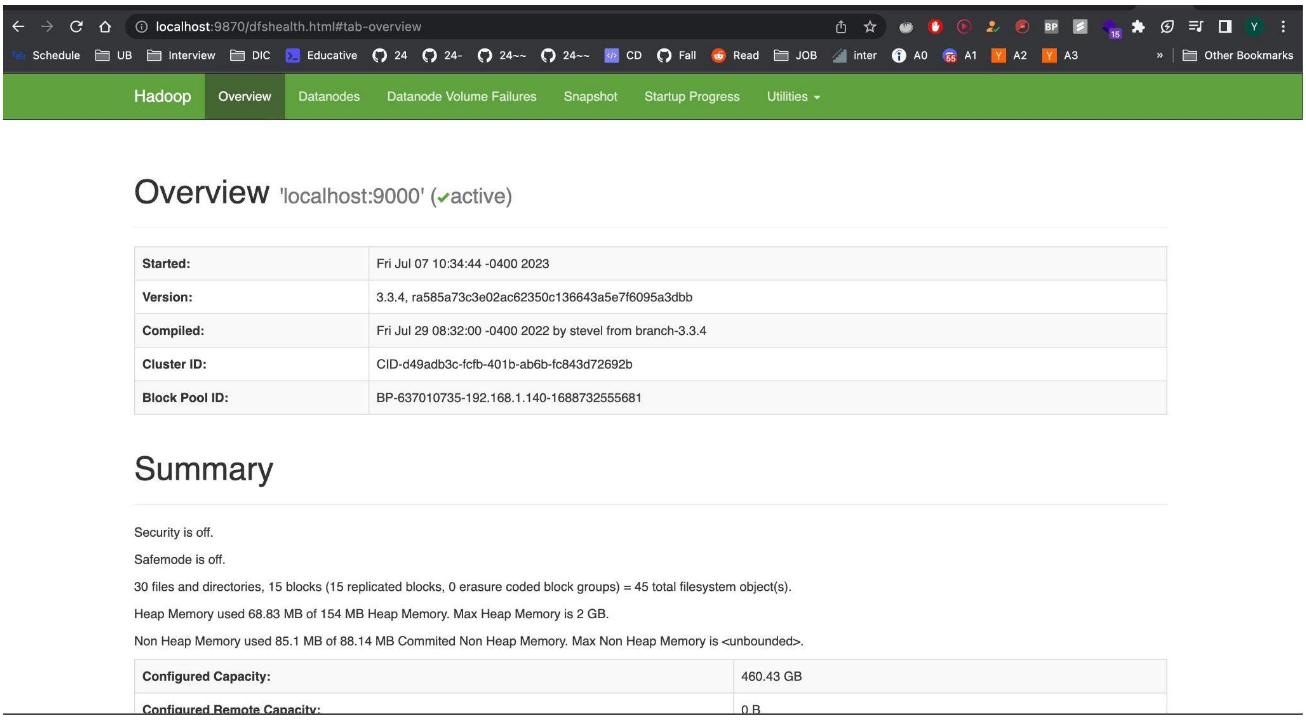
Team members : Dhiraj Gunasheela Nikitha Sadhanandha, Rakshith Venkatesh Murthy Gowda, Yashwanth Rama Krishna Reddy, Dhiraj

1.

Created local instance of Hadoop



To check Hadoop health in command prompt

```
~ jps
15879 JobHistoryServer
19817 NameNode
19930 DataNode
20075 SecondaryNameNode
22187 Jps
18811 XMLServerLauncher
20380 NodeManager
20270 ResourceManager
~
```

The `jps` command lists out all Java processes running on a machine, useful in Hadoop since its services are Java processes. It aids in debugging by allowing you to verify if all Hadoop services (like NameNode, DataNode, ResourceManager, etc.) are operational.

```python
import pandas as pd
import time

filename = 'test.ft.txt'

try:
    start_time = time.time()
    df = pd.read_csv(filename, sep="\t", header=None)
    end_time = time.time()
    total_time = end_time - start_time
    print(f"Time taken to load the DataFrame: {total_time} seconds")
except FileNotFoundError:
    print(f"File {filename} not found.")
    df = None
```

```
Time taken to load the DataFrame: 2.4636619091033936 seconds
```

```
~ hdfs dfs -mkdir /phase-3

2023-07-07 17:35:11,714 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
~ hdfs dfs -put /Users/yashwanth/DIC/archive/test.ft.txt /phase-3
2023-07-07 17:35:17,924 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
~
```

Created dir phase-3

Loaded amazon review dataset to phase-3 using Hadoop File system put command, which does write operation

We have used textual dataset

```
✘ ~ time hdfs dfs -put /Users/yashwanth/DIC/archive/test.ft.txt /phase-3
2023-07-07 17:46:40,189 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
hdfs dfs -put /Users/yashwanth/DIC/archive/test.ft.txt /phase-3  2.63s user 0.34s system 213% cpu 1.390 total
```

In simple terms, hdfs dfs -put command took 1.39 seconds to complete in real time which is 2x faster than loading in pandas dataframe

2.

hadoop jar /opt/homebrew/Cellar/hadoop/3.3.4/libexec/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar wordcount /phase-3/test.ft.txt /output

Performing mapreduce job on test.ft.txt dataset

```
~ hadoop jar /opt/homebrew/Cellar/hadoop/3.3.4/libexec/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.4.jar wordcount /phase-3/test.ft.txt /output

2023-07-07 19:14:03,211 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2023-07-07 19:14:03,631 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2023-07-07 19:14:03,894 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/yashwanth/.staging/job_1688740494980_0008
2023-07-07 19:14:04,044 INFO input.FileInputFormat: Total input files to process : 1
2023-07-07 19:14:04,105 INFO mapreduce.JobSubmitter: number of splits:2
2023-07-07 19:14:04,239 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1688740494980_0008
2023-07-07 19:14:04,239 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-07-07 19:14:04,341 INFO conf.Configuration: resource-types.xml not found
2023-07-07 19:14:04,342 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-07-07 19:14:04,391 INFO impl.YarnClientImpl: Submitted application application_1688740494980_0008
2023-07-07 19:14:04,412 INFO mapreduce.Job: The url to track the job: http://Yashwanth.lan:8088/proxy/application_1688740494980_0008/
2023-07-07 19:14:04,412 INFO mapreduce.Job: Running job: job_1688740494980_0008
2023-07-07 19:14:09,527 INFO mapreduce.Job: Job job_1688740494980_0008 running in uber mode : false
2023-07-07 19:14:09,530 INFO mapreduce.Job:  map 0% reduce 0%
2023-07-07 19:14:19,651 INFO mapreduce.Job:  map 50% reduce 0%
2023-07-07 19:14:25,693 INFO mapreduce.Job:  map 75% reduce 0%
2023-07-07 19:14:30,719 INFO mapreduce.Job:  map 100% reduce 0%
2023-07-07 19:14:32,733 INFO mapreduce.Job:  map 100% reduce 100%
2023-07-07 19:14:32,741 INFO mapreduce.Job: Job job_1688740494980_0008 completed successfully
2023-07-07 19:14:32,807 INFO mapreduce.Job: Counters: 51
	File System Counters
		FILE: Number of bytes read=53488934
		FILE: Number of bytes written=74612430
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
		HDFS: Number of bytes read=177380501
		HDFS: Number of bytes written=13128135
		HDFS: Number of read operations=11
		HDFS: Number of large read operations=0
		HDFS: Number of write operations=2
		HDFS: Number of bytes read erasure-coded=0
	Job Counters
		Killed map tasks=1
		Launched map tasks=3
		Launched reduce tasks=1
		Data-local map tasks=3
		Total time spent by all maps in occupied slots (ms)=31075
		Total time spent by all reduces in occupied slots (ms)=9784
		Total time spent by all map tasks (ms)=31075
		Total time spent by all reduce tasks (ms)=9784
		Total vcore-milliseconds taken by all map tasks=31075
		Total vcore-milliseconds taken by all reduce tasks=9784
		Total megabyte-milliseconds taken by all map tasks=31820800
		Total megabyte-milliseconds taken by all reduce tasks=10018816
```

Logged in as: dr.whc

MapReduce Job job_1688740494980_0008

Job Overview

Job Name:	word count
User Name:	yashwanth
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Jul 07 19:14:04 EDT 2023
Started:	Fri Jul 07 19:14:07 EDT 2023
Finished:	Fri Jul 07 19:14:30 EDT 2023
Elapsed:	22sec
Diagnostics:	
Average Map Time	13sec
Average Shuffle Time	8sec
Average Merge Time	0sec
Average Reduce Time	0sec

Total time elapsed is 22 seconds which includes various factors mapping time, shuffling, inter process communication, reduce tim.

Average Map Time (13 sec): This is the average time taken by the 'Map' function to process the input data chunks and output key-value pairs. This includes breaking down the text into individual words and emitting a key-value pair for each word, with the word as the key and '1' as the value.

Average Shuffle Time (8 sec): This is the average time taken to perform the 'Shuffle & Sort' phase. This phase collects the outputs from all the 'Map' tasks, sorts these key-value pairs based on the keys (the words), and groups the values (counts) for each unique key together.

ApplicationMaster

Attempt Number	Start Time	Node	Logs
	Fri Jul 07 19:14:05 EDT 2023	yashwanth.lan:8042	logs

Task Type	Total	Complete	
Map	2	2	
Reduce	1	1	
Attempt Type	Failed	Killed	Successful
Maps	0	1	2
Reduces	0	0	1

Performing word count operation without mapreduce.

Couldn't run the code, my system couldn't handle the load and I had to restart the kernel everytime

[8]:

```
from collections import Counter
import re

def count_words(filename):
    try:
        with open(filename, 'r') as file:
            text = file.read()
            words = re.findall(r'\b\w+\b', text)
            counter = Counter(words)
            return counter
    except FileNotFoundError:
        print(f"Sorry, the file {filename} does not exist.")
        return None

counter = count_words('test.ft.txt')
```

3.

MapReduce is essentially a programming paradigm that uses the power of parallel processing to handle and analyze vast datasets efficiently. It operates on the principle of breaking down a large task into smaller subtasks, which can be processed independently and parallelly

The MapReduce consists of three main stages: Map, Shuffle & Sort, and Reduce.

- **Map**: In this stage, the input data is divided into chunks and processed in parallel by multiple worker nodes. Each worker applies a given function, called the "map function," to the input data it receives. The map function transforms the input data into a set of intermediate key-value pairs.

- **Shuffle & Sort**: Once the map stage is complete, the intermediate key-value pairs generated by the workers are collected and sorted based on the keys. This sorting step is crucial to ensure that all the values associated with a particular key are grouped together.

- **Reduce**: In the reduce stage, another set of worker nodes takes the sorted intermediate data and applies a given function, called the "reduce function," to produce the final output. The reduce function aggregates the values associated with each key and generates a set of final output key-value pairs.

Throughout the MapReduce process, there are two key actors involved:

1. Master: The master node coordinates the overall execution of the MapReduce job. It assigns map and reduce tasks to the available worker nodes, monitors their progress, and manages the data exchange between the workers.

2. Workers: The worker nodes are responsible for performing the actual computation. They execute the assigned map and reduce tasks on their allocated portions of the data. Each worker processes its data independently of other workers.

In the context of word count

Map phase

- **Input**: The input data for the word count task is a collection of text documents.
- **Map function**: Each mapper processes a portion of the input data. The map function takes a document as input and emits key-value pairs. The map function tokenizes the document into

individual words and emits a key-value pair for each word, where the word is the key and the value is set to 1.

Shuffle and Sort Phase

- **Shuffle**: The MapReduce framework collects all the intermediate key-value pairs generated by the map function and redistributes them based on the keys.
- **Sort**: Within each group, the intermediate key-value pairs are sorted based on the keys in ascending order.

Reduce phase

- **Reduce function**: Each reducer takes a group of key-value pairs with the same key as input. The reduce function receives a key (a word) and a list of values (a list of 1s).

The final output of the MapReduce job is a collection of key-value pairs where the key represents a unique word, and the value represents the total count of occurrences of that word across all the input documents.

Extra Credit

In [7]:
```python
import pandas as pd
from gensim.models import Word2Vec, FastText
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

# Read the CSV file into a DataFrame
df = pd.read_csv('twitter_training.csv')

# Keep the first 2000 rows and drop any rows with missing text values
df = df[:2000].dropna(subset=['text'])

# Tokenize the text
df['tokenized_text'] = df['text'].apply(lambda x: x.split())

# Train Word2Vec model
word2vec = Word2Vec(df['tokenized_text'], min_count=2)

# Train FastText model
fasttext = FastText(df['tokenized_text'], min_count=2)

# Convert each tweet's tokenized text to a vector representation
def text_to_vector(text):
    vectors = [word2vec.wv.get_vector(w) for w in text if w in word2vec.wv.key_to_index]
    return np.mean(vectors, axis=0) if vectors else np.zeros(word2vec.vector_size)
```

```python
df['text_vec'] = df['tokenized_text'].apply(text_to_vector)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df['text_vec'].tolist(), df['sentiment'], test_size=0.2, random_state=42)

# Train a logistic regression classifier
clf = LogisticRegression().fit(X_train, y_train)

# Evaluate the classifier's accuracy on the test set
score = clf.score(X_test, y_test)
print("Accuracy:", score)
```

```
0.45112781954887216
```