Rohan Pancholi
N0982031

Distributed Database Engineering

COMP30261

Laboratory Report

Rohan Pancholi
N0982031

Rohan Pancholi
N0982031

Table of Contents

1. Introduction 3

1.1 Hadoop Ecosystem Overview 4

1.1.1 HDFS 4
1.1.2 YARN 5
1.1.3.MapReduce 5
1.1.4 Apache Spark 5

2. Tasks and Implementation 6

2.1 Installation and Configuration of Ubuntu 20.0.4 6

2.2 Installation and Configuration of Java_JDK 8u251 on Virtual Machine Environment 6

2.3 Installation and Configuration of Hadoop 2.9.0 6

2.3.1 Core-site.xml 7
2.3.2 Hdfs-site.xml 7
2.3.3 Mapred-site.xml 7
2.3.4 Yarn-site.xml 7
2.3.5 Slaves 7

2.4 Configuration of a Multi-Node Hadoop Setup 8

2.5 Installation of Eclipse 2020-06 4.160 9

2.6 Writing MapReduce Java Scripts 9

2.6.1 RoadMapper.java 9
2.6.2 RoadReducer.java 9
2.6.3 RoadDriver.java 10

2.7 Running MapReduce Java Scripts 10

2.8 Installation of Apache Spark 10

2.9 Results 10

3. Critique, Discussion and Summary 11

4. References 12

5. Appendix 13

Rohan Pancholi
N0982031

1. Introduction

The objective of this task is the process and analyse the road safety data for the year 2018 using the 'RoadSafetyData_Casualties_2018.csv' dataset. The dataset at hand contains multiple attributes and figures about the road safety data, figure 1 shows the header rows of the dataset with figure 2 giving a brief description of the attributes.

The primary goal at hand is to utilise the Hadoop Framework and more specifically, leverage Java MapReduce, to analyse the dataset. The analysis aims to count the total number of occurrences of accidents indexed by each band of causality severity. This process will be conducted in a distributed computing environment, requiring the configuration of a multi-cluster system with a minimum of three nodes to efficiently handle data processing whilst all being completed through the use of virtual machines.

The Global Status Report by Margaret Chan on Road Safety underscores the critical importance of road safety on a global scale, highlighting the vast health and developmental challenges posed by road traffic injuries. It reveals that over 1.2 million people die annually on the world's roads, with 20 to 50 million suffering non-fatal injuries. The report emphasises that the majority of fatalities occur in low- and middle-income countries, despite these countries having less than half of the world's vehicles. It highlights the necessity of adopting comprehensive road safety measures to mitigate this global public health problem (Chan, Global status report on road safety)

Big data analytics can assist in playing a major role in tackling road safety concerns. Statistical techniques and tools are being applied on large scales to discover patterns and trends to predict and prevent future accidents (Rai, 2023). Analysing large-scale data such as road conditions, weather, and time of day can help predict accidents in certain areas at certain times. There are multiple solutions for big data analytics including Apache Hadoop and spark and Cassandra or mongoDB.

Apache Hadoop is an open-source framework which can help process large scale datasets. Noteworthy for its distributed processing, scalability, fault tolerance, it encompasses components like HDFS, YARN, and MapReduce.

In distributed processing, as the name implies, tasks and data are spread across multiple interconnected computers or servers known as nodes. Each node contributes towards the processing and sharing of the workload. For example, if you wanted to count the number of words in a large portion of text, this could initially be done by counting each word chronologically throughout the text. However, with access to multiple nodes, the portion of text could be broken down into smaller chunks such as paragraphs or sentences depending on how many nodes there are and worked on in parallel with other nodes.
As the nodes are all interconnected, they can all work together simultaneously, allowing for a significant increase in the systems performance. Parallel processing is crucial for big data projects where petabytes or terabytes of data need to be analysed as quickly as possible.
Scalability is another important feature within Hadoop and distributed processing. More nodes can be incorporated into to the network to enhance processing capabilities and large-scale data workloads. Either vertical (improving node resources) or horizontal (addition of more nodes) scalability can be accommodated working hand in hand with parallelism as more nodes means data can be split into smaller loads.
Due to all nodes concurrently working together, if one node fails the system can continue to operate as all nodes are interconnected, this is shown in figure 5. This feature is vital in big data projects where failure or loss of data may lead to significant setbacks.

Through installations and configurations I aim to provide the following key features

- Distributed storage (HDFS)
- Parallelism
- Scalability
- Fault tolerance
- Web based monitoring of cluster

1.1 Hadoop Ecosystem Overview

Hadoop is an open-source framework widely used for distributed storage and processing of large datasets. Prior to the emergence of Hadoop, managing and analysing large scale data posed significant hurdles. Hadoop has since become the remedy for these challenges revolutionising data storage and processing. Hadoop offers cheaper, faster and better results compared to other frameworks offering similar capabilities. Hadoop consists of three main components; HDFS, YARN, MapReduce

1.1.1 HDFS

HDFS (Hadoop Distributed File System) is a component of the Hadoop Ecosystem as shown in figure 3. It is the most important component because the entire eco system depends upon it. HDFS is based upon Google File System, which is a file system which runs on many computers to provide large amounts of storage. HDFS is the main storage layer and is ideal if you want to store petabytes of data and distribute processing over multiple machines simultaneously (Giri, Overview of Apache hadoop ecosystem). It uses a block structured file system, where files are broken down into small chunks known as blocks, a physical representation of data ranging from 64mb for Hadoop 1x or 128mb for Hadoop 2x. Blocks are then replicated for fault tolerance and distributed over data nodes, which are machines which are used to store the data blocks.

As seen in figure 4, Hadoop's HDFS follows a master/slave architecture that is key to its functionality and efficiency in handling large datasets. At the heart of this architecture are the NameNode and DataNodes, each playing a crucial role in the management and storage of data.
The NameNode acts as the master server and is central to the HDFS file system. It maintains the directory tree of all files in the system and manages the metadata for these files. This includes information such as permissions, modification and access times, and the structure of the file system itself. Importantly, the NameNode keeps track of where within the cluster the data for each file is located, orchestrating the placement and replication of data blocks across DataNodes. Originally, the NameNode represented a single point of failure in HDFS, but advancements in Hadoop 2.x and later versions introduced High Availability configurations, which use multiple NameNodes in an active/passive setup to provide failover capabilities, thereby mitigating this risk.

DataNodes are the slave nodes that physically store the data. When files are written to HDFS, they are broken down into blocks, and these blocks are distributed and stored across multiple DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. They also perform block creation, deletion, and replication under the direction of the NameNode. To ensure the integrity and availability of data, DataNodes send regular reports to the NameNode, including heartbeats (to signal their health and status) and block reports (to update the NameNode on the blocks they are storing).
The Secondary NameNode is somewhat misleadingly named, as it does not serve as a backup to the NameNode in case of failure. Instead, its role is to support the NameNode by processing its

metadata. It periodically merges the namespace image with the edit log to ensure that the log does not grow indefinitely, which helps in maintaining the efficiency of the NameNodes operations.

HDFS is commonly used in conjunction with other Hadoop ecosystem components such as MapReduce or YARN, HDFS stores the data whilst YARN divides the tasks and MapReduce processes the data.

1.1.2 YARN

YARN (Yet Another Resource Negotiator) is another component within the Hadoop Ecosystem and is the resource management layer. YARN keeps track of all the resources (CPU and Memory) of the machines in the network and runs the applications. Any application which wants to run in a distributed fashion would interact with YARN

1.1.3.MapReduce

MapReduce is the application layer in the Hadoop ecosystem. MapReduce takes away the complexity of the distributed programming by splitting it into two parts, Map and Reduce.

1.1.3.1 Map

The map function takes the input data and converts it into key / value pairs. The nature of these pairs varies on the specific problem. Carrying on from the word count example, the Map function reads in a document and outputs a key/value pair for each word, where the key is the word itself and the value is 1 (indicating a single occurrence of the word). Overall the Map phase aims to organise the data in such a way that all the data belonging to a single key is brought together. This is essential for the next phase of processing, where data will be aggregated or summarised.

1.1.3.2 Reduce

The Reduce function takes the intermediate key/value pairs produced by the Map phase, groups them by key, and processes each group in some way to produce the final output. Continuing with the word count example, the Reduce function would take all key/value pairs for a given word, sum up the values (to get the total count for that word), and output a final key/value pair with the word and its total count.
The goal of the Reduce phase is to aggregate, summarise, filter, or transform the intermediate data into the final output format that solves the problem at hand.

1.1.4 Apache Spark

Apache Spark is an open-source, distributed computing system designed to provide a fast and versatile cluster-computing framework for big data processing and analytics. It emerged as an improvement over the traditional MapReduce model, aiming to enhance the efficiency and accessibility of data processing tasks.
In terms of the problems it tackles, Spark addresses the slow performance of MapReduce by employing in-memory processing, reducing the reliance on disk I/O. It simplifies the complexity of data processing by offering high-level APIs and a unified framework, making it more accessible to developers. Spark's versatility addresses the limited workload support of MapReduce, as it can handle batch processing, interactive queries, streaming, and machine learning within the same framework. Additionally, Spark is adept at handling iterative machine learning algorithms, overcoming the challenges posed by traditional MapReduce models involving multiple maps and

reduce stages. In summary, Apache Spark enhances big data processing by providing speed, ease of use, versatility, fault tolerance, and support for various workloads within a unified and efficient framework.

2. Tasks and Implementation

There were 7 main tasks to work to my goal, each task had multiple sub tasks associated with it involving either installations or further configurations of files.
Each task, in attempt to achieve my goal, carried equal weight. Sequentially, the tasks build upon each other making each task dependent on the previous, requiring it to be fully installed, configured and working before starting the next installation. Hence, I aim to take a slow but methodological approach making sure I complete each installation correctly preventing the cause more errors further down the line.

2.1 Installation and Configuration of Ubuntu 20.0.4

The first installation made was Ubuntu version 20.0.4, this was installed from a git Repository. Ubuntu runs Linux which is the primary operating system used for the rest of this project. The set-up of the virtual machine came next and as seen in figure 6, I selected Linux and ubuntu-64 as the guest operating system with the resources allocated as shown in figure 7. The basic utility tools were installed onto Linux after it had finished setting up, these included : sudo apt-get update, sudo apt-get install wget, sudo apt-get install openssh-server, sudo apt-get install vim, sudo apt-get install open-vm-tools. These utility tools streamline various tasks, from file management to system administration, ultimately improving productivity and efficiency in computing tasks. This task was replicated three times to create three virtual machines, L1, L2, and L3 each with the same starting environment and resource allocations.

2.2 Installation and Configuration of Java_JDK 8u251 on Virtual Machine Environment

Installation of Java development kit was done through the virtual machine on the guest operating system. The version of java installed was version “8u251” and was installed through a git repository for compatibility reasons and is a prerequisite for Hadoop. Java Development Kit (JDK) includes the java runtime environment (JRE), an interpreter (Java), a compiler (Javac), an archiver (jar) and a documentation generator (Javadoc). The java development kit was installed under the file path ‘/usr/liv/jvm/jdk.1.8.0_251’ and the ‘PATH’ variable, as shown in figure 8, was updated to include ‘$JAVA_HOME/bin’ to ensure java commands are accessible from anywhere in the terminal. Figure 9 shows the terminal command to verify java that is installed before installation and figures 10, 11, 12 show java being successfully installed on all 3 virtual machines.

2.3 Installation and Configuration of Hadoop 2.9.0

Hadoop version 2.9.0 was installed and used for stability reasons. The installation of Hadoop, as mentioned in the introduction, allows for the set-up of a single node or multi node configuration. The installation of Hadoop was fairly simple when compared to the configuration. Installation was set up within the directory ‘/usr/local/hadoop’, figures 13 – 15 show the directory before Hadoop was installed and figures 16 – 18 show the directory after Hadoop was installed, notice the change in total blocks.

Similar to java, we have to add the Hadoop path to the path variable, this can be seen in figure 19 where additional lines are added describing different paths to Hadoop file locations. As shown in figure 20, we also have to replicate this for the 'Hadoop-env.sh' file to add the file paths for java home and hadoop.

After creating a symbolic link (figures 21 – 23) , in case another version of Hadoop is installed in the future, I ran a command prompt to verify Hadoop had been installed correctly, this is shown in figure 24. After the verification, we can confirm hadoop has been fully installed on our virtual machines, the stage is to configure Hadoops core files under the location '/usr/local/hadoop/etc/Hadoop'.

The main files to be configured were core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml and slaves. Each file needs configuring to allow for the setup and successful operation of a Hadoop cluster, these files should be configured and replicated on each virtual machine with minor changes in variable names.

2.3.1 Core-site.xml

This configuration file contains Hadoop core settings, crucial for the installation and operation of Hadoop. It specifies the NameNode URI (Uniform Resource Identifier), which is essential for identifying the location of the NameNode within the cluster, which in my case is LR1, formatting of the NameNode will come at a later stage. This setup is fundamental for the Hadoop ecosystem as it informs all the Hadoop services and clients where to find the master node for HDFS operations.

2.3.2 Hdfs-site.xml

Dedicated to configuring HDFS (Hadoop Distributed File System) settings, this file allows you to specify the replication factor (dfs.replication), which determines the number of data copies to be maintained across the cluster. It also configures the directories for NameNode (dfs.namenode.name.dir) and DataNode storage (dfs.datanode.data.dir), which are crucial for defining where metadata and actual data are stored, respectively.

2.3.3 Mapred-site.xml

This file focuses on the configuration of MapReduce settings, including the framework name (mapreduce.framework.name), which indicates whether MapReduce jobs should run on YARN or some other framework. It also sets environmental variables for the MapReduce framework, optimising the execution environment for processing tasks.

2.3.4 Yarn-site.xml

This configuration file is specific to YARN (Yet Another Resource Negotiator) settings, including the resource manager address (yarn.resourcemanager.address), which is vital for specifying the location of the ResourceManager. ResourceManager is the authority that manages resources and job scheduling across the cluster. Correct configuration of this file is crucial for resource management and for enabling scalable processing.

2.3.5 Slaves

This is not a configuration file like the others but is a plain text file that lists the machines (one per line) within the cluster that each run a DataNode and a NodeManager (formerly known as a

TaskTracker). This file is essential for defining the nodes that will store data and execute tasks, therefore it is critical for the operational structure of the Hadoop cluster.

Following configuration of the files, the next task involves setting up SSH (secure shell) access. SSH is needed to allow for password less access for each node, enabling seemless and secure interactions between the master and worker nodes. In a nutshell, SSH provides a way to establish a secure and encrypted connection between a client (your computer) and a server (a remote computer) over an unsecured network, such as the internet. SSH access was completed on all three virtual machines by running command line code 'ssh-keygen –t rsa' to produce a public / private key pair which can be seen in figures 25 – 27. Finally, to copy the public key from LR1 to the other machines in the cluster, we use 'ssh-copy-id -i $HOME/.ssh/id_rsa.pub dde@LR2/3'.

2.4 Configuration of a Multi-Node Hadoop Setup

As mentioned earlier, I created 3 virtual machines (L1, L2, L3). All steps until now have been replicated amongst all three virtual machines. The formatting only needs to be done once on the NameNode to create the initial metadata which in our case is LR1. This is achieved by running the command line code 'hdfs namenode-format'. The output after formatting can be seen in figure 28 where the most important line 'storage directory /abc/name has been successfully formatted is shown. This line indicates that the storage directory '/abc/name' has been formatted meaning it has prepared the storage directory for use by erasing and existing data or structures within it and set up the necessary file system layout. To make some checks this is done correctly we can use ' ls -all /abc/name/current' to list all the files and directories in the file path, this can be seen in figure 29. We can also check the actual file path where the data is stored which can be seen in figure 30.

The first step taken to configure our multi node setup is to configure the '/etc/hosts' file, this file contains a line by line of the nodes host name aswell as the ip address of the node. By updating this file on all nodes we can ensure that each node can communicate with the others by name, without relying solely on IP addresses. This facilitates easier management and communication within the multi-node setup, as it allows nodes to reference each other using familiar hostnames instead of cryptic IP addresses. Additionally, maintaining consistency across all nodes' "/etc/hosts" files helps in avoiding potential DNS resolution issues and ensures seamless communication between the nodes in the network. The entry of the file is shown below

192.168.106.150 LR1
192.168.106.161 LR2
192.168.106.162 LR3

After the configuration of Java, Hadoop and SSH we can attempt to test our Hadoop daemons and start all the nodes within the cluster. This is achieved by 'start-all.sh'. This command starts two internal scripts 'start-dfs.sh' and 'start-yarn.sh'. The command 'start-dfs.sh' initiates the Hadoop Distributed File System (HDFS) services, responsible for storing and managing data across the cluster. On the other hand, 'start-yarn.sh' launches the Yet Another Resource Negotiator (YARN) services, which handle resource allocation and task scheduling for distributed processing within the Hadoop ecosystem. This code is used to start all the daemons in the Hadoop cluster in one go and is run from the name node which in this case is LR1. The daemons include services such as the NameNode – LR, DataNode – LR2, SecondaryNameNode – LR2 and ResourceManager – LR1.

A successful launch of the clusters can be seen in figure 31, as no passwords are required, and all nodes are successfully logged into. We double check the clusters are running correctly by launching

the web interface, this can be done by searching “LR1:50070” in the web address space. As seen in figure 32, the web interface shows us information about the nodes such as ; HTTP address, Capacity, Blocks, Block Pool used and the version of Hadoop it is running. Another measure we can take is to run JPS, JPS is a command used in Java environments, including Hadoop, to list Java processes running on a machine. It stands for "Java Virtual Machine Process Status Tool." When you run jps in a terminal, it provides a list of Java processes along with their respective Process IDs (PIDs).
In the context of Hadoop, jps can be used to verify whether various Hadoop daemons, such as NameNode, DataNode, ResourceManager, and NodeManager, are running on the cluster nodes. This command is handy for troubleshooting and monitoring Java-based applications and services. Figures 33 – 35 show jps running for all nodes in my cluster indicating which node is the DataNode, SecondaryNameNode, NameNode and ResourceManager

2.5 Installation of Eclipse 2020-06 4.160

Eclipse serves as the preferred programming environment for executing our MapReduce job. It offers a comprehensive development platform for Java, which serves as the primary language for crafting MapReduce programs, akin to the one we intend to implement later. Moreover, Eclipse extends its functionality by supporting plugins tailored for Hadoop and other prominent big data frameworks. These plugins streamline the process of developing, deploying, and managing Hadoop applications directly within the integrated development environment (IDE). Eclipse version 2020-06 (4.16.0) was installed from google drive for compatibility reasons and directly extracted into the downloads folder. The terminal command ‘./eclipse-inst’ is ran in order to install eclipse from the downloads file, this can be seen in figure 36 where the version of eclipse downloaded is shown.

2.6 Writing MapReduce Java Scripts

The input data used was the ‘RoadSafetyData_Casualties_2018.csv’. As mentioned in the introduction, the dataset contains data relating to road accidents in the year 2018. The main goal and task is to find out the total number of accident index occurrences in each band of causality severity.
The first step in achieving this was to create java scripts and folders containing the code to execute the map reduce job. The three files created were; RoadMapper.java, RoadReducer.java and RoadDriver.java. Each file works alongside each other to create and run a MapReduce job.

2.6.1 RoadMapper.java

The mapper file is the first phase in the map reduce process, it deals with splitting the input data into manageable pieces and processing each piece independently. As mentioned earlier on, the input data is in the form of key-value pairs, where they key is the information you have, and the value is the amount of times it occurs. These results are then sorted by the framework to prepare for the next phase.

2.6.2 RoadReducer.java

The reduce is the second stage in the map reduce process. After sorting the data, the values with the same key are grouped together and passed to the reducer. The reducer processes these values combining them to produce a smaller set of values or in some cases a single value. The output from the reducer is the final output of the map reduce job

2.6.3 RoadDriver.java

The Driver is the control centre of the MapReduce job. It configures the job, sets up the input and output formats, and submits the job to the cluster for execution. It's responsible for specifying the Mapper and Reducer classes, specifying the types of the key-value pairs for inputs and outputs, and finally, for initiating the MapReduce job. The Driver also handles the monitoring of the job and its execution progress, as well as the retrieval of the job's status and results upon completion.
In summary, the Mapper prepares the data, the Reducer processes the prepared data, and the Driver manages the entire process, ensuring that everything runs smoothly and efficiently. These components work together to allow for scalable and efficient processing of large data sets across multiple nodes in a cluster. The Java files (ROADMapper.java, ROADReducer.java, ROADDriver.java) need to be compiled into bytecode, which the Java Virtual Machine (JVM) can execute. This step transforms your human-readable Java code into machine-readable format. The creation of the "ROAD" folder for storing class files organises your compiled code neatly.

2.7 Running MapReduce Java Scripts

The final stage in this task is to run the MapReduce code on the dataset. To begin with this we first have to start the Hadoop clusters, as shown in figures 31 – 35, by running start-all.sh. Now that Hadoop is running without any errors we can copy the input file, in this case the road safety data. Hadoop operates on data stored in HDFS. Therefore, the input data file must be transferred to HDFS before the MapReduce job can process it. This ensures that the data is accessible across the cluster for distributed processing. After the job completion, it's important to verify that it ran successfully and produced the expected output. This involves checking the contents of the output directory in HDFS and reviewing the generated results. Successful execution indicates that the MapReduce job has correctly processed the input data according to the defined logic. To check the code ran succesfully we can check the specified directory to see the number of files within in. As shown in figure 36, we can see the folder ending in road out which contains two more files containing the results.

2.8 Installation of Apache Spark

Whilst Apache spark was not used in the analysis of the dataset, installing Apache Spark gave me good practise and allowed me to figure out the basics of the software and how it differs to MapReduce. Version Apache Spark 3.0.0 was installed on all three nodes as we cannot predict which node will be used by default, also by doing this we can have more control over task distribution, fault tolerance, resource utilisation and scalability. Both files, workers and slaves, under the '/spark/conf' path need to be configured with the node names which in this case is L1, L2, L3. By configuring these files we can manage the distributed computing resources efficiently. Updating the spark-env.sh file is also essential in configuring the environment in which the spark cluster operates. This file can be seen in figure 38 and enables us to set various environmental variables that influence the behaviour of the spark cluster. Each line of code has a different use and can be seen in the table in figure 39

2.9 Results

To check the results of the MapReduce we can either see them in terminal, as shown in figure 40, or through the web browser like in figure 41. The block pool ID (BP-2800321229-127.0.0.1-1709130388894) shown in figure 42 represents a collection of blocks within the HDFS. All the blocks that share this block pool ID belong to the same namespace, which means they are part of a single

HDFS file system. The block pool ID displayed in the GUI window matches that in the terminal. This confirms that the part-00000 file, for which the details are shown, is stored within the same HDFS as the blocks listed in the terminal. The consistency between the block pool IDs confirms that you are looking at the same HDFS namespace in both windows. The match between the block pool ID shown in the terminal and the GUI indicates that the file contents shown in the GUI are indeed physically stored within the blocks listed in the terminal.

3. Critique, Discussion and Summary

The project aimed to analyse a dataset related to accident indices and casualty severity, employing MapReduce, Hadoop, and Java. The primary achievement was the successful computation of the total number of accident indices across different severity bands. This outcome was reached after overcoming initial challenges related to the setup and configuration of the necessary tools. The repeated process of setting up and configuring the tools, despite being time-consuming and error-prone, was invaluable. It provided deep insights into the functionality and interdependencies of the tools involved. Understanding these nuances is critical, as it lays the foundation for troubleshooting and optimising data processing tasks in complex systems. The project succeeded in achieving its analytical goals, demonstrating the effectiveness of using MapReduce, Hadoop, and Java for data processing tasks. However, the setup and configuration process proved to be a significant challenge.
The complexity and length of the setup process introduced numerous opportunities for errors, which, when occurred early on, had cascading effects on the project's progress. Identifying and rectifying these errors was difficult, often necessitating a complete restart of the setup process. While the analytical objectives were met, the project highlighted the importance of a meticulous and precise setup process for data processing tools. The difficulties encountered underscore the need for a more streamlined approach to installation and configuration, which could mitigate the risk of errors and improve efficiency. Additionally, the reliance on MapReduce and Hadoop, while effective, also points to the potential benefits of exploring more modern and advanced tools like Apache Spark for future projects. To advance the work further, integrating Apache Spark could provide a more robust and efficient framework for data processing, leveraging its capabilities for handling large datasets more effectively than MapReduce. Moreover, finding ways to streamline the setup process would significantly benefit the project's efficiency and reduce the potential for errors. This could involve developing a more integrated setup guide or automating parts of the installation and configuration process. Gaining further experience and knowledge in these areas will be crucial for enhancing the project's outcomes and applicability in real-world scenarios.

In summary, the project achieved its analytical goals but also highlighted critical areas for improvement in terms of setup efficiency and the potential for incorporating more advanced technologies. The insights gained from both the successes and challenges provide a valuable foundation for advancing the work further, offering clear pathways for improvement and exploration.

4. References

- Chan, M. (-) Global status report on road safety, Global Status Report on Road Safety - Time for Action. Available at: https://www.afro.who.int/sites/default/files/2017-06/vid_global_status_report_en.pdf (Accessed: 29 February 2024).
- Rai, V. (2023) Revolutionising Road safety with unprecedented power of data, Mobility Outlook. Available at: https://www.mobilityoutlook.com/commentary/revolutionising-road-safety-with-unprecedented-power-of-data/#:~:text=Role%20Of%20Technology%20In%20Data,predict%20and%20prevent%20future%20incidents. (Accessed: 29 February 2024).
- What is mapreduce? (2024) Databricks. Available at: https://www.databricks.com/glossary/mapreduce#:~:text=MapReduce%20is%20a%20Java%2Dbased,split%20between%20parallel%20processing%20tasks. (Accessed: 29 February 2024).
- Giri, S. (no date) *Overview of apache hadoop ecosystem*, *CloudxLab*. Available at: https://cloudxlab.com/assessment/displayslide/125/overview-of-apache-hadoop-ecosystem?cv=1 (Accessed: 12 March 2024).

5. Appendix

dftRoadSafetyData_Casualties_2018

Accident_Index	Vehicle_Reference	Casualty_Reference	Casualty_Class	Sex_of_Casualty	Age_of_Casualty	Age_Band_of_Casualty	Casualty_Severity	Pedestrian_Location	Pedestrian_Movement	Car_Passenger	Bus_or_Coach_Passenger	Pedestrian_Road_Maintenance_Worker	Casualty_Type	Casualty_Home_Area_Type	Casualty_IMD_Decile
2018010080971	1	1	2	2	50	8	3	0	0	2	0	0	9	1	8
2018010080971	2	2	1	1	48	8	3	0	0	0	0	0	8	1	1
2018010080973	1	1	3	1	29	6	3	5	1	0	0	2	0	1	3
2018010080974	1	1	1	1	40	7	3	0	0	0	0	0	8	1	3
2018010080981	1	1	1	1	27	6	2	0	0	0	0	0	9	1	7
2018010080982	1	1	1	1	43	7	3	0	0	0	0	0	8	1	4
2018010080982	2	2	2	2	58	9	2	0	0	1	0	0	9	1	3
2018010080983	1	1	2	1	69	10	3	0	0	1	0	0	9	1	10
2018010080983	1	2	2	1	56	9	3	0	0	2	0	0	9	1	10
2018010080983	1	3	2	2	60	9	3	0	0	2	0	0	9	1	10
2018010080983	2	4	1	2	41	7	3	0	0	0	0	0	9	1	9
2018010080986	2	1	1	1	27	6	2	0	0	0	0	0	1	1	3
2018010080987	1	1	1	2	36	7	3	0	0	0	0	0	8	1	3

Figure 1 - Header Rows from Dataset

Column Name	Description
Accident_Index	A unique identifier for each accident.
Vehicle_Reference	A numeric reference for the vehicle involved in the accident.
Casualty_Reference	A numeric reference for the casualty involved in the accident.
Casualty_Class	Indicates the class of the casualty (e.g., driver, passenger).
Sex_of_Casualty	The gender of the casualty.
Age_of_Casualty	The age of the casualty.
Age_Band_of_Casualty	A categorisation of the casualty's age into bands.
Casualty_Severity	Indicates the severity of the casualty's injuries.
Pedestrian_Location	Details about the pedestrian's location at the time of the accident.
Pedestrian_Movement	Describes how the pedestrian was moving at the time of the accident.
Car_Passenger	Indicates if the casualty was a car passenger.
Bus_or_Coach_Passenger	Indicates if the casualty was a bus or coach passenger.
Pedestrian_Road_Maintenance_Worker	Specifies if the casualty was a road maintenance worker.
Casualty_Type	The type of casualty (e.g., pedestrian, cyclist).
Casualty_Home_Area_Type	The type of area the casualty comes from (e.g., urban, rural).
Casualty_IMD_Decile	A decile ranking of the casualty's area based on the Index of Multiple Deprivation.

Figure 2 - Dataset Attribute Description

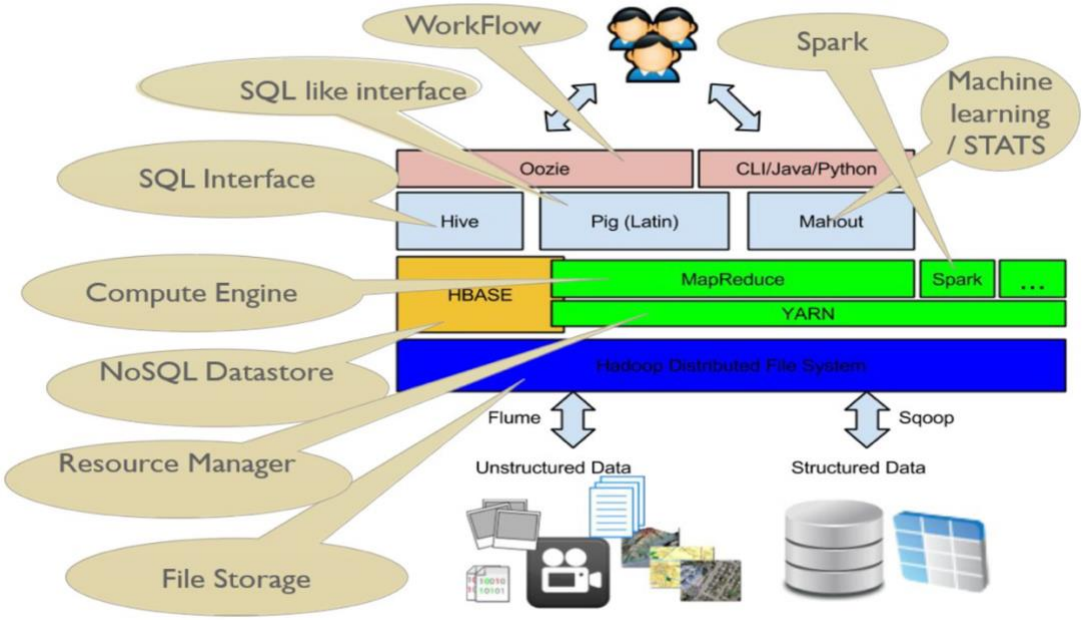

Figure 3 - Hadoop Ecosystem

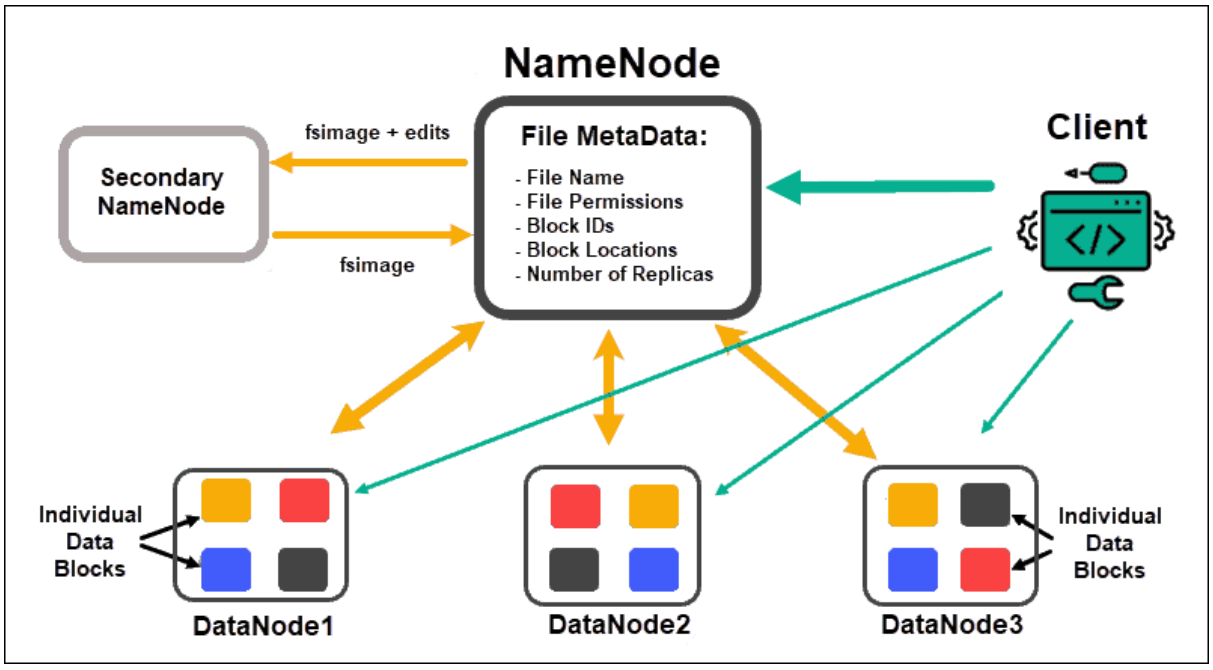

Figure 4 - Hadoop Architecture

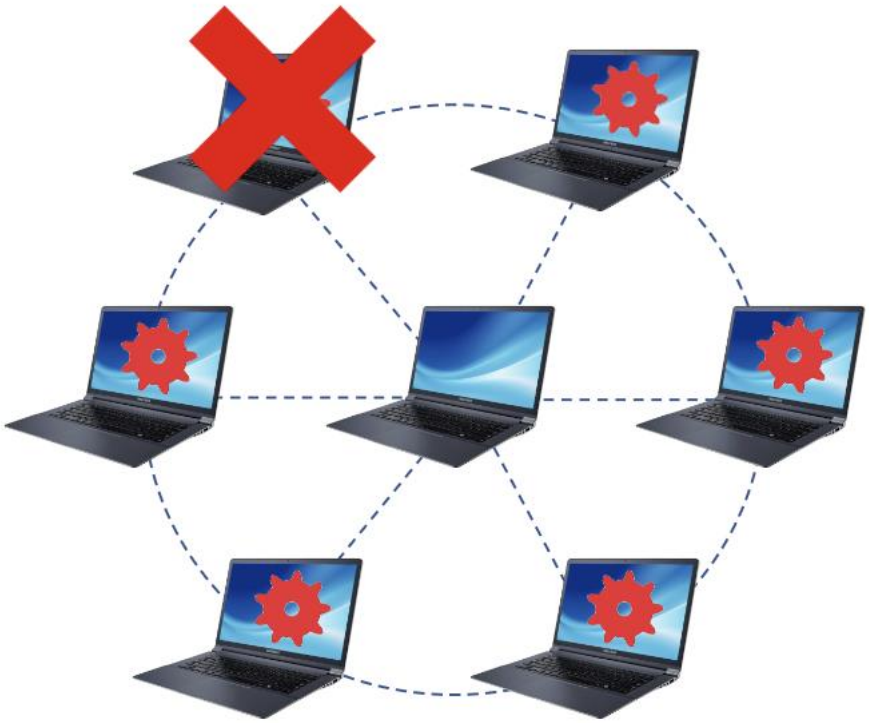

Figure 5 - Node Fault Tolerance

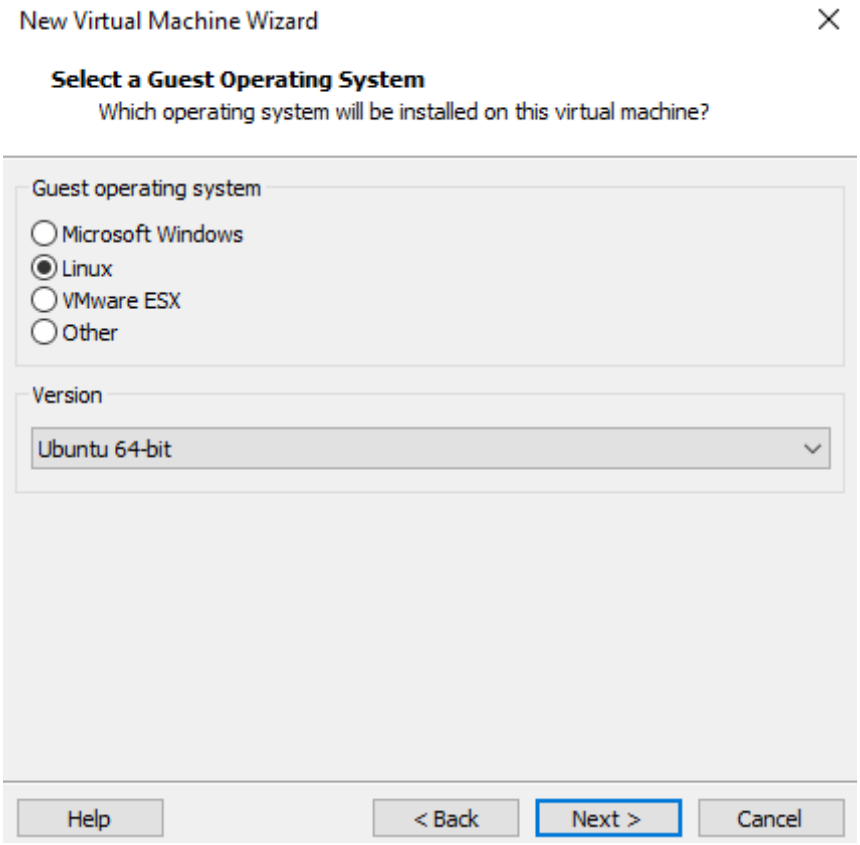

Figure 6 - Operating System Set Up

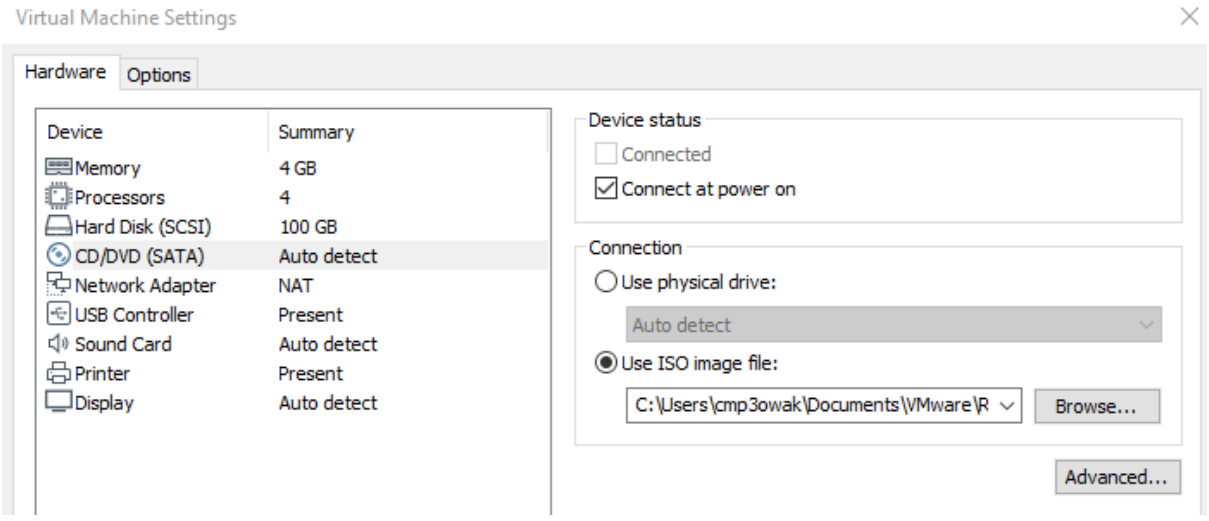

Figure 7 - Allocated Resources

```
  GNU nano 4.8                                          /home/n0982031/.bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export PATH=$PATH:$JAVA_HOME/bin
```

Figure 8 - PATH variable for Java

```
n0982031@LR1:~$ java -version

Command 'java' not found, but can be installed with:

sudo apt install openjdk-11-jre-headless  # version 11.0.20.1+1-0ubuntu1~20.04, or
sudo apt install default-jre              # version 2:1.11-72
sudo apt install openjdk-13-jre-headless  # version 13.0.7+5-0ubuntu1~20.04
sudo apt install openjdk-16-jre-headless  # version 16.0.1+9-1~20.04
sudo apt install openjdk-17-jre-headless  # version 17.0.8.1+1~us1-0ubuntu1~20.04
sudo apt install openjdk-8-jre-headless   # version 8u382-ga-1~20.04.1
```

Figure 9 - Java command before installation

```
n0982031@LR1:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
n0982031@LR1:~$
```

Figure 10 - Java successfully installed (L1)

```
n0982031@LR2:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
n0982031@LR2:~$
```

Figure 11 - Java successfully installed (L2)

```
n0982031@LR3:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
n0982031@LR3:~$
```

Figure 12 - Java successfully installed (L3)

```
n0982031@LR1:/usr/local$ ls -all
total 40
drwxr-xr-x 10 root root 4096 Apr 23  2020 .
drwxr-xr-x 14 root root 4096 Apr 23  2020 ..
drwxr-xr-x  2 root root 4096 Apr 23  2020 bin
drwxr-xr-x  2 root root 4096 Apr 23  2020 etc
drwxr-xr-x  2 root root 4096 Apr 23  2020 games
drwxr-xr-x  2 root root 4096 Apr 23  2020 include
drwxr-xr-x  3 root root 4096 Apr 23  2020 lib
lrwxrwxrwx  1 root root    9 Feb 28 13:31 man -> share/man
drwxr-xr-x  2 root root 4096 Apr 23  2020 sbin
drwxr-xr-x  7 root root 4096 Apr 23  2020 share
drwxr-xr-x  2 root root 4096 Apr 23  2020 src
n0982031@LR1:/usr/local$
```

Figure 13 - Local before installation of Hadoop (L1)

```
n0982031@LR2:/usr/local$ ls -all
total 40
drwxr-xr-x 10 root root 4096 Apr 23  2020 .
drwxr-xr-x 14 root root 4096 Apr 23  2020 ..
drwxr-xr-x  2 root root 4096 Apr 23  2020 bin
drwxr-xr-x  2 root root 4096 Apr 23  2020 etc
drwxr-xr-x  2 root root 4096 Apr 23  2020 games
drwxr-xr-x  2 root root 4096 Apr 23  2020 include
drwxr-xr-x  3 root root 4096 Apr 23  2020 lib
lrwxrwxrwx  1 root root    9 Feb 28 13:34 man -> share/man
drwxr-xr-x  2 root root 4096 Apr 23  2020 sbin
drwxr-xr-x  7 root root 4096 Apr 23  2020 share
drwxr-xr-x  2 root root 4096 Apr 23  2020 src
n0982031@LR2:/usr/local$
```

Figure 14 - Local before installation of Hadoop (L2)

```
n0982031@LR3:/usr/local$ ls -all
total 40
drwxr-xr-x 10 root root 4096 Apr 23  2020 .
drwxr-xr-x 14 root root 4096 Apr 23  2020 ..
drwxr-xr-x  2 root root 4096 Apr 23  2020 bin
drwxr-xr-x  2 root root 4096 Apr 23  2020 etc
drwxr-xr-x  2 root root 4096 Apr 23  2020 games
drwxr-xr-x  2 root root 4096 Apr 23  2020 include
drwxr-xr-x  3 root root 4096 Apr 23  2020 lib
lrwxrwxrwx  1 root root    9 Feb 28 13:36 man -> share/man
drwxr-xr-x  2 root root 4096 Apr 23  2020 sbin
drwxr-xr-x  7 root root 4096 Apr 23  2020 share
drwxr-xr-x  2 root root 4096 Apr 23  2020 src
n0982031@LR3:/usr/local$
```

Figure 15 - Local before installation of Hadoop (L2)

```
n0982031@LR1:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 13:59 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:31 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR1:/usr/local$
```

Figure 16 - Local after installation of Hadoop (L1)

```
n0982031@LR2:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 13:59 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:34 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR2:/usr/local$
```

Figure 17 - Local after installation of Hadoop (L2)

```
n0982031@LR3:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 13:59 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:36 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR3:/usr/local$
```

Figure 18 - Local after installation of Hadoop (L3)

```
 GNU nano 4.8                                  /home/n0982031/.bashrc
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_CONF_DIR=$HADOOP_INSTALL/etc/hadoop
export HADOOP_HOME=/usr/local/hadoop
```

Figure 19 - Hadoop Path Variables

```
n0982031@LR2: /usr/local/hadoop/etc/hadoop
 GNU nano 4.8                    hadoop-env.sh                    Modified
# and therefore may override any similar flags set in HADOOP_OPTS
#
# export HADOOP_MOVER_OPTS=""

###
# Advanced Users Only!
###

# The directory where pid files are stored. /tmp by default.
# NOTE: this should be set to a directory that can only be written to by
#       the user that will run the hadoop daemons.  Otherwise there is the
#       potential for a symlink attack.
export HADOOP_PID_DIR=${HADOOP_PID_DIR}
export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}

# A string representing this instance of hadoop. $USER by default.
export HADOOP_IDENT_STRING=$USER
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export PATH=$PATH:$JAVA_HOME/bin
export HADOOP_CONF_DIR=$HADOOP_INSTALL/etc/hadoop
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text^T To Spell  ^  Go To Line
```

Figure 20 - Hadoop-env.sh configuration

```
n0982031@LR1:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 14:00 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
lrwxrwxrwx  1 root     root       12 Feb 28 14:00 hadoop -> hadoop-2.9.0
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:31 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR1:/usr/local$
```

Figure 21 - Symbolic link of Hadoop (L1)

```
n0982031@LR2:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 14:00 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
lrwxrwxrwx  1 root     root       12 Feb 28 14:00 hadoop -> hadoop-2.9.0
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:34 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR2:/usr/local$
```

Figure 22 - Symbolic link of Hadoop (L2)

```
n0982031@LR3:/usr/local$ ls -all
total 44
drwxr-xr-x 11 root     root     4096 Feb 28 14:00 .
drwxr-xr-x 14 root     root     4096 Apr 23  2020 ..
drwxr-xr-x  2 root     root     4096 Apr 23  2020 bin
drwxr-xr-x  2 root     root     4096 Apr 23  2020 etc
drwxr-xr-x  2 root     root     4096 Apr 23  2020 games
lrwxrwxrwx  1 root     root       12 Feb 28 14:00 hadoop -> hadoop-2.9.0
drwxr-xr-x  9 n0982031 n0982031 4096 Nov 13  2017 hadoop-2.9.0
drwxr-xr-x  2 root     root     4096 Apr 23  2020 include
drwxr-xr-x  3 root     root     4096 Apr 23  2020 lib
lrwxrwxrwx  1 root     root        9 Feb 28 13:36 man -> share/man
drwxr-xr-x  2 root     root     4096 Apr 23  2020 sbin
drwxr-xr-x  7 root     root     4096 Apr 23  2020 share
drwxr-xr-x  2 root     root     4096 Apr 23  2020 src
n0982031@LR3:/usr/local$
```

Figure 23 - Symbolic link of Hadoop (L3)

```
n0982031@LR1:~$ hadoop version
Hadoop 2.9.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 756ebc8394e473ac25feac05fa493f6d612e6c50
Compiled by arsuresh on 2017-11-13T23:15Z
Compiled with protoc 2.5.0
From source with checksum 0a76a9a32a5257331741f8d5932f183
This command was run using /usr/local/hadoop-2.9.0/share/hadoop/common/hadoop-common-2.9.0.jar
n0982031@LR1:~$
```

Figure 24 - Hadoop Installation Verification

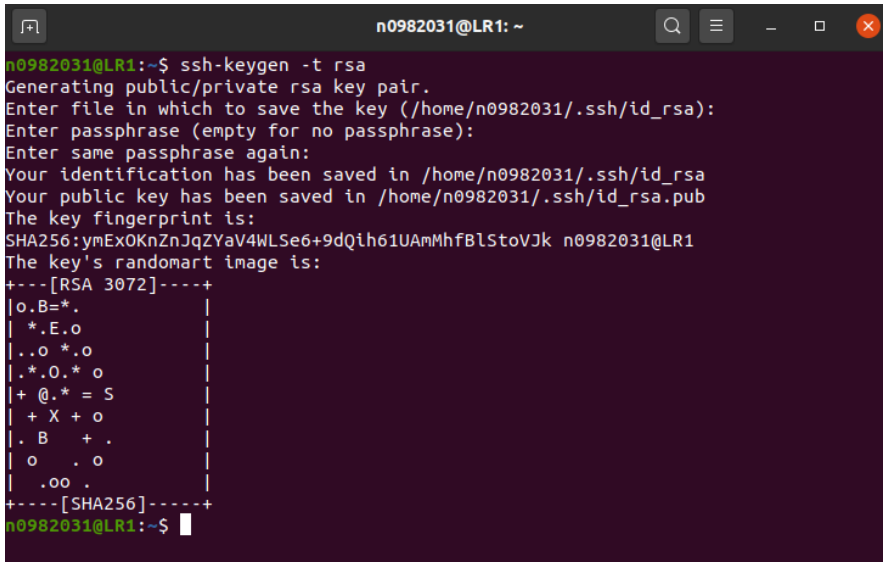

Figure 25 - SSH access for L1

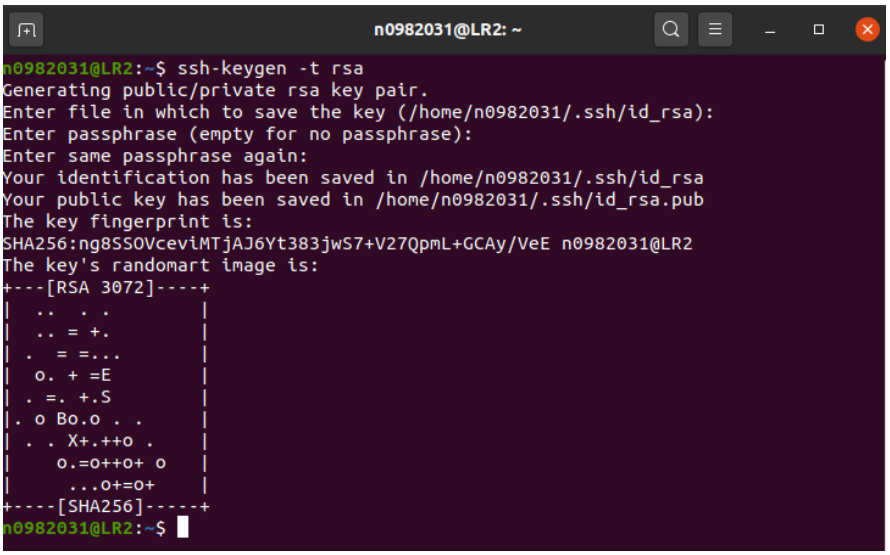

Figure 26 - SSH access for L2

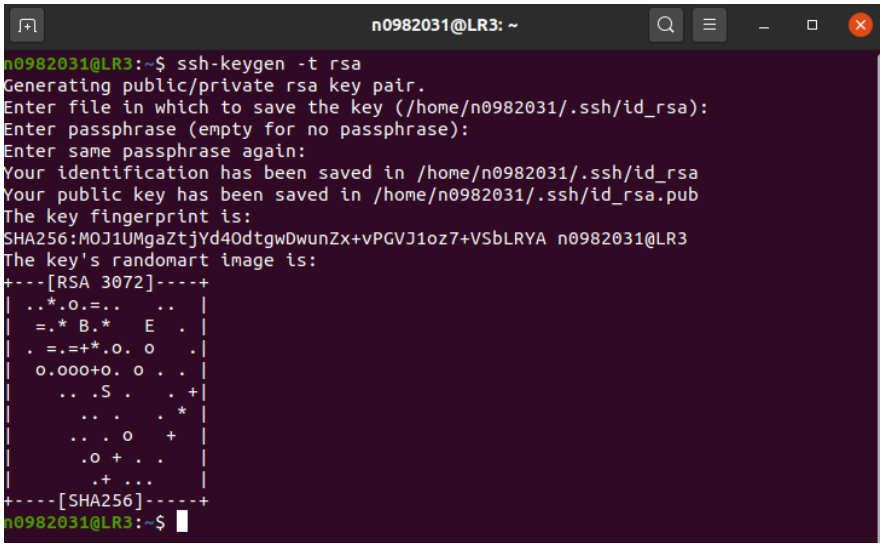

Figure 27 - SSH access for L3

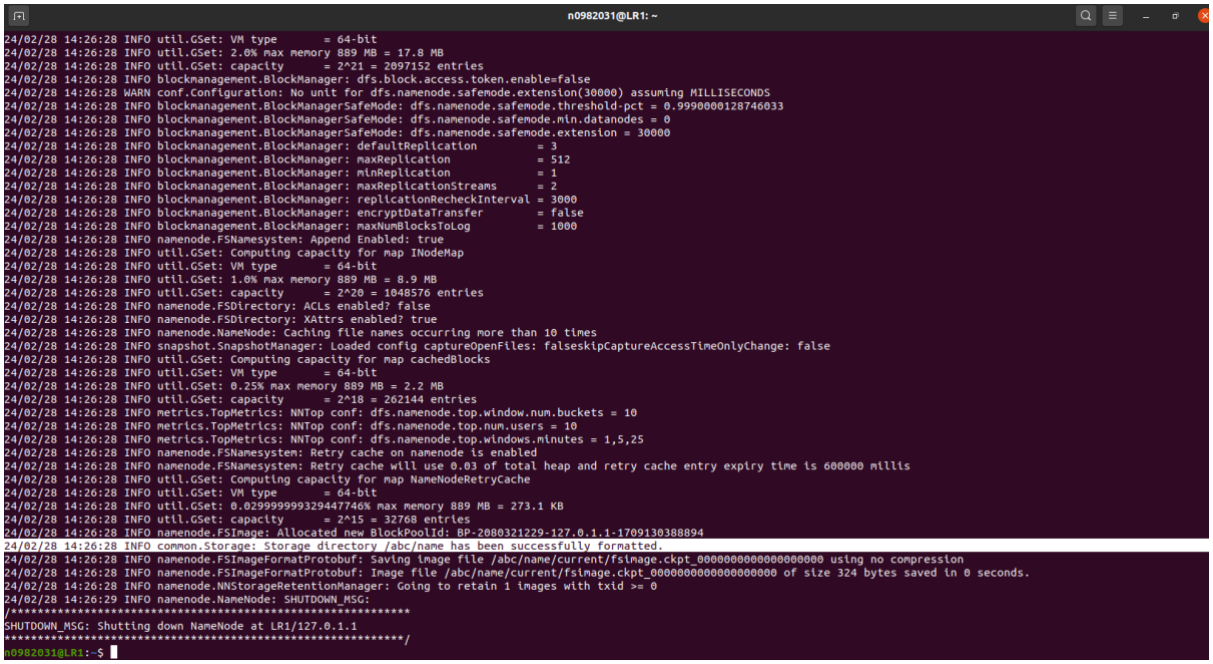

Figure 28 - Formatting of the NameNode

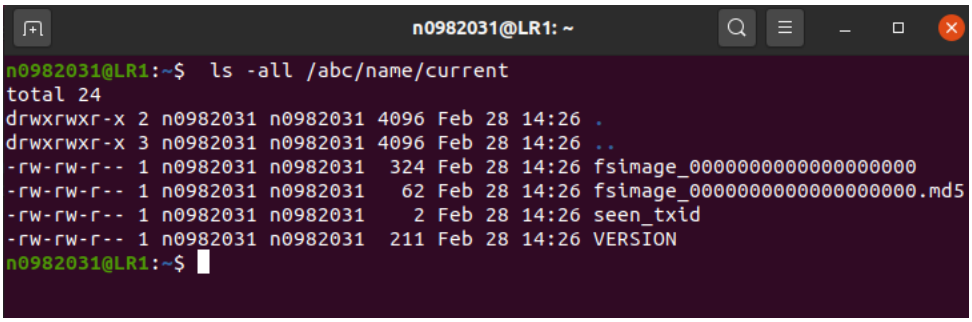

Figure 29 - Storage Formatted

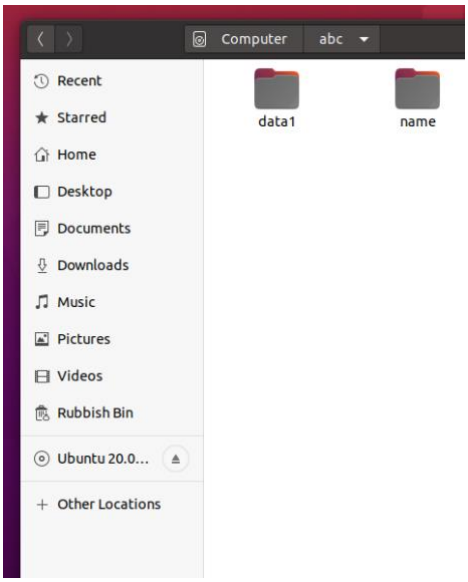

Figure 30 - NameNode Formatted

```
n0982031@LR1:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [LR1]
LR1: starting namenode, logging to /usr/local/hadoop-2.9.0/logs/hadoop-n0982031-namenode-LR1.out
LR1: starting datanode, logging to /usr/local/hadoop-2.9.0/logs/hadoop-n0982031-datanode-LR1.out
LR3: starting datanode, logging to /usr/local/hadoop-2.9.0/logs/hadoop-n0982031-datanode-LR3.out
LR2: starting datanode, logging to /usr/local/hadoop-2.9.0/logs/hadoop-n0982031-datanode-LR2.out
Starting secondary namenodes [LR2]
LR2: starting secondarynamenode, logging to /usr/local/hadoop-2.9.0/logs/hadoop-n0982031-secondarynamenode-LR2.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop-2.9.0/logs/yarn-n0982031-resourcemanager-LR1.out
LR3: starting nodemanager, logging to /usr/local/hadoop-2.9.0/logs/yarn-n0982031-nodemanager-LR3.out
LR2: starting nodemanager, logging to /usr/local/hadoop-2.9.0/logs/yarn-n0982031-nodemanager-LR2.out
LR1: starting nodemanager, logging to /usr/local/hadoop-2.9.0/logs/yarn-n0982031-nodemanager-LR1.out
n0982031@LR1:~$
```

Figure 31 - Successful start of cluster

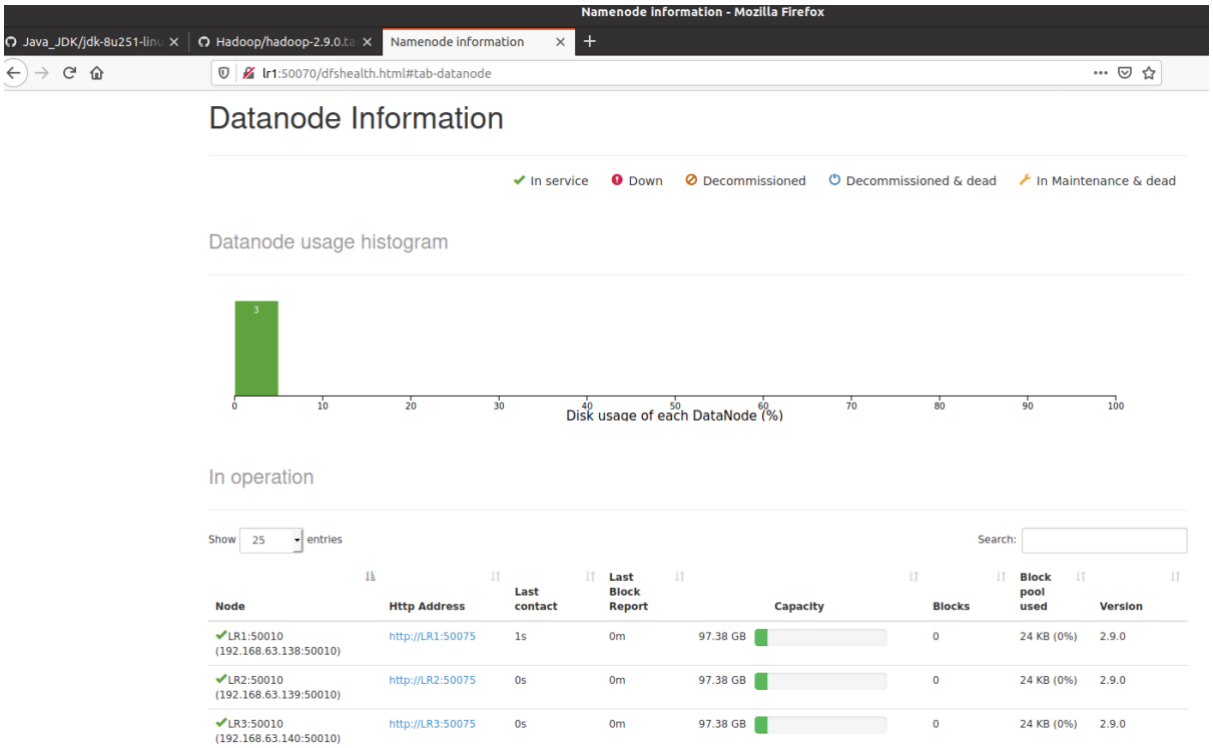

Figure 32 - Hadoop Cluster Web Interface

```
n0982031@LR1:~$ jps
15314 DataNode
15842 NodeManager
16134 Jps
15142 NameNode
15530 ResourceManager
```

Figure 33 - JPS for LR1

```
n0982031@LR2:~$ jps
13266 Jps
13140 NodeManager
13029 SecondaryNameNode
12888 DataNode
```

Figure 34 - JPS for LR2

```
n0982031@LR3:~$ jps
12833 NodeManager
12955 Jps
12687 DataNode
```

Figure 35 - JPS for LR3

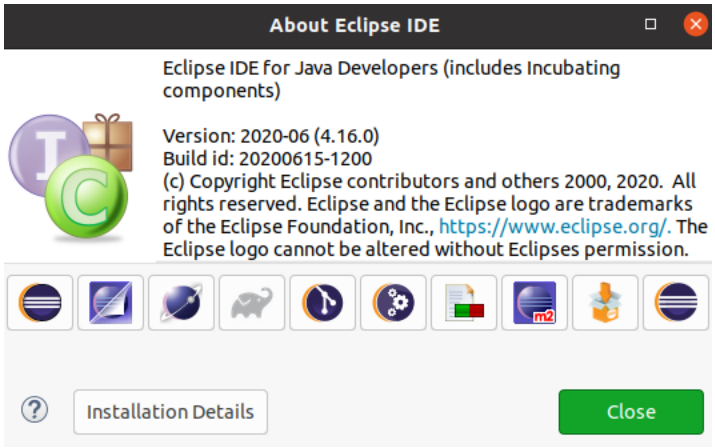

Figure 36 - Eclipse Version Installation

```
n0982031@LR1:~/MapReduceRoad$ hadoop fs -ls /user/n0982031/JavaExampleRoad2
Found 2 items
drwxr-xr-x   - n0982031 supergroup          0 2024-02-28 17:34 /user/n0982031/JavaExampleRoad2/Road_out
-rw-r--r--   3 n0982031 supergroup    7466192 2024-02-28 17:33 /user/n0982031/JavaExampleRoad2/dftRoadSafetyData_Casualties_2018.csv
n0982031@LR1:~/MapReduceRoad$ hadoop fs -ls /user/n0982031/JavaExampleRoad2/Road_out
Found 2 items
-rw-r--r--   3 n0982031 supergroup          0 2024-02-28 17:34 /user/n0982031/JavaExampleRoad2/Road_out/_SUCCESS
-rw-r--r--   3 n0982031 supergroup         44 2024-02-28 17:34 /user/n0982031/JavaExampleRoad2/Road_out/part-00000
n0982031@LR1:~/MapReduceRoad$
```

Figure 37 - Data Folder

```
n0982031@LR1: /usr/local/spark/conf
 GNU nano 4.8                                    spark-env.sh
#!/usr/bin/env bash

export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export SPARK_HOME=/usr/local/spark
export SCALA_HOME=/usr/local/spark/examples/src/main

export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
export SPARK_WORKER_DIR=/usr/local/spark/conf
export SPARK_LOG_DIR=/usr/local/spark/conf
export SPARK_MASTER_IP=LR1
```

Figure 38 - Spark-env.sh File

Variable	Value	Description
JAVA_HOME	/usr/lib/jvm/jdk1.8.0_251	The path to the Java Development Kit installation. Required by Spark to run Java applications.
SPARK_HOME	/usr/local/spark	The directory where Apache Spark is

		installed. Used by Spark scripts to locate Spark libraries and other files.
SCALA_HOME	/usr/local/spark/examples/src/main	This should point to the installation directory of Scala. However, in this file, it's incorrectly pointing to Spark's examples source path.
HADOOP_CONF_DIR	$HADOOP_HOME/etc/hadoop	The directory where Hadoop's configuration files are located, allowing Spark to integrate with Hadoop's file system (HDFS) and YARN.
YARN_CONF_DIR	$HADOOP_HOME/etc/hadoop	Points to the directory containing YARN's configuration files. This allows Spark to run on the YARN resource manager.
SPARK_WORKER_DIR	/usr/local/spark/conf	Specifies the directory for Spark's worker node configurations, used in standalone cluster mode.
SPARK_LOG_DIR	/usr/local/spark/conf	Indicates where Spark's log files should be stored. Typically, this should be a directory intended specifically for logs.
SPARK_MASTER_IP	LR1	The hostname or IP address of the Spark master node. Informs worker nodes about the master node's location.

Figure 39 - Spark-env.sh variable descriptions

```
n0982031@LR1:~$ cd MapReduceRoad
n0982031@LR1:~/MapReduceRoad$ hadoop fs -cat /user/n0982031/JavaExampleRoad2/Roa
d_out/part-00000
1       1784
2       25511
3       133302
Casualty_Severity       1
n0982031@LR1:~/MapReduceRoad$
```

Figure 40- Results shown in terminal

Figure 41 - Results shown in web browser

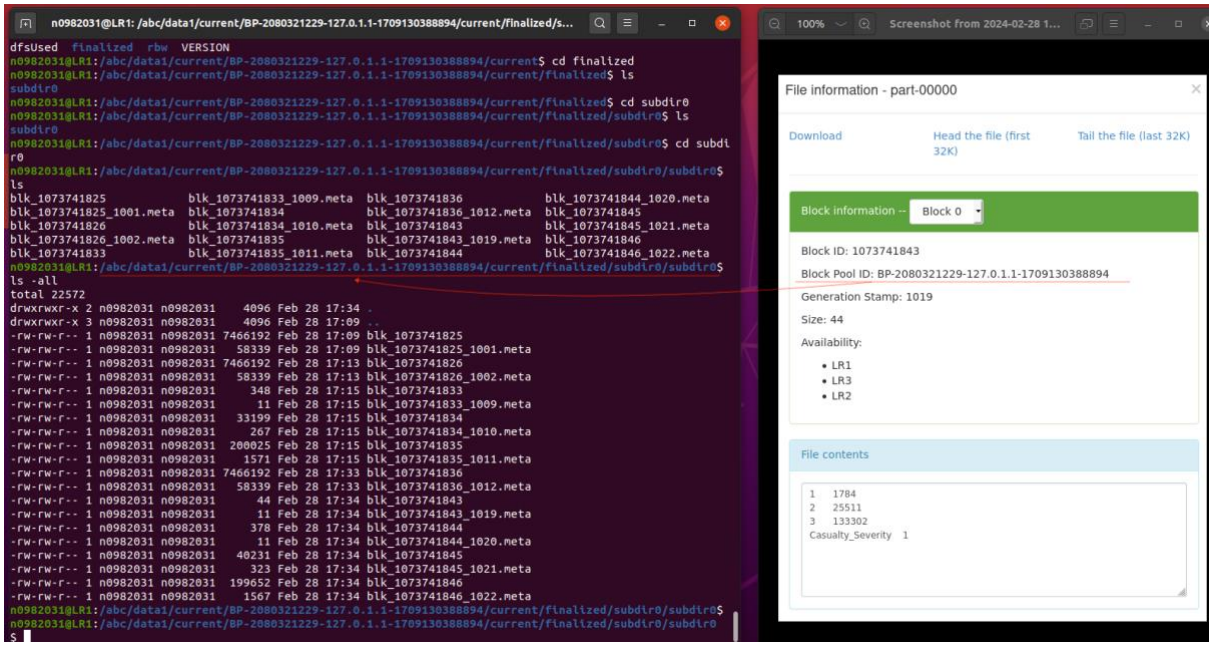

Figure 42 - Block Pool ID