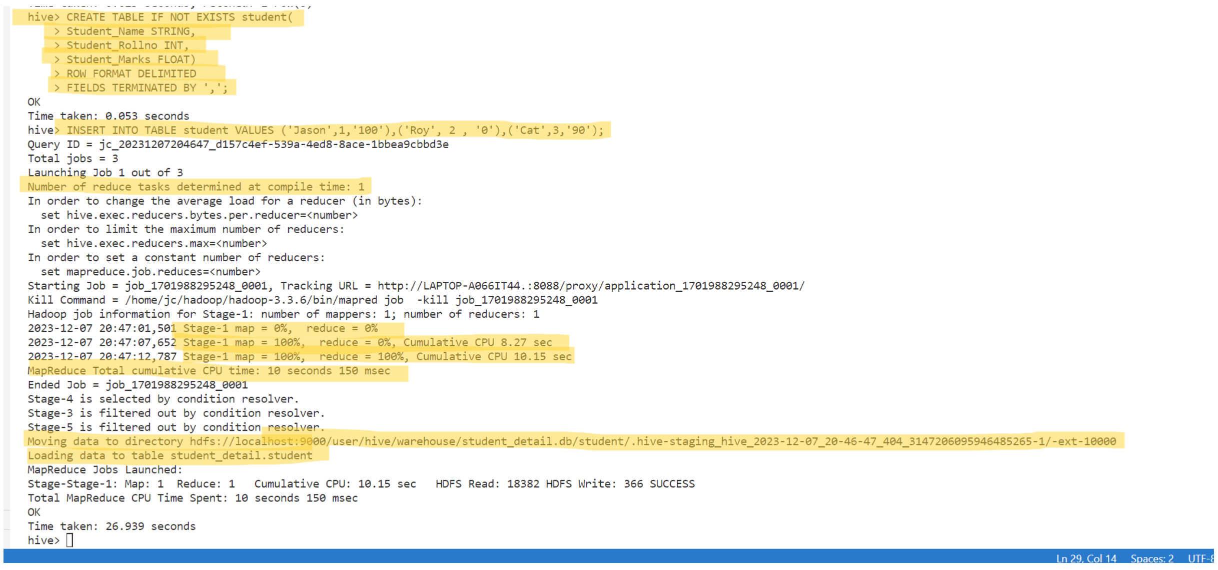
Figure: JPS command shows running ResourceManager and NodeManager.

After running your start commands you will see a hive session ID, meaning you have started running Hive in Hadoop and can now run SQL commands to create and modify databases.

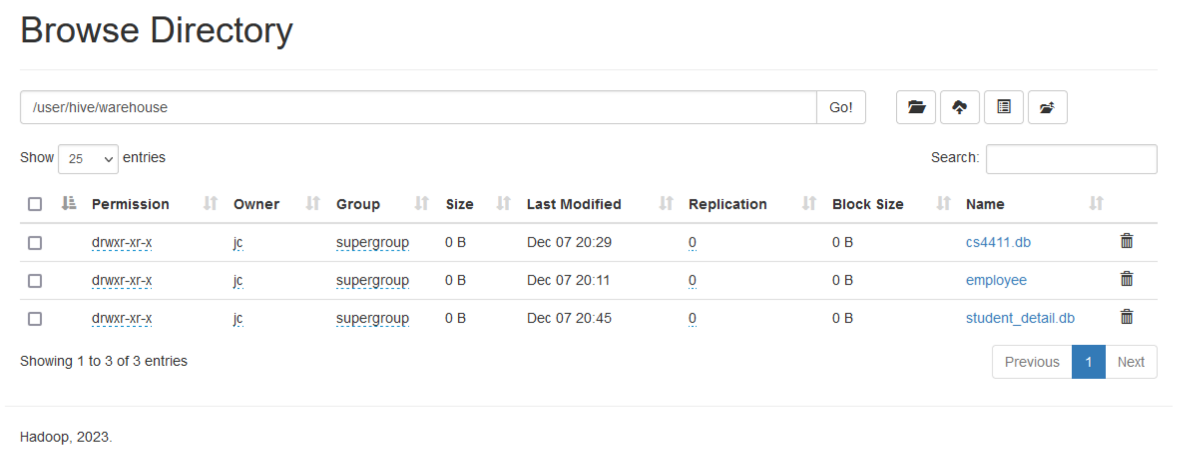
4.0.2 Using Hadoop & Hive

Figure: creating a table called "students" and adding data to the table.

- CREATE DATABASE student_details;
- USE student_detail;
- CREATE TABLE IF NOT EXISTS student(Student_Name STRING, Student_No INT, Student_Mark FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ’,’;
- INSERT INTO TABLE student VALUES (’student1’,1,’100’),(’student2’,2,’0’),(’student3’,3,’90’);



```
hive> CREATE TABLE IF NOT EXISTS student(
    > Student_Name STRING,
    > Student_Rollno INT,
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 0.053 seconds
hive> INSERT INTO TABLE student VALUES ('Jason',1,'100'),('Roy', 2 , '0'),('Cat',3,'90');
Query ID = jc_20231207204647_d157c4ef-539a-4ed8-8ace-1bbea9cbbd3e
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1701988295248_0001, Tracking URL = http://LAPTOP-A066IT44.:8088/proxy/application_1701988295248_0001/
Kill Command = /home/jc/hadoop/hadoop-3.3.6/bin/mapred job  -kill job_1701988295248_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-12-07 20:47:01,501 Stage-1 map = 0%,  reduce = 0%
2023-12-07 20:47:07,652 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.27 sec
2023-12-07 20:47:12,787 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 10.15 sec
MapReduce Total cumulative CPU time: 10 seconds 150 msec
Ended Job = job_1701988295248_0001
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/student_detail.db/student/.hive-staging_hive_2023-12-07_20-46-47_404_3147206095946485265-1/-ext-10000
Loading data to table student_detail.student
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.15 sec   HDFS Read: 18382 HDFS Write: 366 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 150 msec
OK
Time taken: 26.939 seconds
hive>
```



Browse Directory

/user/hive/warehouse

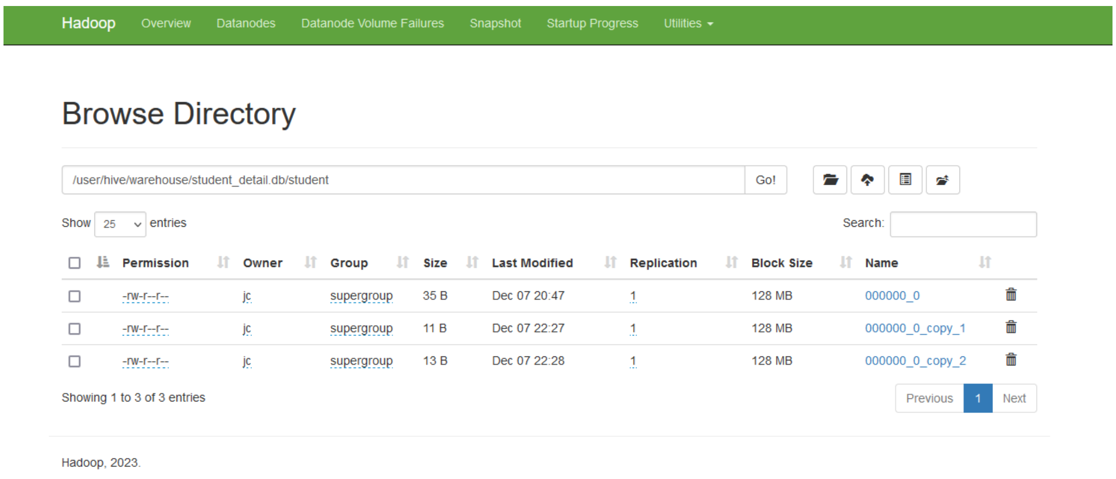
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	drwxr-xr-x	jc	supergroup	0 B	Dec 07 20:29	0	0 B	cs4411.db
☐	drwxr-xr-x	jc	supergroup	0 B	Dec 07 20:11	0	0 B	employee
☐	drwxr-xr-x	jc	supergroup	0 B	Dec 07 20:45	0	0 B	student_detail.db

Showing 1 to 3 of 3 entries

Hadoop, 2023.

After we input an insert command, we see Hadoop attempting to optimize the transactions by splitting the query job into smaller parts. The mapper's job is to process input data and put them into chunks of data. The reducer will then take that data and then produce a new output to be stored in the HDFS. We see that it is moved into the HDFS towards the end. We did not have a large enough query for hadoop to take advantage of our perforance to split up our query, so there is only 1 map and 1 reducer allocated to this query.

In the Directory below, we are viewing the Hadoop HDFS details where it displays the current DBs in the Hive warehouse. Webpage loacated in: http://localhost:9870 when running Hadoop.



Hadoop | Overview | Datanodes | Datanode Volume Failures | Snapshot | Startup Progress | Utilities

Browse Directory

/user/hive/warehouse/student_detail.db/student Go!

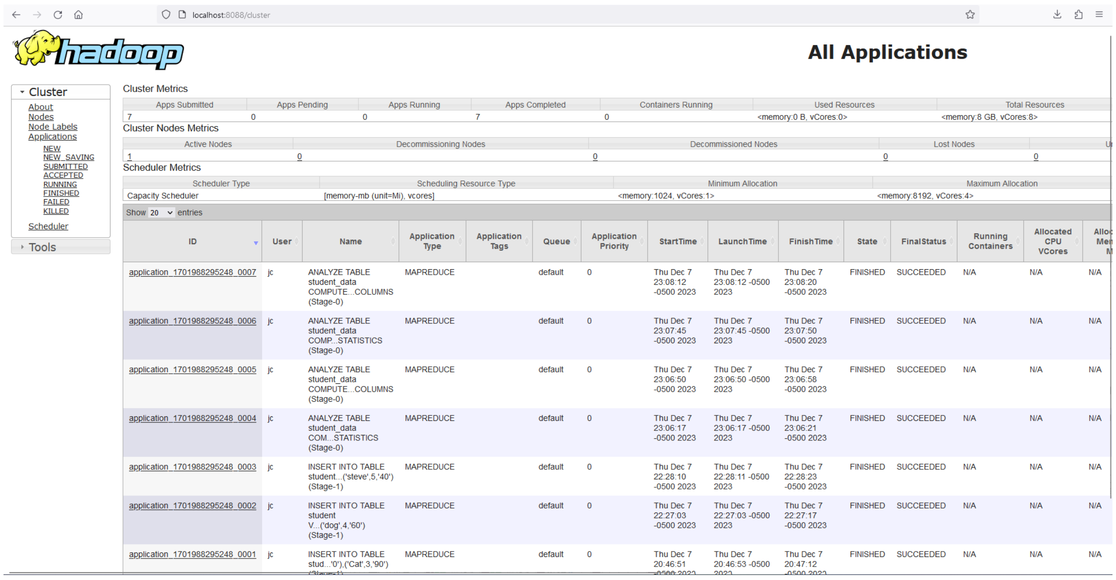
Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
☐	-rw-r--r--	jc	supergroup	35 B	Dec 07 20:47	1	128 MB	000000_0	
☐	-rw-r--r--	jc	supergroup	11 B	Dec 07 22:27	1	128 MB	000000_0_copy_1	
☐	-rw-r--r--	jc	supergroup	13 B	Dec 07 22:28	1	128 MB	000000_0_copy_2	

Showing 1 to 3 of 3 entries Previous 1 Next

Hadoop, 2023.

Here we have the Hadoop directory showing the different transactions/queries we have made on table student. Here I have made 3 different insert queries into the table student. Hadoop keeps the log of inserts.



localhost:8088/cluster

All Applications

Cluster: About, Nodes, Node Labels, Applications (NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), Scheduler; Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources
7	0	0	7	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
1	0	0	0

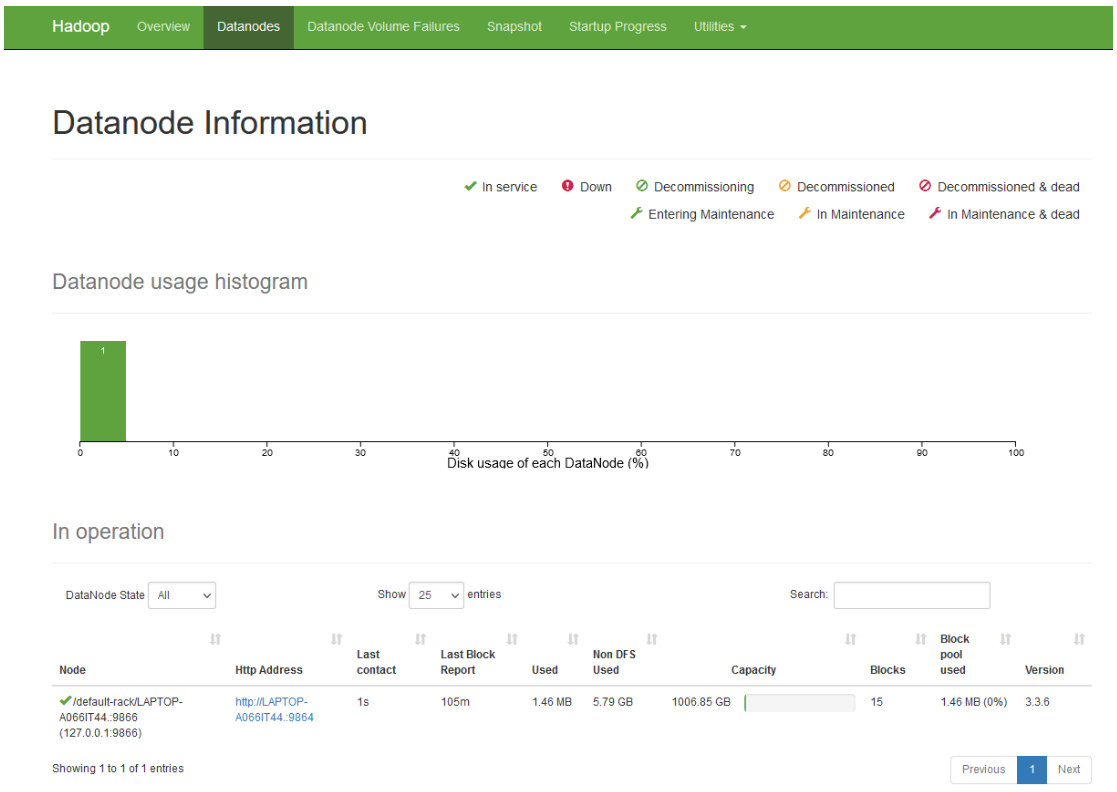
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores
application_1701988295248_0007	jc	ANALYZE TABLE student_data COMPUTE...COLUMNS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:08:12 -0500 2023	Thu Dec 7 23:08:12 -0500 2023	Thu Dec 7 23:08:20 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0006	jc	ANALYZE TABLE student_data COMP...STATISTICS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:07:45 -0500 2023	Thu Dec 7 23:07:45 -0500 2023	Thu Dec 7 23:07:50 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0005	jc	ANALYZE TABLE student_data COMPUTE...COLUMNS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:06:50 -0500 2023	Thu Dec 7 23:06:50 -0500 2023	Thu Dec 7 23:06:58 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0004	jc	ANALYZE TABLE student_data COM...STATISTICS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:06:17 -0500 2023	Thu Dec 7 23:06:17 -0500 2023	Thu Dec 7 23:06:21 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0003	jc	INSERT INTO TABLE student...('steve',5,'40') (Stage-1)	MAPREDUCE		default	0	Thu Dec 7 22:28:10 -0500 2023	Thu Dec 7 22:28:11 -0500 2023	Thu Dec 7 22:28:23 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0002	jc	INSERT INTO TABLE student V...('dog',4,'60') (Stage-1)	MAPREDUCE		default	0	Thu Dec 7 22:27:03 -0500 2023	Thu Dec 7 22:27:03 -0500 2023	Thu Dec 7 22:27:17 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0001	jc	INSERT INTO TABLE stud...'0'),('Cat',3,'90')	MAPREDUCE		default	0	Thu Dec 7 20:46:51	Thu Dec 7 20:46:53 -0500	Thu Dec 7 20:47:12	FINISHED	SUCCEEDED	N/A	N/A	N/A

Here we have the Hadoop webpage (http://localhost:8088) logging all of the queries made onto our databases. We have made a few INSERT and ANALYZE inputs into our table student. Hadoop logs the queries and tells you the method it used to help optimize the query. We see that it used only MapReduce in these successfully finished queries.



Datanode Information

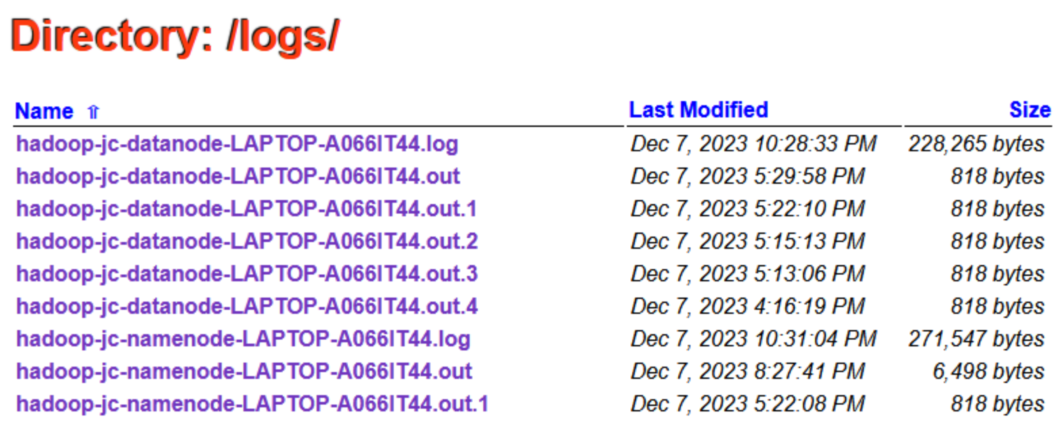
Datanode usage histogram

In operation

Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity		Blocks	Block pool used	Version
✓/default-rack/LAPTOP-A066IT44.:9866 (127.0.0.1:9866)	http://LAPTOP-A066IT44.:9864	1s	105m	1.46 MB	5.79 GB	1006.85 GB		15	1.46 MB (0%)	3.3.6

Showing 1 to 1 of 1 entries

Hadoop webpage (http://localhost:8088) also displays the current usage statistics such as Disk usage of each DiskNode, each individual active or decommissioned node, the node's last contact, used storage space, Hadoop version, and blocks allocated.



Directory: /logs/

Name ⇑	Last Modified	Size
hadoop-jc-datanode-LAPTOP-A066IT44.log	Dec 7, 2023 10:28:33 PM	228,265 bytes
hadoop-jc-datanode-LAPTOP-A066IT44.out	Dec 7, 2023 5:29:58 PM	818 bytes
hadoop-jc-datanode-LAPTOP-A066IT44.out.1	Dec 7, 2023 5:22:10 PM	818 bytes
hadoop-jc-datanode-LAPTOP-A066IT44.out.2	Dec 7, 2023 5:15:13 PM	818 bytes
hadoop-jc-datanode-LAPTOP-A066IT44.out.3	Dec 7, 2023 5:13:06 PM	818 bytes
hadoop-jc-datanode-LAPTOP-A066IT44.out.4	Dec 7, 2023 4:16:19 PM	818 bytes
hadoop-jc-namenode-LAPTOP-A066IT44.log	Dec 7, 2023 10:31:04 PM	271,547 bytes
hadoop-jc-namenode-LAPTOP-A066IT44.out	Dec 7, 2023 8:27:41 PM	6,498 bytes
hadoop-jc-namenode-LAPTOP-A066IT44.out.1	Dec 7, 2023 5:22:08 PM	818 bytes

Hadoop also has logs to help with data protection. You could download the logs and check your databases activity.

5 Results

5.0.1 Findings

- **Scalable:** We found that with Hadoop's interface, we could see the different systems that would be able to connect and work on our databases together. This would mean that Hadoop could be scaled larger so that multiple systems could

work simultaneously on the database. The data statistics meant that it could scale to larger databases that manage very large datasets
- **Fault tolerance and data protection:** The transaction logs from Hadoop UI were able to give us an idea of how it tracked queries done on our databases and tables. This logged our queries and reported if they were successful, if they finished or cancelled. This gave us a better idea of how Hadoop is fault tolerant and can protect data in case of failure.
- **Cost-effective:** We found that running Apache Hadoop and Hive was quite cost-effective as it is free and open-source. We were able to run it on our systems to test out some of its features.
- **Flexibility:** We found that Hadoop was quiet flexible with data. It being able to handle structured and unstructured data in its processing.

The results we found were that Apache Hadoop was a powerful framework and Hive was an efficient tool in processing queries. We found these cost-effective tools to be quite handy and impressive as we used them to make our basic databases, tables, and queries.

With Hive, we were able to create SQL databases and tables, insert data, and see the MapReduce performance statistics for queries performed. This was quite handy as SQL is quite simple and commonly used in databases. This and the open-source feature of Hive makes this incredibly useful for people looking to start using DBMS softwares.

In Hadoop, we were able to use HDFS, YARN, and a few other features. These features allowed us to combine Hive and it's SQL commands to view what we had created. We created databases and tables in Hive and they were automatically displayed in our Hadoop localhost webapge. This is something very handy for many DBMS users as a visual repsentation of their current database. Seeing logs of our queries on our table was quite important in case of a failure, we would be able to see if it processed successfully or if it failed. This gives us more confidence in Hadoop as we can see it logging the data. One of Hadoop Hive's features is fault-tolerance, and this gives us a visual view of it.

5.0.2 Challenges

- **complexity:** We found out that Hadoop was quite complex and that it needed a lot of tuning before you can really use it for its true potential. Using it proved that a GUI was needed as the local webpage for Hadoop was not quite sufficient for managing the database. Finding a GUI would greatly benefit Hadoop.

We had many challenges in getting Apache Hadoop and Hive working, and creating enough data to test the larger features of the softwares. The first issue was installation. We first tried to install on Windows by following a variety of guides on Youtube, Stackoverflow, Google, etc. Unfortunately we would encounter a variety of errors when running on Windows such as Hadoop being unable to find our datanode/namenode, issue with figuring out which versions of Java, Hadoop and Hive we needed to run and some files malfunctioning were just a few of the many issues we had with installing

Hadoop and Hive working on Windows. We then figured out we can run it on WSL (Windows Subsystem for Linux) and managed to get it running with a lot less issues.

After that the next challenge was getting large datasets to test the major features of Hadoop and Hive. We were not able to get a large enough dataset to test, as Hadoop deals with thousand and millions of rows of data. We tested both softwares with small databases of a few tables and rows of data. It was enough to see Hadoop's Mapper and Reducer on each query implemented onto the tables in our databases.

6 Resources & Tools

6.0.1 Code

- start-dfs.sh
- start-yarn.sh
- start-all.sh
- jps
- hive
- CREATE DATABASE student_details;
- USE student_detail;
- CREATE TABLE IF NOT EXISTS student(Student_Name STRING, Student_No INT, Student_Mark FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
- INSERT INTO TABLE student VALUES ('student1',1,'100'),('student2',2,'0'),('student3',3,'90');

6.0.2 Download Resources

- Java: `https://www.oracle.com/java/technologies/downloads/`
- Apache Hadoop 3.3.6: `https://hadoop.apache.org/`
- Apache Hive 3.1.3: `https://hive.apache.org/`

References