Final Year Project Report

Full Unit – Final Report

Massive Scale Data Analytics

Cosmin Cristian Sirbu

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Riquelme Granada

Department of Computer Science

Royal Holloway, University of London

March 21, 2023

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 23210 (without appendix)

Student Name: Cosmin Cristian Sirbu

Date of Submission: 21/03/2023

Signature: Cosmin Sirbu

CONTENTS

ABSTRACT

My fascination with Data Science was sparked for the first time at the end of my first year. During the summer break, I took on a Deep Learning course in which I developed complex Machine Learning programs like object recognition and image style transfer. Seeing the endless possibilities that Data Science gives you as a programmer I decided to specialize in it and do my dissertation project in this field.

The reason I chose Hadoop and Spark as frameworks for my final year project was that I was able to see them in action when I worked at Nomura as part of my industrial placement. Here teams were using these tools alongside machine learning to analyse millions of trades and provide feedback and statistics back to the brokers and investors, influencing the flow of millions or billions of pounds. Seeing the power these tools have in real-world scenarios made my final-year project choice a natural decision.

The main aim of this project is to build and maintain reliable and scalable distributed systems and apply data analytics techniques for various difficulties. To achieve this goal, I developed MapReduce programs with Hadoop and Spark, I understood and used distributed file systems and constructed a natural processing model. The project is structured in two parts corresponding to the two academic terms.

In the first part, I developed two MapReduce programs with Hadoop: word count and word grep and deployed them on a real cluster, Royal Holloway Big Data Cluster. As a data source, I used a large collection of web-scrapped articles of around 6GB in size. For each problem, I carefully analysed its performance and fine-tuned its parameters to achieve the best execution time.

In the second part, I developed a program to calculate the historical significance of individuals on the entire Wikipedia. Using the entire Wikipedia data source, Spark, and natural language processing, my program attributes a score to every person with a Wikipedia article. At the core of this program lay 2 sub-problems: Page Rank and Natural Language Processing for text semantic categorization. To achieve the final results, I combine the results of each problem to create a sorted list of the most significant personalities in history. Using the Royal Holloway Big Data cluster, for each problem I carefully analysed its performance and fine-tuned its parameters to achieve the best execution time.

In the software development lifecycle of my project, I used my experience with DevOps tools and Agile development gained in my industrial placement to develop and maintain the software to the highest standard. I introduced release & testing automation, quality assurance and efficient planning and management with the help of Jira, GitLab CI/CD, Gitlab, Junit and Docker.

In this report the reader will firstly be introduced to the background theory necessary to understand the key concepts of the programs. Then the implementation of each problem will be presented, accompanied by a critical analysis of the tools, methodologies, experiments, and results obtained. Then I will present my software engineering and workflow done for this project. Finally, I will showcase the professional and technical issues encountered and then the finish with a reflective conclusion and some suggested future improvements for the project.

PROJECT GOALS

- Store & manipulate large datasets with HDFS.
- Run distributed programs with Hadoop and Spark.
- Apply data analytics techniques on large datasets efficiently to answer interesting real-world questions.
- Understand how Spark is different from Hadoop.
- Obtain in-depth applied knowledge of both Spark & Hadoop ecosystems.
- Gain practical experience with the workflow of a professional Data Analytics project.
- Hands-on experience with optimizations and performance analysis for massive-scale data.
- Understand the power and the scale of Big Data Analytics solutions for real-world problems.
- Curate and construct a qualitative data source suited for a Machine Learning model.
- Build a Machine Learning Model from first principles.
- Create a Natural Language Processing pipeline.
- Analyse and fine-tune the performance of a machine learning model.
- Scale a machine learning model in a distributed environment.

IMPROVEMENTS FROM THE INTERIM VERSION

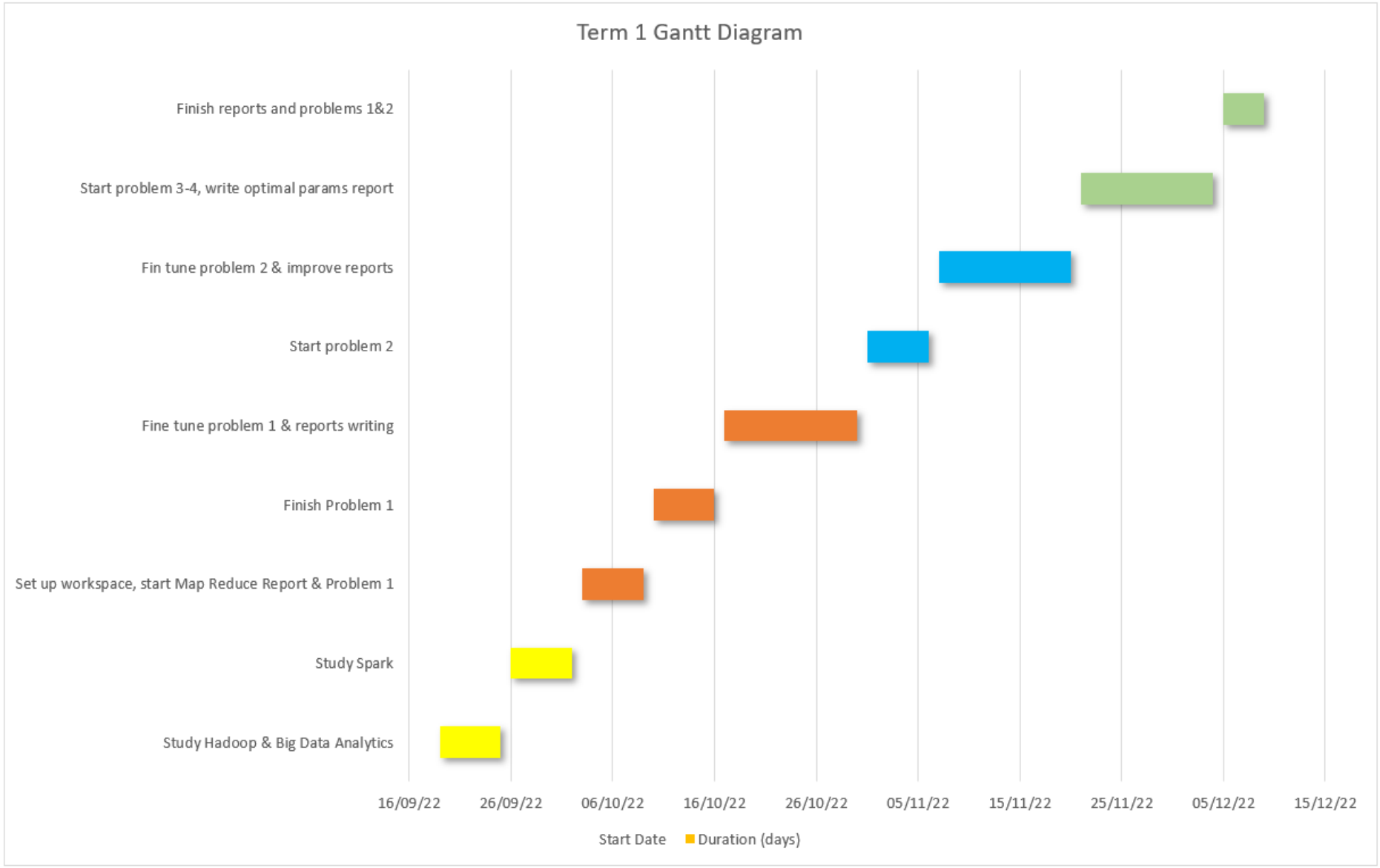
In the interim version, my focus was to learn Hadoop and apply Big Data Analytic tools to extensively analyse and fine-tune the performance of two programs: word count and word grep. For the final version of the project, my goal was to create an entirely different program from first principles, the historical significance program. Therefore, the programs developed over the second term two are page rank, text semantic classifier and historical significance. Each of these problems was developed, tested and fine-tuned on a real cluster, the Royal Holloway Big Data cluster. For them, I performed extensive testing, experimentation, and performance analysis to achieve the best execution time.

PROJECT TIMELINE

At the beginning of term one, I created the entire timeline of my project for both terms. In term one I took this project planning stage very seriously and tried to create a plan as feasible as possible, taking into account other external impacts on the project such as assignments or exams. Throughout the first term, I aimed to respect myself as much as possible my timeline and mostly succeeded. As in real-life projects, impediments or miscalculations may happen so in the cases when I couldn't strictly follow my self-imposed deadlines, I made sure to recuperate. At the of term one, I carefully re-accessed my progress and adapted my plan accordingly. Compared to term one, in the second term, I believe I did an even better job of strictly respecting my plan by having a better experience with working with Big Data tools and a better assessment of my own learning and coding speed. In the following tables and diagrams, the reader can see the main steps involved in the development of this project over the two terms.

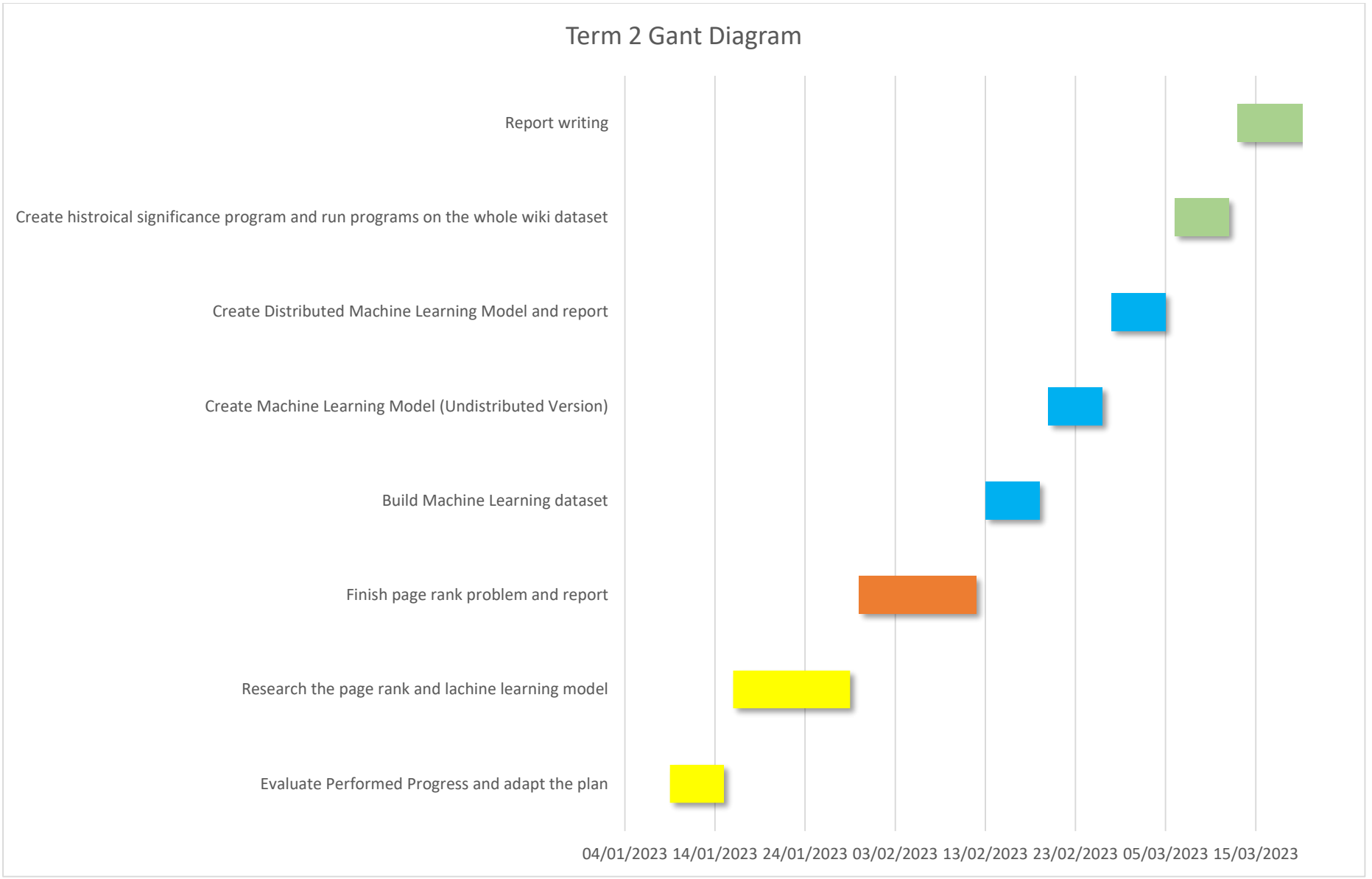
TERM 1 (19 SEPTEMBER TO 9 DECEMBER 2022)

TASK	START DATE	END DATE	DESCRIPTION
STUDY HADOOP & BIG DATA ANALYTICS	19/09/2022	25/09/2022	Study & understand the basics of Data Analytics tools and concepts by studying Hadoop the Definitive guide & TDWI best practices report as well as look at online resources
STUDY SPARK	26/09/2022	02/10/2022	Study the Spark ecosystem and how it is different from the MapReduce using the Big Data Analytics & Spark: Cluster computing with working sets research paper.
SET UP WORKSPACE, START MAP REDUCE REPORT & PROBLEM 1	03/10/2022	09/10/2022	Set up Gitlab, Jira, Jenkins and start writing the MR report, Jira tasks creation and initial work for problem 1 in Hadoop.
FINISH PROBLEM 1	10/10/2022	16/10/2022	Finish basic problem 1 Hadoop solution, start testing on incrementally larger datasets.
FINE TUNE PROBLEM 1 & START WRITING PROBLEMS, HADOOP CLUSTER & PERFORMANCE REPORTS	17/10/2022	30/10/2022	Refactor problem 1 based on SE principles and optimizations for larger datasets. Based on gained experience I will start writing the mentioned reports.
START PROBLEM 2	31/10/2022	06/11/2022	Start working on problem 2 in Hadoop. After finishing basic version, I will start testing the solution on incrementally larger datasets.
FINE TUNE PROBLEM 2 & IMPROVE THE REPORTS	07/11/2022	20/11/2022	Refactor problem 2 based on SE principles and optimizations for larger datasets. Further improve all the reports based on new findings.
START WORKING ON PROBLEM 3, 4 & WRITE THE OPTIMAL PARAMETERS REPORT	21/11/2022	04/12/2022	Based on optimization experience gained for problems 1 & 2, I will start writing the Optimal parameters report and start working on the last 2 problems.
FINALISE REPORTS AND PROBLEMS 1 & 2	05/12/2022	09/12/2022	In the final week, I will finish all the reports and polish up all my code and repository.


Term 1 Gantt Diagram
Finish reports and problems 1&2
Start problem 3-4, write optimal params report
Fin tune problem 2 & improve reports
Start problem 2
Fine tune problem 1 & reports writing
Finish Problem 1
Set up workspace, start Map Reduce Report & Problem 1
Study Spark
Study Hadoop & Big Data Analytics
16/09/22
26/09/22
06/10/22
16/10/22
26/10/22
05/11/22
15/11/22
25/11/22
05/12/22
15/12/22
Start Date
Duration (days)

TERM 2 (9 JANUARY TO 24 MARCH 2023)

TASK	START DATE	END DATE	DESCRIPTION
EVALUATE PERFORMED PROGRESS AND ADAPT THE PLAN	09/01/2023	15/01/2023	Based on the first term progress and documentation of the new tool to be used, adapt my strategy and plan.
RESEARCH THE PAGE RANK AND MACHINE LEARNING MODEL	16/01/2023	29/01/2023	Research the requirements, feasibility, theory and tools for the two problems.
FINISH PAGE RANK PROBLEM AND REPORT	30/01/2023	12/02/2023	Finalise the page rank problem and run it on Wikipedia data in local mode. Finish writing the draft for the page rank report
BUILD MACHINE LEARNING DATASET	13/02/2023	19/02/2023	Research the model categories and created the dataset.
CREATE MACHINE LEARNING MODEL (UNDISTRIBUTED VERSION)	20/02/2023	26/02/2023	Create the machine learning model.
CREATE DISTRIBUTED MACHINE LEARNING MODEL AND REPORT	27/02/2023	05/03/2023	Create the distributed version of the machine learning model and finish the machine learning draft report.
CREATE HISTORICAL SIGNIFICANCE PROGRAM AND RUN PROGRAMS ON THE WHOLE WIKI DATASET	06/03/2023	12/03/2023	Create the historical significance program and deploy the model on the cluster, fine-tuning their performance.
REPORT WRITING	13/03/2023	24/03/2023	Finalise the final report.


Term 2 Gant Diagram
Report writing
Create histroical significance program and run programs on the whole wiki dataset
Create Distributed Machine Learning Model and report
Create Machine Learning Model (Undistributed Version)
Build Machine Learning dataset
Finish page rank problem and report
Research the page rank and lachine learning model
Evaluate Performed Progress and adapt the plan
04/01/2023 14/01/2023 24/01/2023 03/02/2023 13/02/2023 23/02/2023 05/03/2023 15/03/2023

1. THEORETICAL BACKGROUND

In the following section, I will discuss the theoretical side of the tools, methodologies, and programs I used for my project. The reader will firstly be introduced to Massive Scale Data Analytics, then to the Page Rank algorithm and finally to Machine Learning and Natural Language Processing. This section will help the reader get a better grasp of the programs explained in the implementation part.

1.1. MASSIVE-SCALE DATA ANALYTICS

In the following section, we will have a look at key Massive Scale Data Analytics tools, frameworks and methodologies. The reader will firstly be introduced to the problem of Massive Scale Data processing. Then we will present a conventional Big Data framework, Hadoop with its programming paradigm, Map Reduce. Finally, we will have a look at Spark, the current de facto enterprise-level solution for Big Data processing and understand why it is better than Hadoop Map Reduce.

1.1.1. DATA STORAGE & ANALYSIS

Data is power. Every day, each one of us and the devices we are using produces an unimaginable amount of data. As internet access became a commodity in the early 2000s, the problem of data flooding became reality. Some overwhelming examples for nowadays big data usage are Instagram which has over forty billion photos and videos in storage and the Statista estimation that there will be 175 Zettabytes of internet data by 2025 [9]. Knowing how to manipulate it, extract useful information from it, and apply it to solve pressing problems or answer important questions is what makes Big Data analytics such an important discipline.

In the early 2000s, the need to create an efficient programming paradigm and framework to process big data was dire [1]. There was a lot of data but not an efficient way to store & analyse it. This need was further fuelled by hardware development stalling. Capacities of hard drives have increased rapidly and greatly but the access speeds have not kept up [1]. As an example, in the nineties, a drive could store 1300 MB of data with a transfer speed of 4.4 MB/s, so all the data on the disk could be read in under 5 minutes [1]. Nowadays, drivers with terabytes capacities are usual but the transfer speed is only 100 MB/s, resulting in a time of more than 2 hours to read one terabyte of data off the drive. This problem affected the initial Big Data processing attempts. [1]

Now imagine one hundred drivers working in parallel to read the 1 terabyte drive. This, in theory, will be one hundred times faster and the drive will be read in 2 minutes. In practice, this approach comes with loads of problems. [1]

The first problem is hardware failure [1]. If you start using hundreds of pieces of hardware, the probability that one or more of these will fail is high. How to manage this? The solution is replication. If copies of data are made and spread over the cluster of machines then if a machine fails, the data is not lost. [1]

The second problem is combing the data after it was processed in parallel [1]. Solving this problem is notoriously challenging but here the MapReduce algorithm comes to the rescue, providing a programming model that abstracts the problem from disk reads and writes into a computation over sets of keys and values [1].

This is in essence what Hadoop provides: a reliable shared storage and analysis system [1]. The storage is provided by the Hadoop Distributed File System (HDFS) and analysis by MapReduce. [1]

1.1.2. HADOOP

Apache Hadoop is a framework that enables distributed processing on huge clusters with hundreds to thousands of nodes to store and process in batch petabytes of data [2]. The strength of the Hadoop cluster design is that it can be constructed with commodity hardware with high failure rates [1]. Hadoop efficiently manages the failures without user intervention making Hadoop clusters very easily scalable and a wise enterprise choice [4]. Hadoop's main approach is to focus the computation power on the data, rather than disk reads/writes, therefore significantly reducing network I/O. Across the network, Hadoop manages the business logic with the use of two types of nodes: data node and master node (see figure 1) [2, 4].

A master node oversees the resource allocation and scheduling and has the role of a manager in the cluster for the data nodes [4]. The robustness of the master nodes is crucial for the success of the cluster. This is because it is the single fail point for the program [4]. If the master node of a cluster dies during job execution, then the whole job fails. Even though multiple master nodes can co-exist for large clusters, the failure of any of them means the failure of all its managed data nodes and therefore the failure of the whole job. [4]

A data node does the actual business logic (running the MapReduce program code) and is managed by a master node [1]. Unlike master nodes, data nodes can fail without affecting the result of the job. This is because of the replication discussed before. If a data node dies (and is not revived after a couple of attempts by the master) then its data and execution will be allocated by the master to another data node. [1]

This implicitly affects the overall job performance and is ideal that as few data node failures happen during job execution. However, for large clusters, node failure is inevitable, but Hadoop has the right tools and approach to handle this with the use of its component: Yet Another Resource Negotiator (YARN) [4]. All negotiations, resource allocation and scheduling of all the nodes are done graciously in the background by YARN without any programmer intervention [1]. Consequently, the focus of a programmer using Hadoop becomes to create efficient MapReduce programs and fine-tune the parameters for the execution. [1]

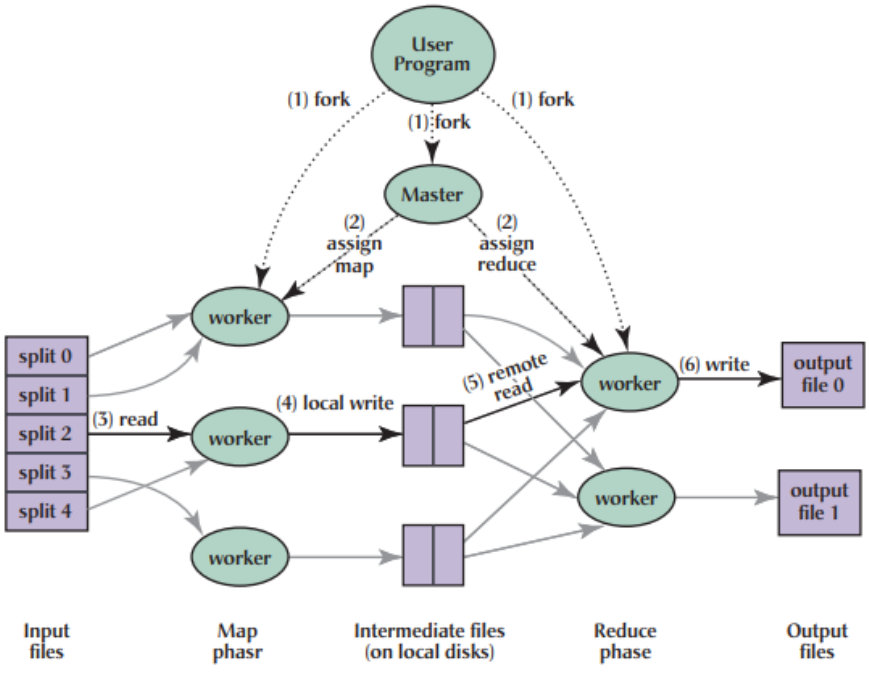


Figure 1 Hadoop master-worker architecture [1]

Hadoop has four main components:

1. **Hadoop Common**: the internal code component & API
2. **Hadoop Distributed File System (HDFS):** used to store the data
3. **Yet Another Resource Negotiator (YARN):** used to manage resources (CPU & memory)
4. **MapReduce algorithm program**: the actual program that is used to process the data

1.1.3. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

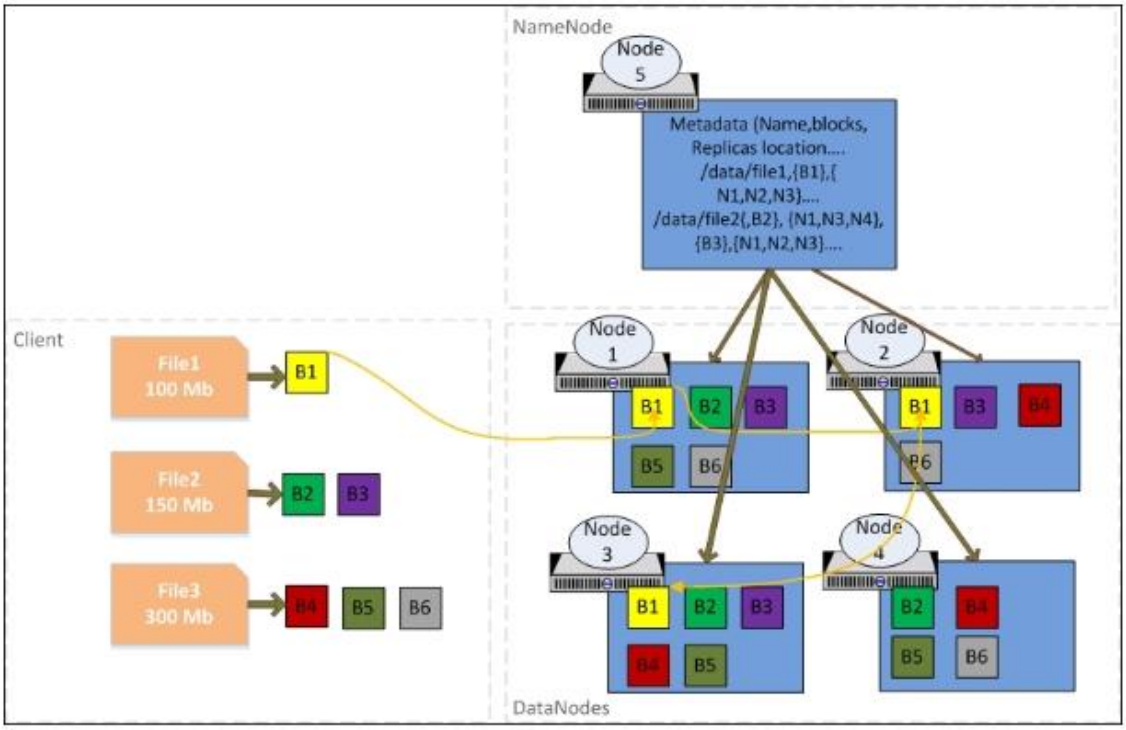


Figure 2 HDFS overview [2]

A distributed file system (DFS) is a file system that is distributed amongst multiple servers at multiple locations [11]. Programs that are running on a DFS can access files and share resources from any of the nodes that are on the DFS which provides location transparency and redundancy. [11] For Hadoop, having an efficient and dependable DFS is crucial given that its architecture is created to accommodate thousands of nodes. Concisely this is what HDFS provides, a highly scalable and reliable DFS that can easily be integrated into a large cluster with commodity hardware. [2]

Files stored in the HDFS are split into chunks of 120MB in size and distributed across the network [1]. Each block is replicated by default three times to handle hardware failure [1]. For example, if you have a file of 110MB this will count as a single split. If you have a file of 200MB this will be split into two files: a 120MB split and an 80MB split. As can be seen in figure 2, each file sent by the client has split accordingly to its size and replicated three times across the network. [11]

Therefore, it is ideal when pre-processing your data that your data is split into blocks of the size of the HDFS block each. If this is not done, unnecessary data splits will be done, and the overall runtime will be affected. One important concept in Hadoop is data granularity, i.e., Hadoop works best with a smaller number of larger files than with a larger number of small files [1]. This is because each input split is managed individually and in isolation from the others. To do so, Hadoop instantiates a new Java Virtual Machine (JVM) for each input split [1]. On one

side this ensures that if a mapping job fails, then the whole job does not fail. On the other side, this is a resource-consuming process. Each of these settings can be configured (for example the block or split size) but this needs to be treated with diligence as it can greatly influence the performance of the job. [2, 11]

1.1.4. MAPREDUCE

MapReduce is a programming model and implementation for processing and generating large datasets and therefore can be applied to many real-world tasks [1]. Some examples are trade processing, weather data analysis, or cancer detection. The possibilities are virtually endless as long there is data. Users customise their program in terms of a map and a reduce function, and the designated run time system (I.e., Hadoop) automatically parallelizes the job across a cluster of machines which can vary from just one to thousands of nodes [1]. With the aid of the MR job, in the background, Hadoop takes care of machine failures, scheduling, job allocation management and inter-machine communication such that efficient use of network and disks is achieved. [1, 5]

1.1.4.1. HOW DOES THE MAP REDUCE WORK?

The program is split into two main parts: map & reduce. The Map method receives initially a key-value pair representing an initial key assigned by Hadoop and the value to be processed [1]. After processing, a new set of key-value pairs is produced. The reduce method merges all the intermediate mapped values that have the mapped key and performs some extra logic [1]. The map and reduce functions follow this general format:

```
map: (K1, V1) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

Figure 3 MapReduce Format [1]

The map input key and value types (K1, V1) are different from the map output types (K2, V2). The reduce input, however, must be the same type as the map output, although the reduce output types may be different again (K3, V3). Here is how the Java interface implements this form:

```java
public class DemoReducer<K2, V2, K3, V3> extends Reducer<K2, V2, K3, V3> {
    public void reduce(K2 key2, Iterable<V2> values, Context context) throws IOException, InterruptedException {
    }
}
```

Figure 4 Java Reducer interface

```java
public class DemoMapper<K1, V1, K2, V2> extends Mapper<K1, V1, K2, V2>{
    public void map(K1 key1, V1 value1, Context context) throws IOException, InterruptedException {
    }
}
```

Figure 5 Java Mapper interface

In figures 4 and 5, the Context object is shared between the reducer and the mapper and is used for emitting key-value pairs. In the map, method context is used to map key-value pairs which are then accessed through the Inerrable in the reduce method. But how are the mapped values combined by key?

The combiner is the Hadoop primitive that oversees combining values by key, and it is an important part of the Map Reduce program [1]. Combining records by key is in the backend by Hadoop by default but can be customised if necessary. In that case, the above workflow transforms to [1]:

```
map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)
```

Figure 6 Map, Combine & Reduce Format [1]

1.1.4.2. ANATOMY OF MAPREDUCE IN HADOOP

The process is illustrated in figure 7. At the highest level, there are four independent agents:

- The Client, which submits the MapReduce job request
- Job Tracker, which coordinates the job run
- Task Tracker, which runs the tasks the job has been split into
- HDFS, which is used for sharing job files between the other agents

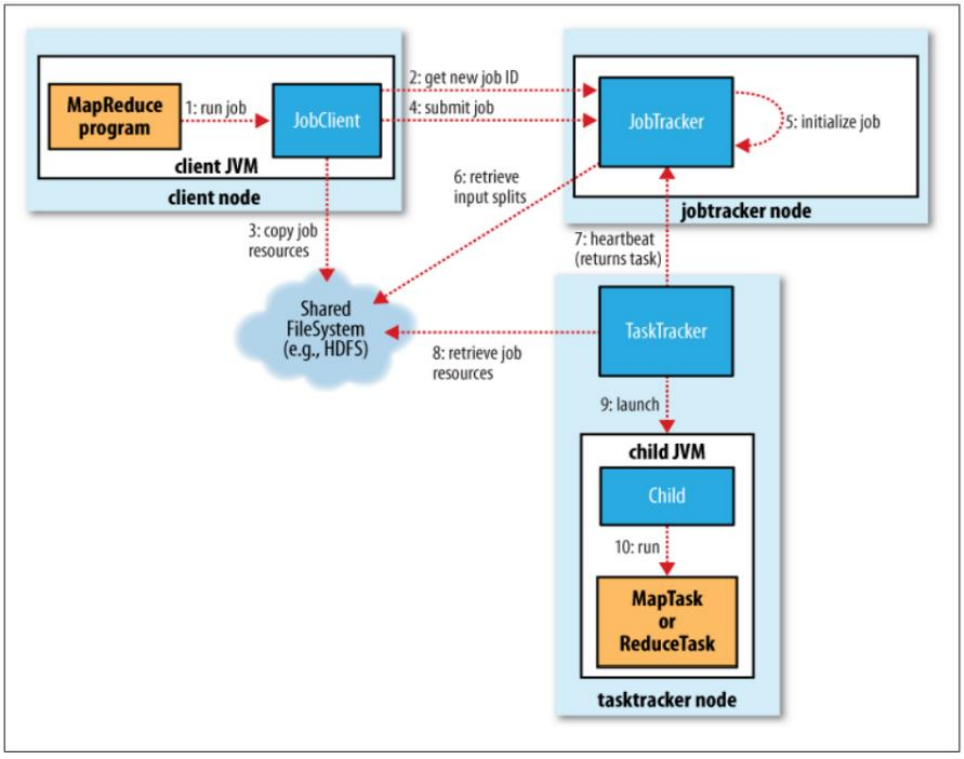


Figure 7 How Hadoop runs an MR job [1]

1.1.5. SPARK

Spark is a fast enterprise-grade large-scale data processing engine which can interoperate with many distributed systems (like HDFS) [2]. Nowadays the majority of new applications are using Spark for its ease of use and speed. Compared to Hadoop, Spark can be 100 times faster in memory and 10 times faster on disk [2].

Hadoop and MapReduce have been the de facto solution for processing massive data at high performance. However, MapReduce lacks performance in iterative computing where the output of multiple jobs has to be written to the HDFS [2]. This is because of the downsides of the MR framework.

To understand why let's have a glance at the history of computer trends to understand how computer trends have changed over the years [2]. It all boils down to costs. In the 90s, the network was cheaper so the trend was to **reference the URI** [2]. In the early 2000s, **replication** became cheaper and frameworks like Hadoop came into popularity [2]. Currently, since 2010 memory processing has become cheaper so **re-computing** has become the norm with the raise in popularity of Spark [2].

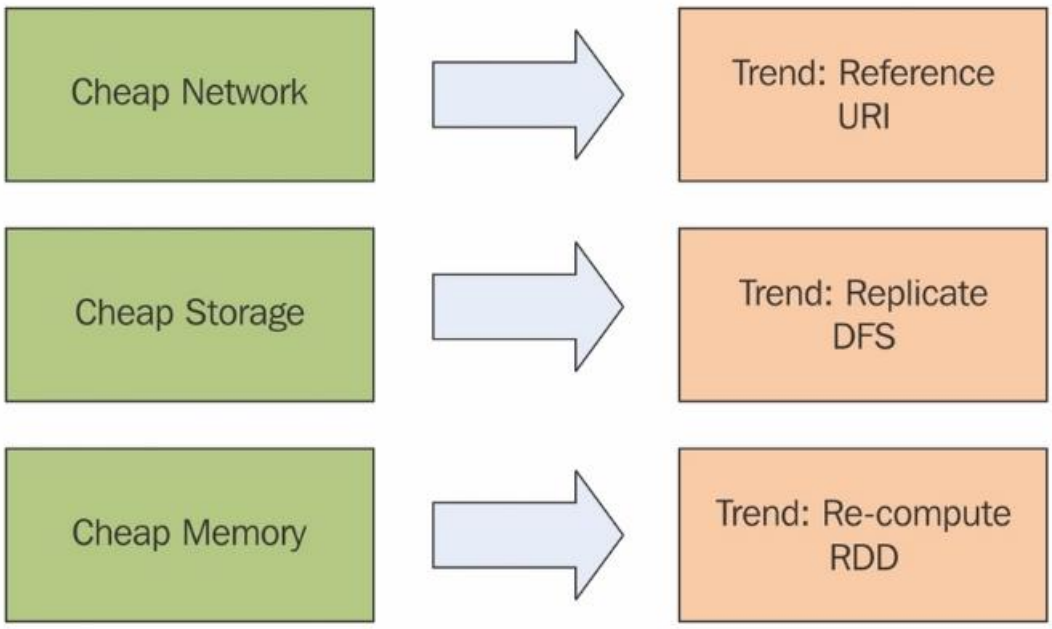


Figure 8 Computing Trends [2]

1.1.5.1. WHAT CHANGED OVER TIME?

The tape is dead, then the disk has become the new tape and currently, the SSD has become the new disk [2]. It was mentioned above that Spark is 100 times faster than Hadoop in memory. This is because of Spark's in-memory processing (in RAM). This can be best illustrated in the figure below where we can see and compare the transfer rates to the CPU from different mediums [2]. Disk to CPU is 100MB/s, SSD to CPU is 600MB/s and over the network to CPU is 1MB/s to 1GB/s [2]. However, RAM to CPU is very fast, 10GB/s typically [2].

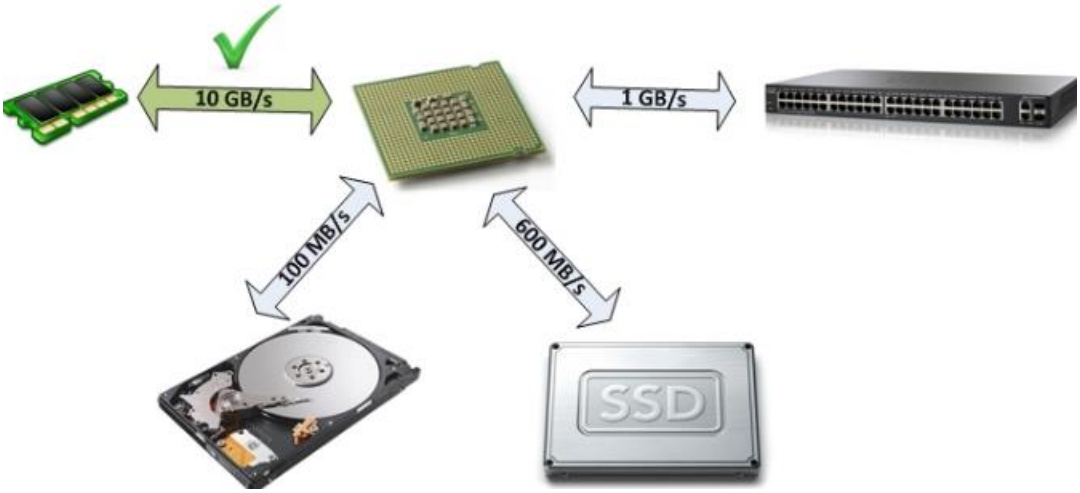


Figure 9 Why Memory? [2]

1.1.5.2. WHAT IS SPARK?

Here are all the reasons why Spark is a force to reckon with in Big Data Analytics:

- Very fast parallel processing in memory [2].
- Written in Scala, it's both object-orientated and functional [2].
- Suitable for iterative programs since each iteration is computed in memory [2].
- High compatibility with a large number of languages and frameworks. It provides native support for Scala, Java, Python and R [2].
- Clean and straightforward API with a high-level optimised implementation of many functionalities [2].
- Powered by a large stack of libraries such as Spark SQL for fast querying, GraphX for graph processing and algorithms, and GraphML for machine learning program scaling [2]. Furthermore, all these powerful features can be combined efficiently in the same application.
- Not opinionated in infrastructure scaling. It can work with HDFS, Mesos, Hadoop, standalone cluster managers, on-premise hardware or the cloud [2].

1.1.5.3. WHAT SPARK IS NOT

Spark is not an all-in-one solution like Hadoop. Hadoop provides HDFS for storage and MR for computing [2]. However, Spark doesn't provide any specific storage environment [2]. Spark is mainly a powerful computing engine meant to be plugged in with different distributed storage environments [2]. Spark is not Hadoop nor Hadoop is necessary for Spark to run. However, Hadoop and Spark work together very well. Therefore, the best way to think about Spark is not Hadoop's replacement, but MapReduce's replacement [2].

1.1.5.4. RESILIENT DISTRIBUTED DATASET (RDD)

RDDs are a fundamental data unit in Spark [2]. All Spark computations revolve around creating and performing operations on RDDs [2]. They are immutable collections partitioned across the network which can be re-computed if a partition is lost [2]. They can be created by applying transformations on data using data flow operations like map, filter, group-by, and reduce [2]. They can be cached in memory across parallel operations. Why are they called RDDs?

- Resilient: if an RDD partition block is lost, it can be re-computed [2].
- Distributed: distributed across clusters [2].
- Dataset: RDD data can come from a file or programmatically [2].

How is an RDD can be created?

Method 1: Programmatically

The first way to create an RDD is programmatically, by parallelising a collection (i.e. a python list) [2].

```
// Parallelize a list in Java
JavaRDD<String> myRDD = sc.parallelize(Arrays.asList("hdfs",
"spark", "rdd"));

# Parallelize a list in Python
myRDD = sc.parallelize(["hdfs", "spark", "rdd"])

// Parallelize a list in Scala
val myRDD= sc.parallelize(List("hdfs", "spark", "rdd"))
```

Figure 10 Creating an RDD [2]

Method 2: Reading from a file

The second method is loading a file's contents into an RDD [2]. This can be done from a local file system file or a distributed environment like HDFS. The input can be a single file or an entire directory.

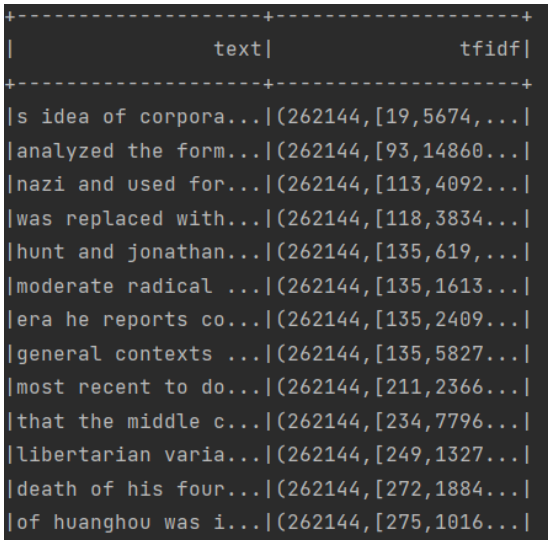
```
// Read a file in Java
JavaRDD<String> inputRDD = sc.textFile("/path/to/input");

# Read a file in Python
inputRDD = sc.textFile("/path/to/input")

// Read a file in Scala
val inputRDD = sc.textFile("/path/to/input")
```

Figure 11 Creating an RDD [2]

1.1.5.5. DATAFRAMES

We have seen RDDs and their importance to Spark. Nowadays, RDDs are decreasing in popularity in favour of Dataframes. A Dataframe is a data structure which structures data into a 2D table of rows and columns [2]. This is very similar to SQL tables and therefore Spark supports native SQL queries adapted to Dataframes. In modern data analytics they are one of the most common data structures because of their high flexibility and intuitive of data storage and transformations [2].

Spark Dataframes are organised based on a schema (like in SQL) of columns of types and names [2]. These enable developers to work efficiently with structured or semi-structured data, allowing for higher-level abstractions [2].



```
+--------------------+--------------------+
|                text|               tfidf|
+--------------------+--------------------+
|s idea of corpora...|(262144,[19,5674,...|
|analyzed the form...|(262144,[93,14860...|
|nazi and used for...|(262144,[113,4092...|
|was replaced with...|(262144,[118,3834...|
|hunt and jonathan...|(262144,[135,619,...|
|moderate radical ...|(262144,[135,1613...|
|era he reports co...|(262144,[135,2409...|
|general contexts ...|(262144,[135,5827...|
|most recent to do...|(262144,[211,2366...|
|that the middle c...|(262144,[234,7796...|
|libertarian varia...|(262144,[249,1327...|
|death of his four...|(262144,[272,1884...|
|of huanghou was i...|(262144,[275,1016...|
```

Figure 12 Dataframe Format

1.1.5.6. RDDS VS DATAFRAMES

Because of their clear structure, Spark can optimize operations on them much better [2]. Therefore, they offer superior performances to RDDs [2]. RDDs on the other hand, enable developers to work with unstructured data and be “closer” to the input data, trading performance for control.

1.1.5.7. TRANSFORMATIONS AND ACTIONS

There are two main operations which can be performed on both Dataframes and RDDs. These are transformations and actions [2].

Transformations

- Define new RDDs/Dataframes based on existing RDDs/Dataframes [2].
- Can be created through operations like map, reduce, filter [2].

Actions

- Return values from RDDS or Dataframes [2].
- Can be done through operations like forEach (iterating), collect (collecting all partitions into one or saveAsTextFile (saving an RDD/Dataframe into a text file) [2].

1.1.5.8. CACHING

One of the unique and powerful features of Spark is the ability to cache RDDs or Dataframes in memory [2]. The way to do so is the following:

```
>>> myRDD.cache()
>>> myRDD.persist()
```

Figure 13 Caching [2]

Efficient caching can greatly improve programs' performance [2]. This is because performed operations saved are saved in memory and don't have to be re-computed at later transformations [2]. There are multiple levels of caching. Choosing which one depends on your cluster and program requirements. Here they are:

Storage Level	Meaning
MEMORY_ONLY	Store RDDs in memory only. A partition that does not fit in memory will be re-computed.
MEMORY_AND_DISK	Store RDDs in memory and a partition that does not fit in memory will be stored on disk.
MEMORY_ONLY_SER	Store RDDs in memory only but as serialized Java objects.
MEMORY_AND_DISK_SER	Store RDDs in memory and disk as serialized Java objects.
DISK_ONLY	Store the RDDs on disk only.
MEMORY_ONLY_2 MEMORY_AND_DISK_2	Same as MEMORY_ONLY and MEMORY_AND_DISK, but replicate every partition for faster fault recovery.
OFF_HEAP (experimental)	Store RDDs in eTachyon, which provides less GC overhead.

Figure 14 Caching Levels [2]

1.1.5.9. SERIALIZATION

Every task sent from the driver to the executors gets serialised [2]. Using the right serialization network can greatly improve the performance of your programs. Spark provides two serialization libraries: Java serialization and Kryo serialization. By default, Spark uses Java serialization [2].

Java serialization is intended to be general purpose and therefore it's flexible but it is slow and this can lead to large serialised objects [2]. This means reduced performance in distributed environments where large amounts of data are rapidly transferred over the cluster. The solution for this problem is Kryo serialization which is used for better object compactness, better read/write speeds and overall better performance [2].

1.2. PAGE RANK

Page rank is a very popular and well-known algorithm used by search engines, like Google, to rank web pages in search results [19]. The algorithm was initially developed by one of Google's co-founders Larry Page. In a nutshell, the algorithm's main idea is that importance of a web page can be measured by the number and quality of other pages that link to it [19].

1.2.1. WHAT IS PAGE RANK?

You can think of page rank as a "vote", by all the other pages on the Web about how important a certain page is [19]. A link to a page represents a vote of support and no links mean no support [19]. Not voting in this sense doesn't penalize the rank of the page, it's rather an abstention from voting [19].

The algorithm starts by initially assigning all webpages the same score. The algorithm then iteratively assigns a score to each web page based on the links pointing to it. A web page rank score increases with the number of inbound links it has [19]. However, not all links are equal. A link from a better-scoring website will be more important than one from one with a smaller page rank score [19]. This iteration continues until a point of convergence is reached. This means that the amount of change done after each iteration is smaller than a pre-defined threshold [19].

Page Rank Formula [19]

Let u be a webpage.

Let F_u the set of pages that point to u

Let B_u the set of pages that u points to

Let $N_u = |F_u|$ the number of links from u

Let c a factor used for normalization

Page Rank: $R = c \sum_{v \in B_u} \frac{R(u)}{N_u}$

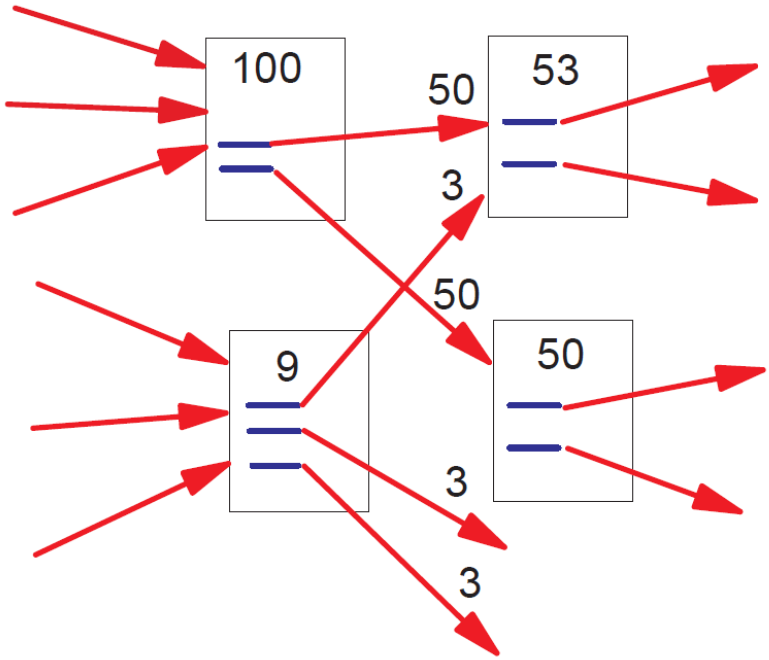


Figure 15 Page Rank Calculation [19]

1.2.2. COMPUTING PAGE RANK

The computation of page rank is pretty straightforward. For this section, I am citing [19].

Let S be a vector of webpages (for example E)

Let A be a square matrix of rows and columns corresponding to the web pages. Let $A_{u,v} = {}^{1}/_{N_u}$ if there is an edge from u to v and $A_{u,v} = 0$ if not.

Let δ be the converge threshold

Let ε be the page rank iteration change

Algorithm:

$R_0 \leftarrow S$

loop:

$$R_{i+1} \leftarrow AR_i$$

$$d \leftarrow \|R_i\|_1 - \|R_{i+1}\|_1$$

$$R_{i+1} \leftarrow R_{i+1} + dE$$

$$\delta \leftarrow \|R_{i+1} - R_i\|_1$$

while $\delta > \varepsilon$

1.2.3. ISSUE OF DANGLING NODES

One issue with this model described is dangling links [19]. A dangling node in a page rank graph is simply a webpage with no outgoing links [19]. They affect the overall performance of the algorithm because their weight cannot be intuitively distributed [19]. Furthermore, in a connection graph such as the web or Wikipedia articles, there is a large number of them. So how to deal with them? When looping the page rank, introduce a small chance (typically 15%) that the page rank graph resets and visits another node [19]. This way, even if there are dangling nodes in the graph, they will eventually be skipped.

1.2.4. CONVERGENCE PROPERTIES

We have seen the notion of convergence. What is typically a good converge threshold and how long does it take to achieve it? Normally the convergence threshold is around 0.0001, meaning that the algorithm stops when it reaches a change in page rank values smaller than this number [19]. As can be seen in the figure below, even for a large number of websites and links (322 million), it takes roughly 52 iterations to reach convergence [19]. This suggests that the program scales well even for extremely large amounts of data, the scaling factor being linear in log(n) [19].

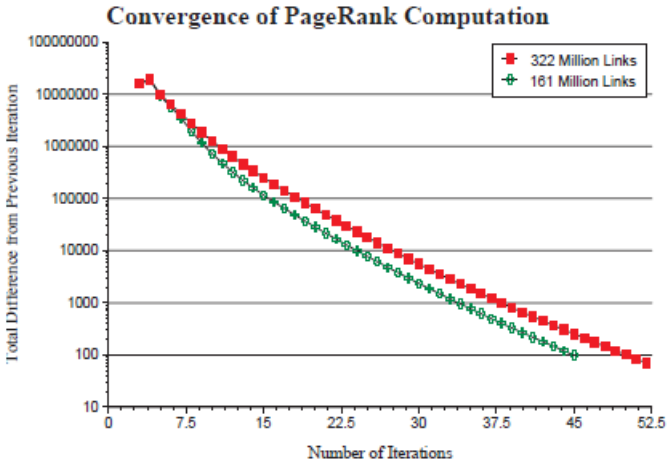


Figure 16 Page Rank Convergence [19]

1.3. MACHINE LEARNING

Machine learning addresses the question of how to build systems that can automatically improve themselves through experience [21]. It is one of the most popular fields in Computer Science, laying at the core of Artificial Intelligence and Data Science. The applications of machine learning are seemingly limitless, spanning from technology and commerce to health care, manufacturing, education, or politics. An example of an application of machine learning is Cancerous Tumours Image Recognition where models are trained to recognise cancerous tumours in images, with accuracies sometimes exceeding that of experienced medical staff [21].

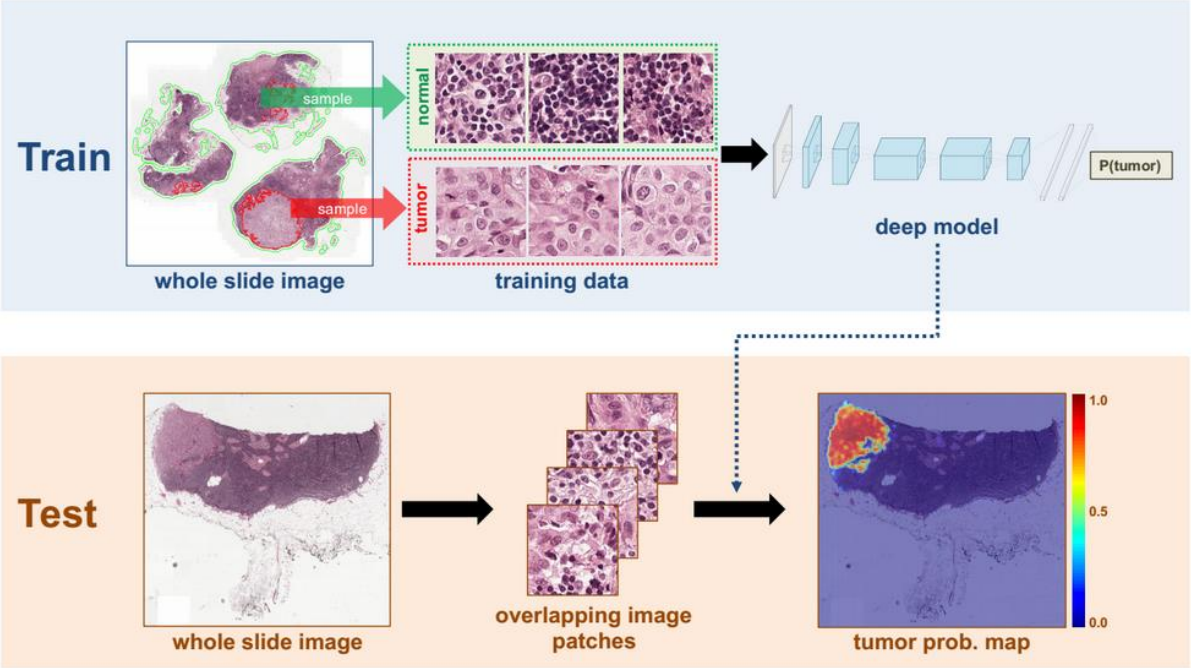


Figure 17 Cancer Detection with Machine Learning [28]

The classical approach to solving a problem is through an algorithm, a highly specific and optimised set of steps. This is called a top-down approach. Machine learning takes the other approach: bottom-up. Rather than trying to find the best algorithm for a problem, machine learning models try to find patterns in the data they are fed during a process called training until the model is good enough at recognising those patterns [27]. After this, the models are tested on data they haven't seen before.

This approach is great at generalising the solutions to problems without being concerned with why the solution is that way. Essentially, through repeated learning, models fine-tune a set of parameters and then make predictions for new entries [27]. There are two main approaches to the learning process: supervised and unsupervised learning. In supervised learning, the models' predictions are compared to the correct predictions (which are hand labelled) and then the model slightly changes its parameters accordingly. In unsupervised learning, the model can just find patterns within the data, without the need for human intervention through hand labelling the samples [27].

1.3.1. CREATING A MACHINE LEARNING MODEL

Crating a machine learning model nowadays is easy because of the large number of online resources, packages and libraries that simplify this process. However, a certain number of steps are required to find the model that suits your requirements. This gradual process is called a pipeline [27].

The first step is understanding what kind of learning approach and model you require. For this, you need to figure out what kind of machine-learning model problem you are trying to solve. There are two kinds of machine learning problems: regression and classification [27]. Regression algorithms take a sample input and make a prediction which has a continuous value. In other words, the possible results of the predictions are infinite. Some examples of this are models predicting quantities like money, score, and weight [27]. Classification algorithms take a sample input and make a prediction which has discrete values. In other words, the possible results of the predictions are finite (from a pre-defined set of output classes) [27]. The cancer detection example from above is an example of a classification problem because the result of the model on the sample image will be cancerous or non-cancerous.

The second step is gathering quality data and pre-processing it. Since data is at the heart of your model, ensuring that it is of good quality and relevant to the problem you are solving is crucial. This data needs to be cleaned and normalised. To detect patterns efficiently, a model needs only relevant information and no “noise” [21]. For example, when analysing a text, you might not be interested in non-alphabetic characters since they bare no importance to the lexicographical pattern finding.

The third step is choosing the right model for the problem. This can be done through research of similar problems or creating several types of models in parallel and comparing their accuracy and performance.

The fourth step is splitting the data into a training and testing set and training the model solely on the training data. It is crucial not to mix the 2 sets at all (also called “data leaking”) [27]. This is because we want to see how the model behaves on unseen data, which would be a good indication of its performance on real-world data. Two important terms for training are underfitting and overfitting. An under-fitted model can neither model training data efficiently, nor generalise the pattern matching to unseen data [27]. An overfitted model trains its training data too well. This means that the model detects too specific training data patterns and can’t properly apply its training to unseen data. In other to create a properly fitted model, we need to ensure that not only our data is of good quality, but it’s also general enough [27].

The fifth step is model evaluation and deployment. We check the accuracy of the model on the testing set. As a rule of thumb, accuracy is normally considered good if it’s above 80% since this means that the model can generalize very well. If it’s too close to 100%, this actually might be an indication of some issue with our model or data [27].

1.3.2. SUPPORT VECTOR MACHINE (SVM)

After comparing the performance of a couple of models on a smaller portion of the dataset, the model I chose for my project is an SVM because of its simple implementation and good performance. SVMs are supervised learning models with associated learning algorithms that analyse the data used for classification and regression problems [22]. Besides being efficient at linear classification, SVMs can perform good non-linear classification using what is called a kernel trick [22]. This means implicitly mapping inputs into a higher dimensional feature space [22]. Thus, classification error is minimised because the margins are drawn in such a way that the distance between the margin and the classes is maximum [22]. Fine-tuning the performance of the model can be done by fine-tuning 2 parameters: C (regression parameter) and N (the maximum number of iterations the SVM re-trains) [22]. These two can greatly affect your model accuracy.

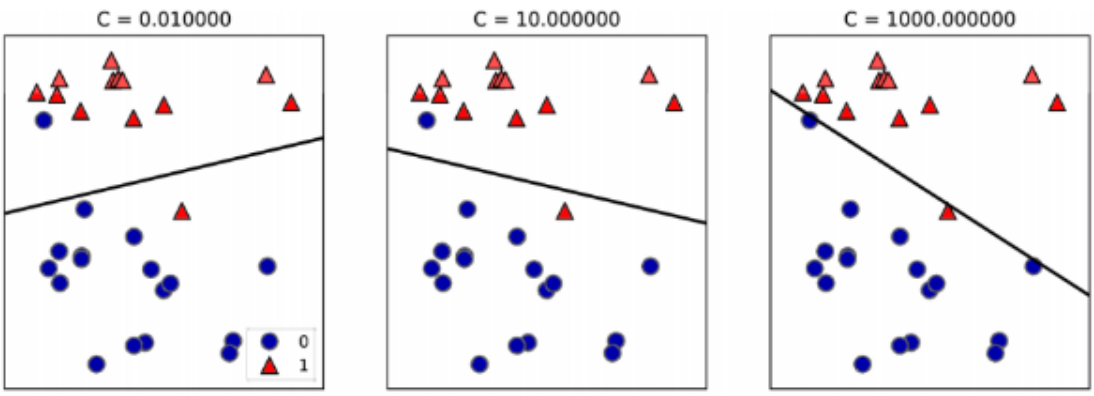


Figure 18 SVM overview [27]

1.3.3. TENSORS & LEARNING

We have seen the way that a machine-learning model works. Now, what does the input look like? The input to an NLP model has to be a tensor. A tensor is a mathematical object which describes a multilinear relationship between sets of algebraic objects related to a vector space [30].

In simpler terms, it is an N-dimensional array of data [30]. Why N dimensions? Well, consider the image classification problem. Each image is made of bytes, let's say 720x720 pixels. And if it's a coloured image, each pixel is represented by 3 values: Red, Green, and Blue (RGB) which can form any colour. So, the sample input image is a 3rd-order tensor (720x720x3).

You can imagine each dimension in the tensor as a different feature of the input sample. Now, how to handle text? The text itself can be transformed into a vector of features through a process of tokenization and TF-IDF which will be presented in the next sections.

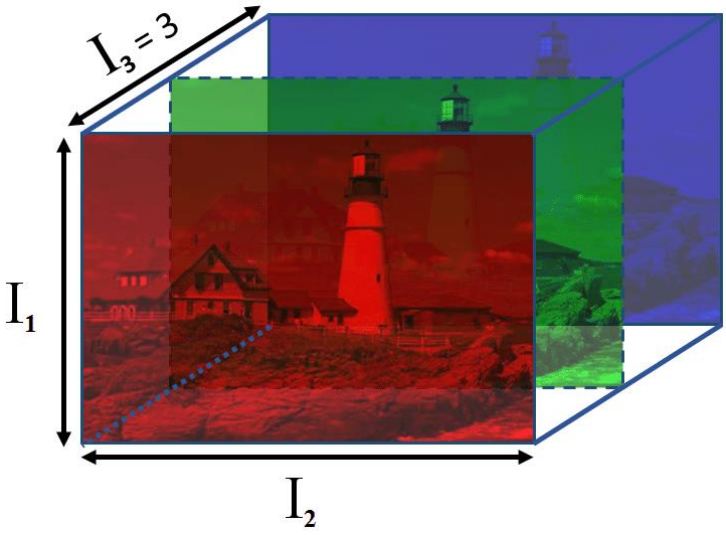


Figure 19 Image Tensor [30]

1.4. NATURAL LANGUAGE PROCESSING (NLP)

Natural language processing is a discipline that employs computational techniques for learning, understanding and producing human language content [20]. In the early days of NLP, the focus was on automating the analysis of the linguistic structure of a language and developing basic technologies like machine translation, speech recognition and speech synthesis [20].

Nowadays, NLP is used and improved constantly for real-world applications such as creating spoken dialogue, speech-to-speech translation engines, mining social data about health, finance or political views as well as identifying sentiments towards products, services, and individuals [20]. This field is rapidly changing and is benefiting from huge success and popularity, with powerful and useful tools like ChatGPT completely changing the economy and the world for good.

1.4.1. TOKENIZATION

Tokenization is the first step in any NLP pipeline as it has an important effect on the rest of the pipeline [26]. A tokenizer breaks unstructured natural language text into chunks of information that can be considered discrete elements [26]. This process turns an unstructured text document into a numerical tensor depending on the n-gram configuration given [26]. An n-gram is a sequence of n words: a bi-gram (2 words) like “please come”, a tri-gram (3 words) like “go home now”, etc… [26]

1-Gram	2-Gram	3-Gram
The	The pizza	The pizza has
pizza	Pizza has	Pizza has not
has	Has not	Has not a
not	Not a	Not a bad
a	A bad	A bad taste
bad	Bad taste	
taste		

1.4.2. STOP WORDS REMOVAL

Stop word removal is one of the most popular and powerful steps used in NLP pre-processing [31]. Essentially, commonly used words that occur in a language (i.e. “is”, “a”, “the”) are removed. This is because these words bare no significance in text categorization since a model would be looking for more unique words to find patterns to match a text to a category [31]. Furthermore, by artificially decreasing the corpora’s size, we will increase the model performance and will empower the model to perform efficient pattern matching on relevant data [31].

1.4.3. LEMMATIZATION

Lemmatization is the process of creating a normalised version of a word [24]. Lemmatization is an important part of NLP because it provides a productive way of generating generic words for search engines or labels for concept maps [24]. Lemmatization is similar to word stemming but does not require the production of a stem of the word. However, it requires replacing the suffix of the word with a different word suffix to get the normalized word form [24]. For example, the suffixes of the words working, works, and worked would change to get the normalised form corresponding to the infinitive tense: work [24]. Lemmatization reduces ambiguity and extracts the essential information from the text, ensuring that the text pattern matching is smoother [24].

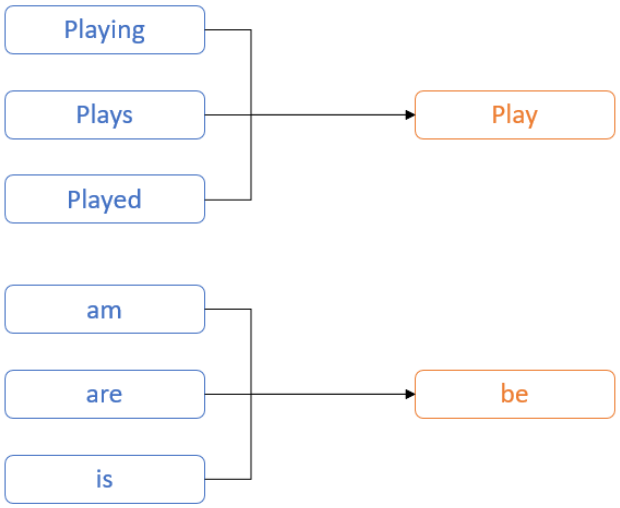


Figure 20 Lemmatization overview

1.4.4. TERM FREQUENCY – INVERSE DOCUMENT FREQUENCY (TF-IDF)

A TF-IDF is a popular research method in NLP and Data Science which determines the relative frequency of words in a specific document through an inverse proportion of the word over the entire document corpus [25]. In this method, TF-IDF uses two main elements: TF- term frequency of terms in the document and IDF: inverse document frequency of a term [25]. This measure can be used to determine the importance and relevance of certain words within a document [25].

A parameter which can greatly influence the quality of the TF-IDF is the number of features to use. This number has to be greater or equal to the number of unique words in your dataset (also called corpora) [25]. This is because the TF-IDF hashes each word encountered to a unique numerical value and if the number of features is smaller than your corpora size, different words will be hashed to the same value [25]. In the following picture, the number of features is 262144 so each entry in the TF-IDF will be represented by a sparse vector of 262144 entries with the sentence words at the corresponding hash position and the rest of the elements with the value 0.

```
+--------------------+--------------------+
|                text|               tfidf|
+--------------------+--------------------+
|s idea of corpora...|(262144,[19,5674,...|
|analyzed the form...|(262144,[93,14860...|
|nazi and used for...|(262144,[113,4092...|
|was replaced with...|(262144,[118,3834...|
|hunt and jonathan...|(262144,[135,619,...|
|moderate radical ...|(262144,[135,1613...|
|era he reports co...|(262144,[135,2409...|
|general contexts ...|(262144,[135,5827...|
|most recent to do...|(262144,[211,2366...|
|that the middle c...|(262144,[234,7796...|
|libertarian varia...|(262144,[249,1327...|
|death of his four...|(262144,[272,1884...|
|of huanghou was i...|(262144,[275,1016...|
```

Figure 21 TF-IDF overview

2. IMPLEMENTATION

In the following section, I am going to describe my implementation for the two parts of the project. For each part, I am going to present the problems' overview, implementation, deployment, experiments and performance analysis performed and present my optimal parameters settings. For each part I will also have a discussion about the results.

2.1. DEPLOYMENT

Before going into the implementation of the programs, let's have a look at the environment in which the programs were developed. For ease of development and preservation of the cluster resources, I separated my workflow into two parts: pseudo-distributed mode on my laptop and distributed mode on the Big Data Royal Holloway Cluster. I developed my programs locally on my machine and after that, I packaged and deployed them on the cluster where I performed extensive parameter fine-tuning and performance analysis.

2.1.1. PSEUDO DISTRIBUTED MODE

When I first started my journey with Hadoop & Big Data Analytics, I initially set up a cluster for pseudo-distributed mode (it means running Hadoop on one machine) on my machine with one name node and one data node. Following [6] and after a troublesome initial set-up and extensive configuration experimenting, I managed to launch the local HDFS cluster (figure 9). I used this environment for development, unit, and integration testing, leaving performance testing and parameter tuning for the RHUL Big Data Cluster.

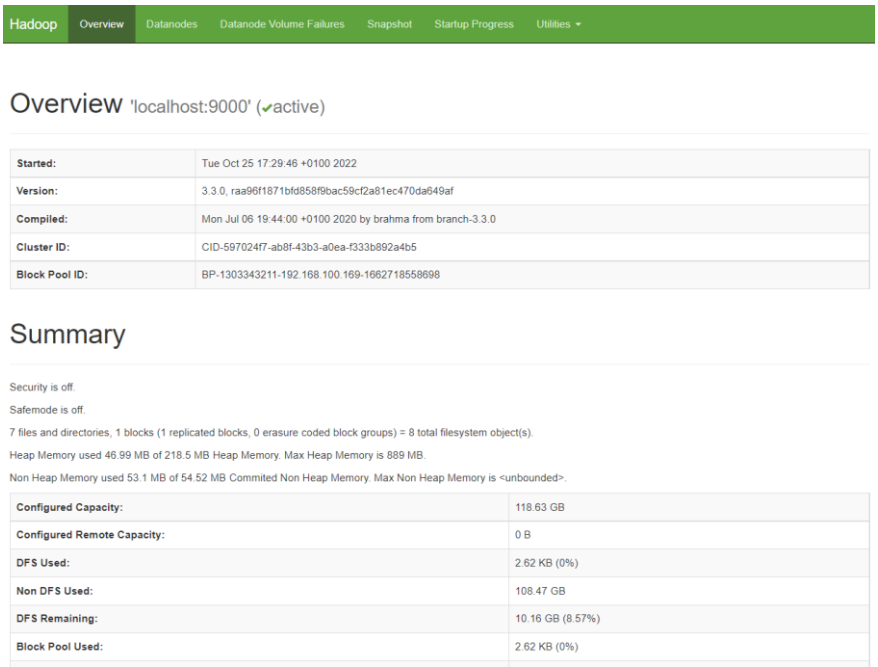


Figure 22 Pseudo-distributed mode configuration

2.1.2. RHUL BIG DATA CLUSTER

The cluster has a 1 master – 10 workers' configuration with 24GB of RAM per node. Because of its high specs, this is the environment I primarily used for integration and performance testing as well as experimenting with different parameters. Furthermore, the configuration of my programs is mostly fine-tuned for this cluster. After some initial issues with Hadoop running by jobs in local mode instead of using yarn, using the cluster was very straightforward. Here is a sample execution screenshot for the word grep problem execution using all 10 nodes (figure 23) and a history of jobs submitted by me (zhac254) (figure 25).

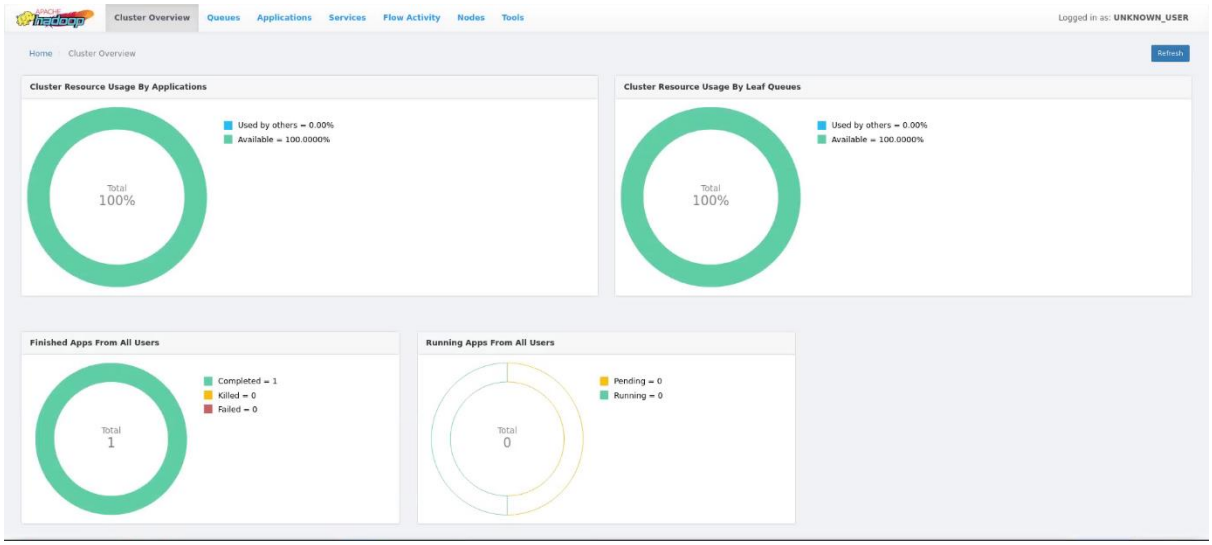


Figure 23 RHUL Big Data cluster overview

Node Label	Rack	Node State	Node Address	Node HTTP Address	Containers	Mem Used	Mem Available	VCores Used
default	/default-rack	RUNNING	hadoop-slave0.cim.rhul.ac.uk:45454	hadoop-slave0.cim.rhul.ac.u...	2	6 GB	18 GB	2
default	/default-rack	RUNNING	hadoop-slave5.cim.rhul.ac.uk:45454	hadoop-slave5.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave7.cim.rhul.ac.uk:45454	hadoop-slave7.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave3.cim.rhul.ac.uk:45454	hadoop-slave3.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave1.cim.rhul.ac.uk:45454	hadoop-slave1.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave6.cim.rhul.ac.uk:45454	hadoop-slave6.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave8.cim.rhul.ac.uk:45454	hadoop-slave8.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave9.cim.rhul.ac.uk:45454	hadoop-slave9.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave4.cim.rhul.ac.uk:45454	hadoop-slave4.cim.rhul.ac.u...	1	3 GB	21 GB	1
default	/default-rack	RUNNING	hadoop-slave2.cim.rhul.ac.uk:45454	hadoop-slave2.cim.rhul.ac.u...	1	3 GB	21 GB	1

Figure 24 Nodes usage

Application ID	Application Ty...	Application N...	User	State	Queue	Progress	Start Time	Elapsed Time	Finished Time
application_1668496677807_0001	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/15 15:...	50s 247ms	2022/11/15 15:...
application_1667892014261_0012	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/15 00:...	1m 19s 264ms	2022/11/15 00:...
application_1667892014261_0011	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/15 00:...	53s 360ms	2022/11/15 00:...
application_1667892014261_0010	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/14 23:...	58s 916ms	2022/11/15 00:...
application_1667892014261_0009	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/14 23:...	52s 842ms	2022/11/14 23:...
application_1667892014261_0008	MAPREDUCE	Final Year Proje...	zhac254	Finished	default	100%	2022/11/14 23:...	11s 825ms	2022/11/14 23:...

Figure 25 Cluster Job Runs History

2.2. PART ONE

This is the first part of the project which was developed in term one.

In the following section, I am going to present the two problems I have implemented: Word Count and Word Grep. Both problems were deployed and tested in distributed mode on a real cluster (Royal Holloway Big Data cluster). For each problem I will go over the following aspects:

- The dataset used for the problems
- Hadoop parameters that were used for both problems
- For each problem:
    - I will discuss its general overview
    - How I implemented it
    - Sample job execution
    - Experiments performed
    - Performance analysis
    - Proposed optimal Hadoop parameters

2.2.1. DATASET

For the implementation of both problems, I used Common Crawl [7], a large repository of scrapped web data. Common Crawl normally does several crawls per year, growing steadily with 200-300 terabytes per month since 2013 [7]. Since a crawl can be dozens of terabytes in size, Common crawl pre-processes the data [7]. The data is split into partitions of exactly 120MB. warc files (standard file format for crawl data) which are afterwards archived using gzip (.gz). This is highly beneficial since I could choose as much data as I wanted, and this partition size exactly matches the block size in the HDFS.

This means that each partition is exactly of the optimal size to be stored and processed on the HDFS as a single file. This means one file will exactly fit a single node in the HDFS cluster, avoiding unnecessary and time-consuming splits. In my case, I used fifteen partitions from the September 2022 crawl summing up to **1.8GB of archived data or around 6GB of uncompressed raw data (575.651 websites).**

```
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00000.warc.wet.gz
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00001.warc.wet.gz
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00002.warc.wet.gz
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00003.warc.wet.gz
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00004.warc.wet.gz
crawl-data/CC-MAIN-2022-27/segments/1656103033816.0/wet/CC-MAIN-20220624213908-20220625003908-00005.warc.wet.gz
```

Figure 26 Common Crawl partitions

2.2.2. HADOOP PARAMETERS

Parameter tuning is an important step in developing a MapReduce application with Hadoop as it can greatly influence the performance of the program. In the following table, there are described the most important parameters that were used for the two problems. Later, for each problem, for these parameters, I will offer my proposal for optimal values.

Parameter	Parameter Name	Description
Number of replicas	dfs.replication	The number of times the files are replicated across the HDFS (default is 3)
Block size	dfs.block.size	The block size in the HDFS (default is 120MB)
Maximum split size	mapred.max.split.size	The maximum size where files are split (this normally should coincide with the block size).
Minimum split size	mapred.min.split.size	The minimum size at which files are split
MapReduce framework name	mapred.framework.name	The name of the framework that handles the MR job.
The number of reduce tasks	mapred.reduce.tasks	The number of reduce tasks.
Name of the default file system	fs.defaultFs	The name of the default file system used by Hadoop

2.2.3. WORD COUNT PROBLEM

2.2.3.1. OVERVIEW

Problem description: given many web pages, count the number of occurrences of each word. This is an easy but ideal problem for Hadoop and MapReduce because it is intuitive to abstract this problem into a Map and Reduce implementation. The base logic to implement this in Hadoop is the following:

1. In the map phase, input files are split into words and the mapper maps each word to the value of one
2. In the combiner phase, all the same words are grouped
3. In the reducer phase, count the number of times a word has been mapped to one
4. Output: list of unique words and their frequency

2.2.3.2. IMPLEMENTATION

In the development of the Word Count problem, I iteratively developed two versions of the implementation:

- Word Count version 1 (V1): a simple implementation that only works with raw text files.
- Word Count version 2 (V2): an advanced implementation that is designed to work with archived warc files (for the Common Crawl repository).

2.2.3.3. WORD COUNT V1

In the following section, I am going to describe my implementation for word count v1.

The Mapper

In figure 27, WordMapper is a Java mapper implementation for the word count problem. Firstly, recall figure 3 of the map and reduce format. The WordMapper extends the Mapper interface (described previously in section 1.5.), defining its K1, K2, V1, and V2 types:

- K1 is an object since Hadoop internally assigns the input a key and we are not concerned about its type.
- V1 is of type Text which is a Hadoop primitive representing a String (since this program is processing simple raw text).
- K2 is of type Text since we are mapping individual words.
- V2 is of type IntWritable which is a Hadoop primitive representing an Integer (since the words will be mapped to their frequencies.

The map method receives a key and a value (K1 & V1) and a Context object which is a Hadoop primitive used to generate and store key-value pairs and is shared between the Map & Reducer. The map method uses a StringTokenizer to split the input into individual words. The program then iterates over the words and maps each word found to the value one using the context. For example, if the word football was found five times, the map output will be football: [1, 1, 1, 1, 1].

```java
/**
 * Implementation of the Mapper for the Word Count Problem.
 */
4 usages  Sirbu Cosmin (2019) ZHAC254
public class WordMapper extends Mapper<Object, Text, Text, IntWritable> {

    /**
     * Receives a partition of the input data, parses it and maps each new word to the value one.
     * Later the combiner will combine the found frequencies of the found words.
     *
     * @param key     the initial system generated key K1 of the input
     * @param value   the initial unparsed value V1 of the input
     * @param context the context used to map key-value pairs
     * @throws IOException          when the input is wrong
     * @throws InterruptedException when the context writing is interrupted
     */
    Sirbu Cosmin (2019) ZHAC254
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer parser = new StringTokenizer(value.toString());
        WordParser wordParser = new WordParser();
        while (parser.hasMoreTokens()) {
            wordParser.parse(parser.nextToken());
            context.write(wordParser.getWord(), wordParser.getOne());
        }
    }
}
```

Figure 27 Word Count V1 Mapper

The Reducer

In figure 28, WordReducer is a Java reducer implementation for the word count problem. The WordReducer extends the Reducer interface (described previously in section 1.5.), defining its K2, V2, K3, and V3 types:

- K2 is an individual word written previously by the mapper
- V2 is the list of word appearances
- K3 is of type Text since the final key will be the individual word
- V3 is of type Text since the found value will be the word's frequency

The reduce method receives K2 & V2 and a context object. Using the previous example, the key is the word football and the Iterable is [1, 1, 1, 1, 1]. The program loops on the frequencies Iterable and calculates the sum which is the final frequency of the word. Finally, using the context the program writes the key-value pair to the context. In our case, this is the final word – frequency (I.e., football - 5).

```java
/**
 * Implementation of the Reducer for the Word Count Problem.
 */
5 usages  Sirbu Cosmin (2019) ZHAC254
public class WordReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    /**
     * Variable that holds the final result of the word frequency analysis addition.
     */
    2 usages
    private final IntWritable result = new IntWritable();

    /**
     * Receives a partition of the mapped input data holding a word and a list of mapped frequencies,
     * adds them up and writes the reduced result to the context.
     *
     * @param key     the mapped word
     * @param values  the list of found word frequency values
     * @param context the context used to reduce the mapped values
     * @throws IOException          when the input is wrong
     * @throws InterruptedException when the context writing is interrupted
     */
    Sirbu Cosmin (2019) ZHAC254
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Figure 28 Word Count V1 reducer

2.2.3.4. WORD COUNT V2

In the following section, I am going to describe my implementation for Word Count V2.

Warc Input Format

Word Count V2 uses the Common Crawl repository and its archived .warc files. To be able to parse archived .warc files, I had to create a custom File Input Format class that un-archives the Common Crawl archived partitions to be easily used by the mapper. To do this, I followed and adapted Common Crawl's example [8].

My custom WARC File Input Format extends the Hadoop default's File Input Format but overrides its createRecordReader() method to return a new instance of my custom record reader: WarcFileRecordReader.

WarcFileRecordReader

WarcFileRecordReader is used to read the archived records and extends Hadoop's default Record Reader. It overrides the initialize() method to use the WARCReaderFactory() primitive with the input path and fileInputSteam as parameters to create the archive reader. This initialised archive reader will be later used by the mapper to read and parse website data at a time and map the individual words.

```
@Override
public void initialize(InputSplit inputSplit, TaskAttemptContext context) throws IOException {
  FileSplit split = (FileSplit) inputSplit;
  Configuration conf = context.getConfiguration();
  Path path = split.getPath();
  FileSystem fs = path.getFileSystem(conf);
  fileInputStream = fs.open(path);
  archiveReaderPath = path.getName();
  archiveReader = WARCReaderFactory.get(path.getName(), fileInputStream, atFirstRecord: true);
}
```

Figure 29 Word Count V2 file reader

The Mapper

Since I used the Warc Input Format for the Word Grep problem too and the archived file reading mechanism is the same, I created a base superclass called WarcMap which is extended by specialised classes for word count v2 and word grep. For Word Count v2 the specialised class is **WordCountV2MapFacade** which extends **FileReader** and implements its version of some methods. Here I applied the façade pattern to hide the complexity of the map method

```
  @Override
  public void map(Text key, ArchiveReader archiveReader, Context context) {
    WordCountV2MapFacade wordCountV2MapFacade = new WordCountV2MapFacade(archiveReader, context);
    wordCountV2MapFacade.mapRecord();
  }
}
```

Figure 30 Word Count V2 mapper

loops through all the archived records, parses the archived data, and maps the words. Some other methods for exception catching or metrics updating are also present but the main two methods that perform the business logic are: **getInputData() and processInputData()**

getInputData() converts the archived record into a byte array, then converts it into a String and finally to a String Tokenizer. The String Tokenizer has each word in the String as a token which can be read and processed.

```
private void getInputData(ArchiveRecord record) throws IOException {
  byte[] rawData = IOUtils.toByteArray(record, record.available());
  String input = new String(rawData);
  tokenizer = new StringTokenizer(input);
}
```

Figure 31 Word Count V2 File Reader getInputData()

processInputData() reads each word, parses it and maps it to the value one using the Context.

```
private void processInputData() throws IOException, InterruptedException {
  if (!tokenizer.hasMoreTokens()) {
    updateEmptyPagesNo();
  } else {
    while (tokenizer.hasMoreTokens()) {
      wordParser.parse(tokenizer.nextToken());
      if (wordParser.isValidWord()) {
        context.write(wordParser.getWord(), wordParser.getOne());
      }
    }
  }
}
```

Figure 32 processInputData

Reducer

The word count v2 reducer is very similar to the Word Count V1 one.

2.2.3.5. DEPLOYMENT

As input, the directory of the archived .warc files (figure 26) is passed to the Word Count V2 program. The contents of each archived .warc file have the format seen in figure 33 (raw website text). After executing the program, the output will be a list of words and their frequencies. For the output, only alphabetic characters from any language are considered (figure 34). For full results check the output/WordCountV1.txt and output/WordCountV2.txt directories of my project.

Sample Input

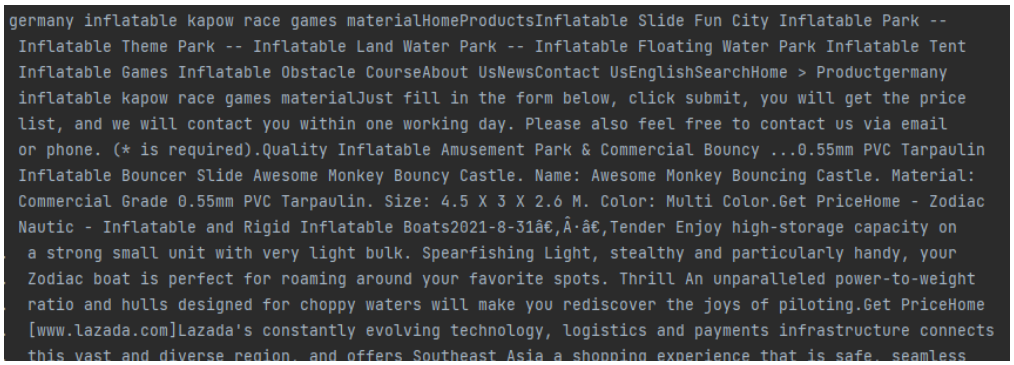


Figure 33 Sample Word Count V2 input

Sample Output

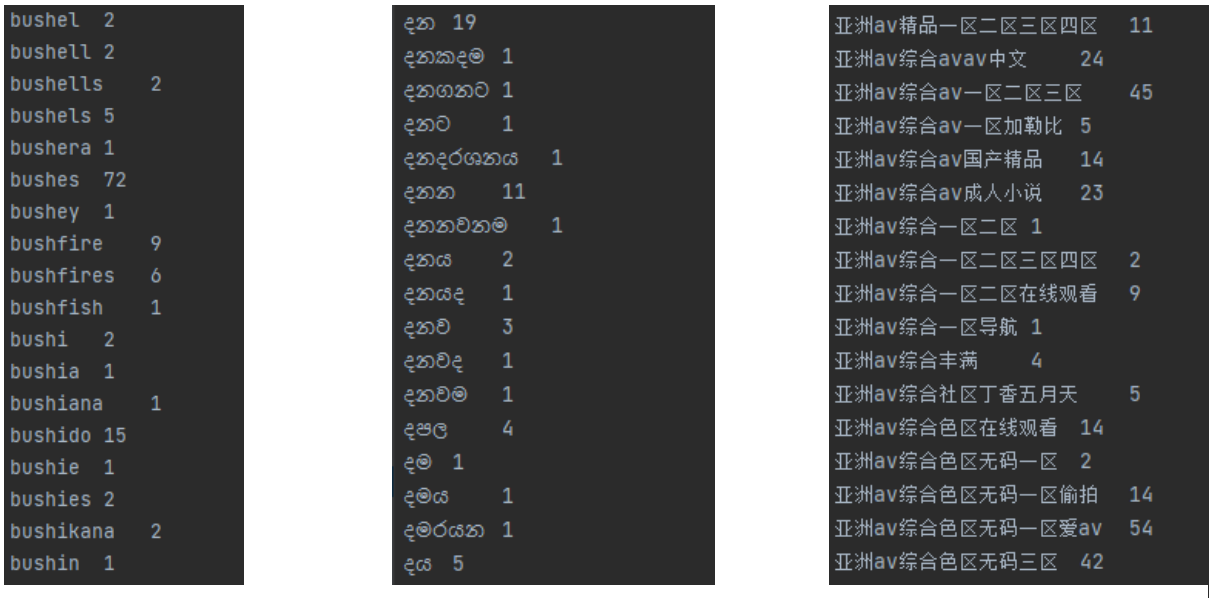


Figure 34 Word Count V2 Output

2.2.3.6. EXPERIMENTS

For the Word Count V2 problem, I performed a series of experiments with different parameters as well as other code modifications to achieve the best performance.

EXPERIMENT 1: COMBINER

Description

I ran the Word Count V2 job with and without a combiner.

Goal

I wanted to see the execution time difference between the two approaches.

Explanation

For small input sizes, a custom combiner is not needed since the data outputted by the mappers is not that big. Therefore, even though the reducer receives more records to parse, overall run time is not greatly affected. That however is not the case with larger data sizes. I ran the Word Count V2 job on pseudo-distributed mode with and without a combiner for one 120MB Common Crawl file. For the combiner, I used my reducer class.

Results

The difference in run time was more than a minute. After investigating the MR job logs, the reason for the big difference was obvious. If I run the program without a combiner, then the reducer has as input 32.151.671 records. If I run it with a combiner, then the reducer has input 2.261.872 records (16x fewer records).

Conclusion

By just adding this intermediate combining step after each map job, we take a large load off the reducer which on larger jobs can result in significant performance gains.

EXPERIMENT 2: HADOOP PRIMITIVE CREATION

Description

I ran the Word Count V2 job with and without proper Hadoop primitive creation

Goal

I wanted to see the execution time difference between the two approaches.

Explanation

Hadoop uses its version of regular primitives (IntWritable, Text, etc.) as a replacement for Int, String and so on. This is because of Hadoop's internal architecture structure and especially for efficiency's sake. However, not treating primitive creation carefully may result in bad performance. This is because, for large data sizes, object initialization is memory costly. As an experiment, for one 120MB file input, for multiple primitives alongside my MR job, instead of initializing it once and later calling its set() method to update their value, I created a new primitive each time.

Results

Not treating primitives correctly has resulted in a 20 seconds increase in execution speed.

Conclusion

This relatively big increase lies solely on poor memory usage and allocation, an important concept in Big Data Analytics.

2.2.3.7. PERFORMANCE ANALYSIS

Description

On the RHUL Big Data Cluster, I performed 15 performance tests for Word Count V2 for inputs in the range of 1 Common Crawl partition to 15 (from 120MB to 1.8GB of archived data). The results can be seen in figure 35.

Goal

Compare and analyse the performance of the Word Count V2 job in terms of execution time, data size and cluster nodes usage to draw some useful conclusions.

Explanation

As can be seen in the graph, the job execution time can be split into two parts:

- **120MB - 1200MB input data size**
  There is a gentle linear increase of execution time until 1200MB. This corresponds to the 10 nodes usage threshold (recall section 2.1.2 RHUL Big Data Cluster). Each node in the cluster has a block size of 120MB and each of my files are 120MB in size. So, until the 1200MB data size, each node processes only one file at a time and thus any unnecessary and resource-consuming extra splits are avoided. Therefore, the map jobs are efficiently parallelised amongst the 10 nodes.

- **1200MB - 1800MB input data size**
  Beyond the 1200MB threshold, there will be multiple files to be processed per node, so the specs of the cluster are not used to their maximum capacity. Therefore, all 10 nodes are used, and multiple maps are executed per node.

Results

- **120MB - 1200MB**
  In this range (x10 size increase), there is **only a 31 seconds increase in execution time**. The increase in execution time is due to the larger number of combiner jobs and a larger amount of data fed into the reducer.

- **1200MB - 1800MB**
  In this range (x1.5 size increase), **there is a 32 seconds increase in execution time!** This is more than the previous 31 seconds increase despite a much smaller data size difference.

Conclusion

In this example, we have seen the importance of both adequate parameter tuning and the actual size and technical specifications of the cluster in the Hadoop job performance. This tells us two lessons:

- Hadoop is ideal for batch processing and easily scalable nodes wise as a good design with enough nodes will result in a good performance. Furthermore, scaling up will not require any major code or parameter changes.
- It is costly. For enterprise-level requirements where terabytes of data are processed regularly unless a large cluster is set up, the job execution time can quickly go out of hand. This is because, unless a minimum number of maps (ideally one) are done per node, then the parallelization advantages of Hadoop are not efficiently used.

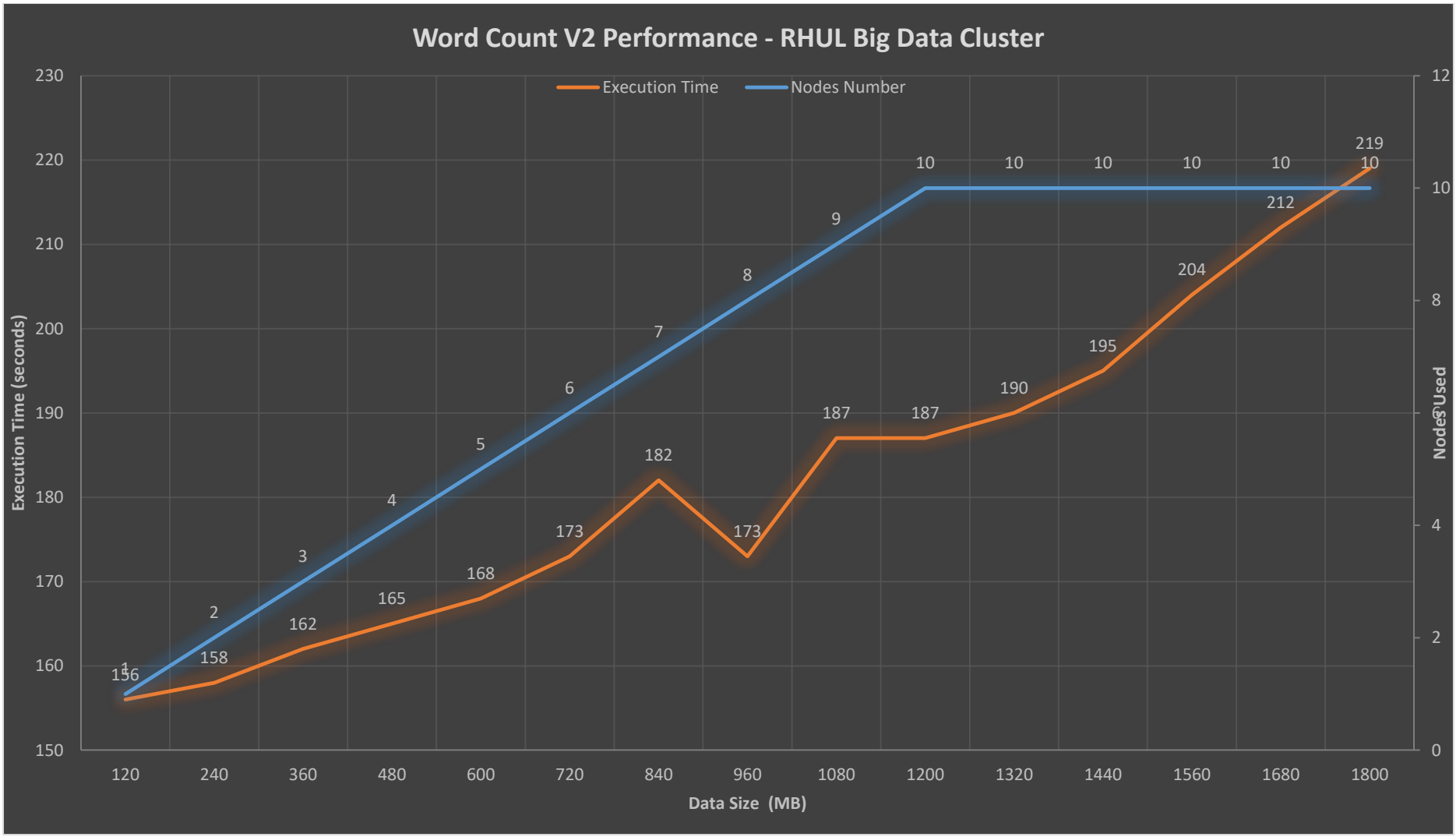


Figure 35 Word Count V2 Performance – RHUL Big Data Cluster

2.2.3.8. OPTIMISED HADOOP PARAMETERS

In the following table, there are my proposed parameters for an optimal Word Count job execution using Hadoop. Please recall section 2.2.2. Hadoop Parameters to understand these settings better.

Parameter	Parameter Value	Explanation
Number of replicas	3	This is the default value which suits the problem's needs as well as the RHUL cluster configuration.
Block size	120MB	Since the common crawl file partitions were 120MB, the ideal size for the HDFS block is 120MB as well.
Maximum split size	120MB	I wanted input slits the same as the HDFS blocks so 120MB is ideal
Minimum split size	1byte	I kept this as the minimum possible split for word count, we are not concerned with this too much.
MapReduce framework name	yarn	The best performance can only be obtained using yarn, not the local file system.
Number of reduce tasks	1	For this problem, a single reducer was enough. However, I also used a Combiner to minimize the amount of data fed into the reducer.
Name of default file system	RHUL Cluster URL	Since the best performance was obtained using the RHUL Custer, this setting will be the RHUL Cluster URL.

2.2.4. DISTRIBUTED WORD GREP PROBLEM

2.2.4.1. OVERVIEW

Problem description: given many web pages and a sample regex expression, output a list of all the websites with the line and offset position where the regex expression is found. More complex than the word count problem, this problem can be extracted into a Map and Reduce implementation in the following way:

1. In the map phase, clean the data and map each website to its parsed data.
2. In the reducer phase, look for the expression in the website data and get its positions.
3. Output the website alongside the website data formatted into lines and the positions where the expression is found.

2.2.4.2. IMPLEMENTATION

For the Word Grep Problem solution, I directly implemented it for the Common Crawl repo (like Word Count V2). For the Input Format class, the same is used for the Word Count V2 (WarcFileInputFormat).

The Mapper

```
public void map(Text key, ArchiveReader archiveReader, Context context) {
  WordGrepMapFacade grepMapperFileReader = new WordGrepMapFacade(archiveReader, context);
  grepMapperFileReader.mapRecord();
}
```

Figure 36 Word Grep mapper

As for Word Count V2, I created a Façade, WordGrepMapFacade that implements the WarcMap superclass. WordGrepMapFacade works in the following way. After its instantiation, its parseRecord() method is called which loops through all the archived records, parses the archived data, and maps the words. Some other methods for exception catching or metrics updating are also present but the main two methods that perform the business logic are getInputData() and processInputData().

```
private void getInputData(ArchiveRecord record) throws IOException {
  byte[] rawData = IOUtils.toByteArray(record, record.available());
  websiteData = new String(rawData).replaceAll( regex: "[\\r\\n]+",  replacement: "");
  mappedWebsite.set(websiteData);
}
```

Figure 37 Word Grep File Reader getInputData()

getInputData() takes in an archived record and replaces all its end-of-line characters for easier parsing in the reducer.

processInputData() takes the website URL from the record header and maps the parsed website data to the website URL.

```
private void processInputData(ArchiveRecord record) throws IOException, InterruptedException {
  String crawledWebsite = record.getHeader().getUrl();
  website.set(crawledWebsite);

  Configuration configuration = context.getConfiguration();
  String searchRegex = configuration.get("searchRegex");

  if (websiteData.length() == 0) {
    updateEmptyPagesNo();
  } else {
    if (websiteData.contains(searchRegex)) {
      context.write(website, mappedWebsite);
    }
  }
}
```

Figure 38 Word Grep File Reader processInputData()

The Reducer

Logic-wise the reducer works in the following way:

1. Splits the website data into lines of a maximum of 100 characters while also considering the end of line words so the end of line words is not split if the line exceeds 100 characters.
2. Gets the lines where regex expression is found
3. Gets the positions within the lines where the regex expression is found
4. Formats the output

2.2.4.3. DEPLOYMENT

As input, the directory of the archived .warc files (figure 26) is passed to the Word Grep program. The contents of each archived .warc file have the format seen in figure 39 (raw website text). After executing the program, the output will be a list of websites with the lines and offset position from the begging of the line where the regex expression is found (figure 40). For full results please check the output/WordGrep.txt directory of my project.

Sample Input

```
Format Photo Sur Cv - All New Resume Examples & Resume TemplateResume DatabaseHomeformat photo sur cv<!-
 Pasang Iklan Disini -->format photo sur cv PDF / ePUB Bookformat photo sur cvartificialgor.herokuapp.com
 9 out of 10 based on 300 ratings. 300 user reviews.Recent Updatebon cv anglais , setone cvs , cv vierge
 a remplir gratuit , cv informatique savoir etre , modele cv du social , cv original gratuit sans photo
 , cv pour stage bts mecp , exemple de cv pour emploi de pisciniste , cv chef atelier , template cv noir
 et jaune , deposer cv et lettre de motivation air france recrutement , modele word mini cv , cv fitness
 , reussir en sambre logiciel cv , cv modele quebecois , gilbert bougi cv , bon niveau anglais cv , competences
 dans le cv administration , cv pdg yahoo , centres d'interet cv cinema blockbuster , modele cv spatieux
 , cv exemple simple telecharger , modele simple de cv , telecharger openoffice gratuit pour fire cv
  , exemple cv jeunesse , template cv communication digital , cv modele , exemple cv cip gratuit , modele
  cv en anglais accounting , free download model cv , cv a telecharger recherche de travail , cv objectif
  professionnel exemple , exemple cv services a la personne , comment faire mon cv pour une reconversation
  , modele cv colonne , modele cv vente original , cv pour assistant pedegogique , meilleur application
  cv gratuit , cv original job ete , lettre de motivation et cv quel police utiliser , exemple cv manoeuvre
  batiment debutant pole emploi , changer un cv sur espace pole emploi , comment integrer ses experiences
  de benevolat dans son cv , modele cv absence pour annee sabbatique , en quel caractere on ecrit un cv
  en anglais , american beauties fotobildband u 4 musik cds earbookla boite a outils de la strategieromische
  geschichte von den anfangen bis zum untergang life lines holding on and letting gola memoire entre neurosciences
```

Figure 39 Word Grep Sample Input

Sample Output: searching for the word "football"

```
http://cien-byarhair.com/archives/4375?replytocom=8979308
football found at positions:
Line 1350 - positions: 80
Line 1351 - positions: 98
Line 1352 - positions: 48
Line 1354 - positions: 33
```

```
1350.  over time.บอลวันที่2021年 11月 01日返信引用Before you start attempting to utilize any football betting tips,
1351.  youneed to ensure that you have chosen the appropriate bookmaker.If you're trying to find mobile football
1352.  betting zambia photos datalinked to the mobile football betting zambia matter, you may have pay a visit
1353.  to the suitable weblog.Keep in mind though that I have additionally put collectively somedefinitive
1354.  guides to each of the following football coupons,and as such if any of those that are listed below do
```

Figure 40 Word Grep output searching for "football"

2.2.4.4. EXPERIMENTS

For the Word Grep problem, I performed a series of experiments with different parameters as well as other code modifications to achieve the best performance.

EXPERIMENT 1: COMBINER

Description

I ran the Word Grep problem job with the reducer as the combiner.

Goal

See if the Word Count V2 Combiner experiment can be replicated.

Explanation

Same explanation as for the Word Count V2 Combiner Experiment.

Results

When I used the reducer as a combiner, the output had multiple websites data combined and the expression was "found" in non-existing places. I quickly realised the reason. The program had this weird behaviour because the combiner was receiving the output results of multiple mappers at the same time and since the reducer is designed to parse only one website data at a time, it ended up combining partitions of the same website and then feeding those as a sole entry to the reducer.

Conclusion

For the word grep problem, I ended up removing the combiner as ultimately it is not needed. Since the expression is looked for in one website at a time, and the mapper keys are the website URLs then the combiner will only have one entry to combine, rendering the combiner useless. This tells us an important story about assumptions. What works well for a problem might not work well for another.

EXPERIMENT 2: FREQUENT WORDS

Description

I ran the Word Grep job in pseudo-distributed mode with the most common word in the English lexicon, "the" and with a less common word, “football”.

Goal

See how my program behaves with different words of different frequencies.

Explanation

Depending on the expression frequency, the execution time can vary as well. If a very common expression is searched for then I assumed that the execution time will go up because the expression is found more time times.

Results

The run time for the job was 80 seconds whereas the run time for searching the word "football" is just 20 seconds.

Conclusion

The word grep problem, due to its very nature, is less scalable for frequent words. Furthermore, implementation-wise there doesn’t exist a way to greatly improve the performance for searching for frequent words. This is because the input files which correspond to a single website data can’t be split up so the expression searching can’t be parallelised. As a lesson, for some problems in Big Data, we should pay great attention to edge cases.

2.2.4.5. PERFORMANCE ANALYSIS

Description

On the RHUL Big Data Cluster, I performed 15 performance tests to check the run time of the Word Grep problem while searching for the word "football" for inputs in the range of 1 Common Crawl partition to 15 (120MB – 1.8GB of archived data). The results can be seen in figure 41.

Goal

Compare and analyse the performance of the Word Grep job in terms of execution time, data size and cluster nodes usage to draw some useful conclusions.

Explanation

As can be seen in the graph, the general trend in the execution times follows the same pattern as in Word Count V2. The reason for this is intuitive, both problems are executed on the same cluster with the same architecture.

Results

- **120MB - 1200MB**
  In this range (**x10 size increase**), the execution time increases by **26 seconds**.

- **1200MB – 1800MB**
  In this range (**x1.5 size increase**) the execution time increases **by 20 seconds**. As expected, after the 10 nodes usage threshold, the execution time increases more rapidly. However, the performance slope is not as steep as for Word Count V2.

Conclusion

The Word Grep scales better overall but worse in edge cases than my implementation for Word Count.

On one side, the general performance is better because, for Word Count V2, each word must be processed, cleaned, and mapped, a resource-consuming process. Whereas for Word Grep, at the map phase, whole website rows are processed while searching for the regex expression. As a result, in case the regex is not found, then the regex searching fails early resulting in a faster execution time.

On the other side, as we have seen in the second experiment, searching for frequent words can result in very bad performances.

In conclusion, for some problems in Big Data analytics, we must be very careful with special cases. Distributed word grep is such a problem. When handling large quantities of data, it is easy to get lost in it. Even though at a glance an implementation seems efficient, only after carefully analysing edge cases we can draw an informed conclusion.

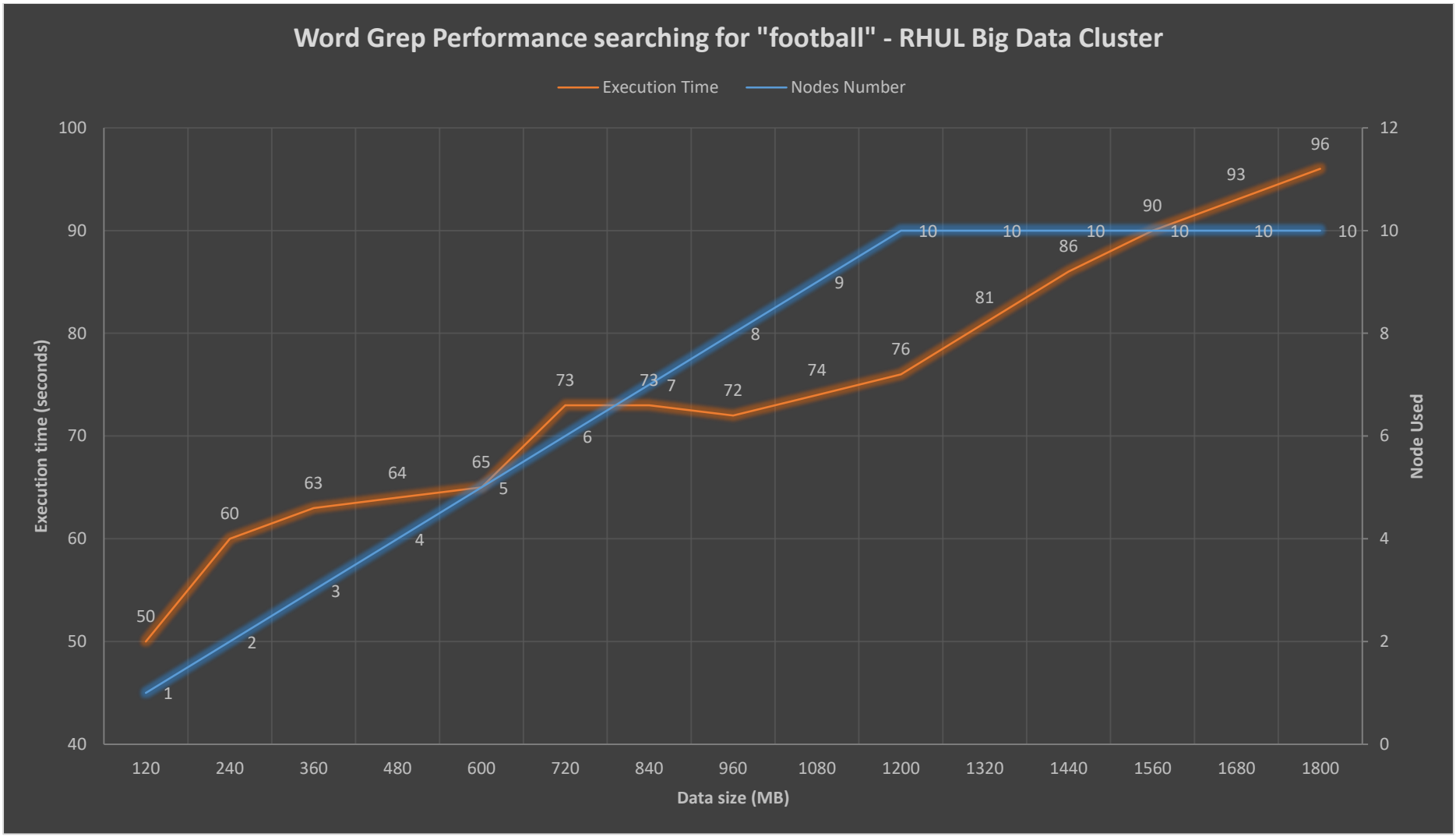


Figure 41 Word Grep Performance searching for "football"– RHUL Big Data Cluster

2.2.4.6. OPTIMISED HADOOP PARAMETERS

In the following table, there are my proposed parameters for an optimal Word Grep job execution using Hadoop. Please recall section 2.2.2. Hadoop Parameters to understand these settings better.

Parameter	Parameter Value	Explanation
Number of replicas	3	This is the default value which suits the problem's needs as well as the RHUL cluster configuration.
Block size	120MB	Since the common crawl file partitions were 120MB, the ideal size for the HDFS block is 120MB as well.
Maximum split size	120MB	I wanted input slits the same as the HDFS blocks so 120MB is ideal
Minimum split size	1byte	I kept this as the minimum possible split for word count, we are not concerned with this too much.
MapReduce framework name	yarn	The best performance can only be obtained using yarn, not the local file system.
Number of reduce tasks	1	For this problem, a single reducer was enough. However, I also used a Combiner to minimize the amount of data fed into the reducer.
Name of default file system	RHUL Cluster URL	Since the best performance was obtained using the RHUL Custer, this setting will be the RHUL Cluster URL

2.2.5. DISCUSSION

In this first part of the project, we have seen the development, deployment, experimentation and performance analysis of two Big Data problems. Let's take a moment and have a discussion about the lessons learned from this Big Data pipeline.

We have seen the power of parallel processing with Hadoop but also some of its downsides. Hadoop is easily scalable and ideal for batch processing but to maintain its good performance, a large cluster with numerous machines is needed. For enterprise-level solutions, this is costly. Furthermore, the MapReduce paradigm in which you are required to fit your Big Data programs is restrictive. This is because transforming a problem into the MapReduce format is not always straightforward and actually might be counter-intuitive and less efficient than using other formats (which Spark lets you have).

Another important aspect we saw is the importance of carefully analysing edge cases for Big Data problems. This is because some problems have highly variable performance depending on the job parameters. Therefore, edge cases must be thoroughly analysed so that the programs maintain their good performance regardless of the data size.

The difficult aspect of this first part was putting all the learned theory into practice to develop and deploying my Big Data programs on the Royal Holloway Cluster. After I started to get more experience and confidence, the crucial but important issue to solve was to analyse the performance of my programs and fine-tune them.

2.3. PART TWO

In the following part, I am going to describe the second part of my project which was developed in term two.

Have you ever wondered who was the most important person in history? Napoleon or Einstein? Da Vinci or Aristotle? The historical significance program will aim to answer this by calculating the historical significance of individuals on the entire Wikipedia.

Using the entire Wikipedia data source, Spark, and Natural language processing, my program attributes a score to every person with a Wikipedia article. The output is a sorted list of the most significant personalities in history. Because of the sheer amount of popularity and influence they have, I expect to see in the top figures like Einstein, Jesus Christ, or Aristotle. However, these are the expected results. What would be the most interesting to see is where lesser-known but highly important individuals are positioned within this list (individuals like Muhammad Al-Khwarizmi, the forefather of algorithms or Cleisthenes, the father of democracy) as well as how the program places individuals across different domains of activity (music, art, science, etc.)

Historical significance is an abstract task and thus hard to quantify using an algorithm. My idea is based on [12] but with some modifications. Nevertheless, to calculate the historical significance score I have in mind two aspects: popularity and domain of activity importance. The reason for this is that popularity alone is not a true indicator of one's importance to history. Their domain of activity is crucial in our case because if we are looking for popularity alone then a modern-day movie star and or singer will end up being more important than someone like Plato.

Therefore, my program will be split into three parts: popularity calculation, domain of activity importance and the final historical significance program. The process is illustrated in the image below.

For the popularity metric calculation, I do the following:

- Parse the entire English Wikipedia to get each website's internal connections.
- Construct a Graph of all internal connections.
- Calculate the Page Rank of all Wikipedia pages.

For the domain of activity importance, I do the following:

- Construct a dataset with texts labelled with their domain: (sports, music, science, etc.).
- Train a Natural Language Processing (NLP) model on the found dataset.
- Using the trained model, I get the domain of activity for each of the individuals' Wikipedia pages.
- Assign a weight to each domain of activity regarding their historical significance. For example, the domain "science" will receive a better weight than the domain "sport".

For the historical significance program, I do the following:

- Combine the results of the two programs to calculate each person's score.
- The output is a sorted list of the most important personalities in history.

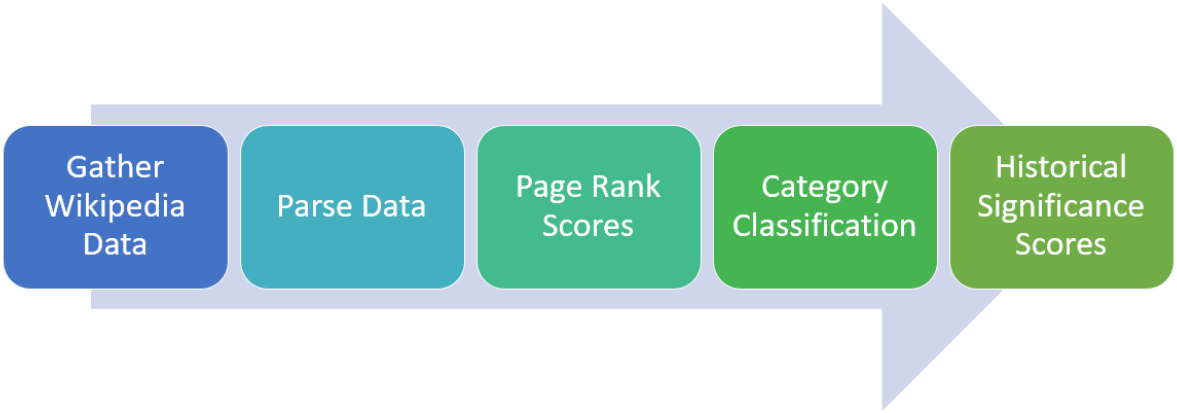


Figure 22 Part Two Program Pipeline

2.3.1. DATASET

For the implementation of the programs in part two, I used the latest English Wikipedia dump from May 2022. This version contains approximately **6.600.000 articles with 4.2 billion words**, having an average of 649 words per article. Out of these articles, **1.258.810 articles are biographic**. The data can be downloaded from the Internet Archives [32] in the format of multiple smaller archived partition dumps or a single large archived file of 20GB in size. Unpacked, the entire English Wikipedia is **100GB in size** (without media, just plain text and mark-up).

2.3.2. SPARK PARAMETERS

Parameter tuning is an important step in developing a Spark application as it can greatly influence the performance of the program. In the following table, there are described the most important parameters that were used for the part two programs. Later, for each problem, for these parameters, I will offer my proposal for optimal values.

Parameter	Parameter Name	Description
Default Parallelism	spark.default.parallelism	The default number of partitions in RDDs returned by transformations like join, reduceByKey, and parallelize.
Spark Serializer	spark.serializer	The type of Serializer to be used by Spark. By default Spark is using the

		Java Serializer but can (and should) be changed to Kryo Serialization
Kryo Registration	spark.kryo.registrationRequired	Whether Kryo Serializer class registration is required or not. If set to yes, the user has to manually register all types of classes used during serialization to Kryo which can lead to slightly better performances.
Driver Memory	spark.driver.memory	The amount of memory (in GB) to be allocated to the driver node.
Executor Memory	spark.executor.memory	The amount of memory (in GB) to be allocated to executors nodes.
The number of executors	spark.executor.instances	The number of executors to be used by the spark job.
Executor Cores	spark.executor.cores	The number of cores to by each spark executor during the job.
The number of shuffle partitions	spark.sql.shuffle.partitions	The number of partitions to use during shuffling operations such as joins or aggregations. Default is 200 and can (and should) be changed according to the job requirements and the number of executors cores.
Extra archived files	spark.yarn.dist.archives	Extra archived files to be passed to each executor and the driver during the job execution. It is important especially during deploying Python apps which have extra dependencies.
Log Level	sparkContext.setLogLevel	The log level to be used during the job execution. Can be INFO, WARN and ERROR.
Jars Packages	spark.jars.packages	A list of comma separated values of Maven artefacts' to be downloaded on each executor and the driver node.

2.3.3. PAGE RANK PROBLEM

2.3.3.1. OVERVIEW

The purpose of this program is to take the entire English Wikipedia articles dataset and apply the Page Rank algorithm to it. The result will be a ranked list of all the biographical Wikipedia pages, from the most important to the least important. The approach is the following and also be seen illustrated below:

1. Gather Wikipedia Data
2. Understand Wikipedia XML format
3. Parse Wikipedia articles to get a list of articles and their connections
4. Create Graph
5. Run Page Rank until a convergence point

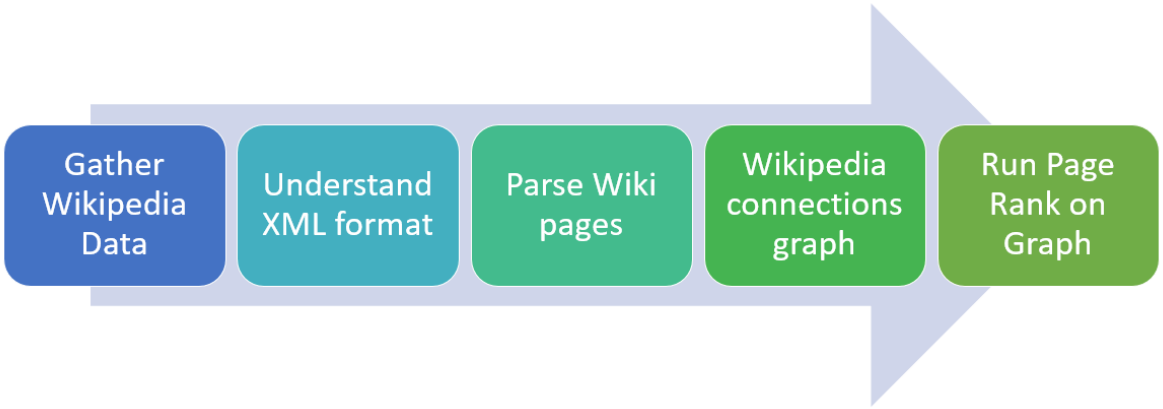


Figure 43 Wikipedia Page Rank Pipeline

2.3.3.2. IMPLEMENTATION

In the development of the Page Rank problem, I iteratively developed two versions of the implementation:

- **Naïve Page Rank**: from first principles implementation of page rank that runs for a fixed number of iterations.
- **Wiki Page Rank:** a complete Page Rank implementation that uses the Spark GraphX library and runs until a convergence point is reached.

2.3.3.3. NAÏVE PAGE RANK

In the following section, I am going to describe my approach to creating the page rank algorithm from scratch. In this version, I am not running the program until converge, but rather just for several iterations. This is why I called it naïve and it is the initial approach the Google Page Rank algorithm followed as well. This approach can give good results but at the same time it doesn't guarantee the most accurate page rank results and can waste computation on re-computing the page rank even through the convergence is already achieved.

Data Used

Unlike the Wiki Page Rank version, this program only works for a simple text file input of page connections. A sample file can be seen below. Each of the a-e values are a node in the graph and a line in the file represents a connection between them.

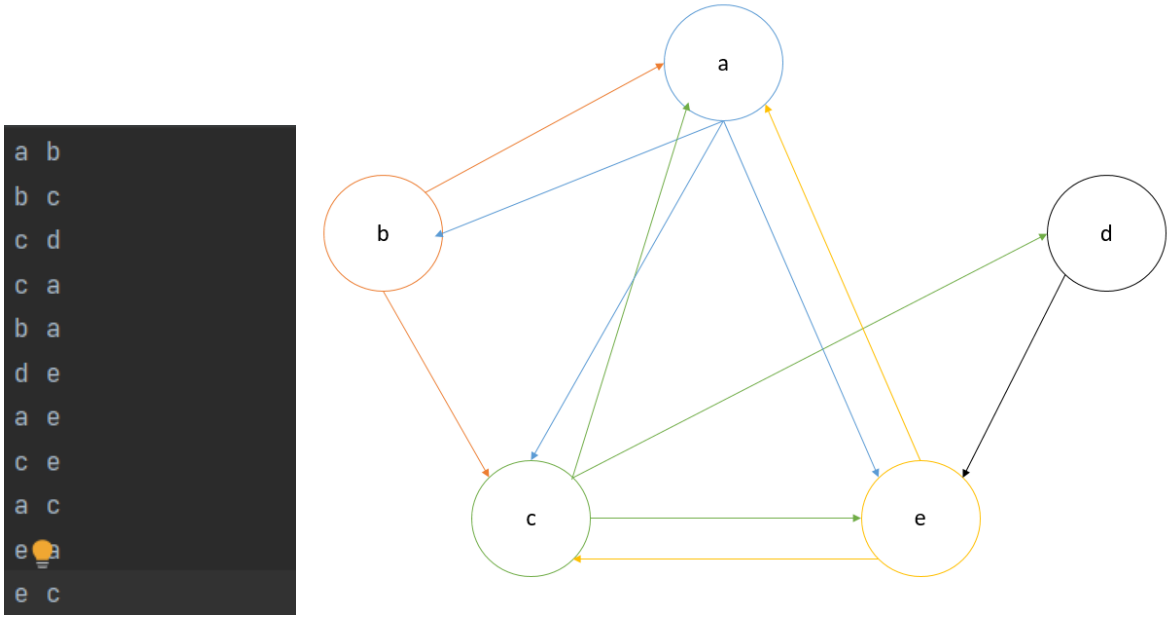


Figure 44 Naive Page Rank Input

Parsing the file

The first step is to parse in the vertex connections. This is done via the following method which reads the file line by line and saves the values in an RDD of unique tuples.

```
1 usage  Cosmin Cristian Sirbu
private void parseVertexConnections() {
  JavaRDD<String> lines = this.spark
      .read() DataFrameReader
      .textFile(this.inputPath) Dataset<String>
      .javaRDD();

  this.links = lines
      .mapToPair(NaivePageRank::parseLinksTuple) JavaPairRDD<String, String>
      .distinct()
      .groupByKey() JavaPairRDD<String, Iterable<String>>
      .persist(StorageLevel.MEMORY_ONLY_SER());
}
```

Figure 45 Parsing Data

Computing the Page Rank

Now that we have our data in a usable format, the next step is to run the page rank algorithm on the parsed input.

Firstly, the links RDD values are initialized to 1.

```
// Loads all URLs with other pages links to from input file and initialize ranks of them to one
this.ranks = this.links.mapValues(rs -> 1.0);
```

Figure 46 Vertex Links Values Initialization

Next, we calculate the page contributions of the current page to the ranks of other pages.

```
1 usage  new *
private static Iterator<Tuple2<String, Double>> pageToOtherPagesContribution(
    Tuple2<Iterable<String>, Double> currentPage) {
  int pageLinksCount = Iterables.size(currentPage._1());
  List<Tuple2<String, Double>> results = new ArrayList<>();
  for (String n : currentPage._1) {
    results.add(new Tuple2<>(n, currentPage._2() / pageLinksCount));
  }
  return results.iterator();
}
```

Figure 47 Current Page to Other Pages Contributions

To calculate the contribution of a page to its neighbours, I am using the following formula:

$$Rank = \frac{current\ page\ rank}{the\ length\ of\ the\ current\ page\ connections}$$

Then I calculate the contributions of other pages to the current page rank.

```
private void otherPagesToPageContributions(JavaPairRDD<String, Double> contributions) {
  this.ranks = contributions
      .reduceByKey(Double::sum)
      .mapValues(rank -> resetProbability + rank * (1 - resetProbability));
}
```

Figure 48 Other Pages to Current Page Contributions

The next step is to calculate the contributions of other pages to the current page which is done by the following formula (for a default reset probability of 0.15):

$$Rank = reset\ probability + rank \times (1 - reset\ probability)$$

Preliminary Results

As input, the directory formatted as above is passed to the Naïve Page Rank program. For the presented dataset, here is the computed page rank.

Vertex	Page Rank
d	0.521
d	1.335
a	1.310
b	0.521
b	1.310

Figure 49 Naive Page Rank Results

2.3.3.4. WIKIPEDIA PAGE RANK

In the following section, I am going to describe the final and complete version of the Page Rank algorithm for Wikipedia pages. This implementation uses the Spark GraphX library and its page rank implementation which uses convergence as a measure to finish the page rank loop. The heavy working from my side was to parse Wikipedia XML dumps, find all the internal links and create a graph which then GraphX can use. To get accurate results, I run the page rank algorithm on all Wikipedia pages and then filter the biographical pages.

Parsing Wikipedia XML dump

Wikipedia Data comes in an XML format. Here is a sample input.

```
  <page>
    <title>AccessibleComputing</title>
    <ns>0</ns>
    <id>10</id>
    <redirect title="Computer accessibility" />
    <revision>
      <id>1002250816</id>
      <parentid>854851586</parentid>
      <timestamp>2021-01-23T15:15:01Z</timestamp>
      <contributor>
        <username>Elli</username>
        <id>20842734</id>
      </contributor>
      <minor />
      <comment>shel</comment>
      <model>wikitext</model>
      <format>text/x-wiki</format>
      <text bytes="111" xml:space="preserve">#REDIRECT [[Computer accessibility]]

{{rcat shell|
{{R from move}}
{{R from CamelCase}}
{{R unprintworthy}}
}}</text>
      <sha1>kmysdltgexdwkv2xsml3j44jb56dxvn</sha1>
    </revision>
```

Figure 50 Wiki XML Dump

Wikipedia provides many fields which are not useful for my use case. For my problem, I am interested in the following fields

- **Title**: the title of the Wikipedia Page
- **Revision: text.** The sub-field text of revision is the actual Wikipedia article

After reading the XML into an RDD, I had to create a method to take out for each page all its outgoing internal links to other wiki pages.

How to Check a Page is biographic?

Wikipedia doesn't have a Boolean flag in its XML schema to denote that a page is a biography. Therefore, I had to get creative. I observed that all Wikipedia biographical pages must have an info box at the beginning of the page. This is just a small informative panel about the person that is formatted in a specific way to be rendered by Wikipedia. I then checked Wikipedia their page about info boxes which had a list of all info box types that can be assigned to an individual. I then use this list in my method to check if the current page has the info box type that corresponds to a biographical page (image below).

```
1 usage   Sirbu Cosmin (2019) ZHAC254 +1
private boolean checkPageIsBiography(String wikiPage) {
  boolean isBiography = false;
  for (String personInfoBox : wikiPersonInfoBoxes) {
    if (wikiPage.contains("{{" + personInfoBox)) {
      isBiography = true;
      break;
    }
  }
  return isBiography;
}
```

Figure 52 Checking if a page is biographical

Figure 51 Biographic Info box

How to get Wikipedia internal links?

The next step is to get all the Wikipedia internal links out of the XML dump. After carefully analysing the XML dumps, I noticed that their internal links follow the following possible formats:

- [[internal link]]
- [[internal link | internal link alias]]
- [[File: file contents]]
- [[category: category link]]
- [[page # sub-section | link]]

Using a pre-complied regex, I look for all the matching results within the page XML, parse them and add them to a list. Finally, I return a tuple with the page, the list of links and the biography flag.

```
this.wikiLinkFormat = Pattern.compile( regex: "(?<=\\[\\[)[^\\[\\]]*(?=\\]\\])");
```

Figure 53 Wiki Link Format Regex

```
private Tuple3<String, HashSet<String>, Boolean> parseWikiPage(String title, String wikiPage) {
  boolean isBiographicPage = checkPageIsBiography(wikiPage);

  Matcher linkMatcher = wikiLinkFormat.matcher(wikiPage);
  HashSet<String> links = new HashSet<>();

  while (linkMatcher.find()) {
    String link = linkMatcher.group();
    if (link.contains("File:") || link.contains("Category:")
        || link.contains("Help:") || link.contains("commons:")) {
      continue;
    } else if (link.contains("#")) {
      link = link.substring( beginIndex: link.indexOf("|") + 1);
    } else if (link.contains("|")) {
      link = link.substring(0, link.indexOf("|"));
    }
    links.add(StringUtils.normalizeSpace(link));
  }

  return new Tuple3<>(title, links, isBiographicPage);
}
```

Figure 54 Parsing Wikipedia internal links

Creating the vertices and edges list

To create a GraphX object, I need two RDDs: an edges list and a vertices list.

Firstly, I "explode" the previous (Page, [list of pages], biography) RDD into a (Page, Other Page, biography RDD)

```
private void createWikiPageTuples() {
  wikiPages = parsedWikiPages
      .flatMapToPair(v -> v
          ._2() HashSet<String>
          .stream() Stream<String>
          .map(link -> new Tuple2<>(v._1(), link)) Stream<Tuple2<String, String>>
          .iterator())
      .persist(StorageLevel.MEMORY_ONLY_SER());
}
```

Figure 55 Creating the connections RDD

To create the graph edges, I construct an RDD of GraphX Edges from the exploded connections RDD.

```
private void createGraphEdges() {
  edges = wikiPages
      .join(vertexIds) JavaPairRDD<String, Tuple2<String, Long>>
      .map(Tuple2::_2) JavaRDD<Tuple2<String, Long>>
      .mapToPair((v) -> v) JavaPairRDD<String, Long>
      .join(vertexIds) JavaPairRDD<String, Tuple2<Long, Long>>
      .map((edge) -> new Edge<>(edge
                                    ._2()
                                    ._1(), edge
                                    ._2()
                                    ._2(),  attr: ""));
}
```

Figure 56 Creating Wikipedia Graph Edges

To create the graph vertices, I assign a unique id to each Wikipedia page

```
private void createVertexIds() {
  vertexIds = wikiPages
      .map(Tuple2::_2) JavaRDD<String>
      .distinct()
      .zipWithUniqueId() JavaPairRDD<String, Long>
      .persist(StorageLevel.MEMORY_ONLY_SER());
}
```

Figure 57 Creating Graph Vertices

The Wikipedia Page Rank

After constructing the vertices and edges RDDs, I can construct the Wikipedia Page Graph and run the page rank algorithm.

I first create the GraphX graph from the two RDDs.

```
this.pageRankGraph = Graph.apply(vertices.rdd(), edges.rdd(),  defaultVertexAttr: "", StorageLevel.MEMORY_ONLY_SER(),
                        StorageLevel.MEMORY_ONLY_SER(), tagString, tagString);
```

Figure 58 Wikipedia Connections Graph

Then I let it compute the page ranks until a convergence of 0.0001 with a reset probability of 0.15.

```
private void computePageRank() {
  pageRankVertices = PageRank
      .runUntilConvergence(pageRankGraph, convergence, resetProbability, evidence$13: null, evidence$14: null)
      .vertices()
      .toJavaRDD()
      .mapToPair(v -> v)
      .persist(StorageLevel.MEMORY_ONLY_SER());
}
```

Figure 59 Running Page Rank

2.3.3.5. DEPLOYMENT

As an input, I have the Wikipedia XML dumps which my program parses and produces a list of all the biographical Wikipedia pages, ranked from the best to the worst. Here is a sample of some famous historical figures and their page ranking.

The resulting file is not large, but it has a large number of records **(approximately 1.2 million** corresponding to the number of biographic Wikipedia articles). For the full results please check the output/PageRank.csv directory of my project.

Person	Page Rank
Charles Dickens	250.616
Benito Mussolini	247.36826
Plato	246.92691
Elton John	246.0858
The Rolling Stones	245.4747
Oliver Cromwell	245.42627
George VI	244.45656
Madonna	243.85686
David Bowie	243.09612
Pope Benedict XVI	242.70044

2.3.3.6. EXPERIMENTS

For the Page Rank Problem, I performed a series of experiments with different parameter settings and code modifications to achieve the best performance.

EXPERIMENT 1: CACHING

Description

Run the Page Rank problem on the RHUL Big Data Cluster with and without caching intermediate results.

Goal

I want to see an improvement in execution when the problem is executed with and without caching.

Explanation

Caching can greatly improve a Spark program's performance because it doesn't have to re-compute previous transformations and get just access the value from the memory and apply the new transformation to it. However, we have to be careful with caching as this can come with certain problems. Please refer to the Part 2 Caching Section. For my specification, I used Memory Only Serialised with the Kryo Serializer.

The first problem is caching unnecessary RDDS or Dataframes. You should only cash intermediate results between transformations. If you cache other RDDs besides these, a lot of time will be lost on the actual caching process which will be unnecessary.

The second problem is using the wrong level of caching. Depending on your job and cluster configuration, using the wrong level of caching can decrease performance. For example, caching on the disk rather than in memory is time costly but can be beneficial for certain types of jobs where the intermediate transformations are too large to fit in memory.

Results

Configuration	Run Time
Without Caching	85 min
Caching without Serialization	72 min
Caching with Java Serializer	66 min
Caching with Kryo Serializer	60 min

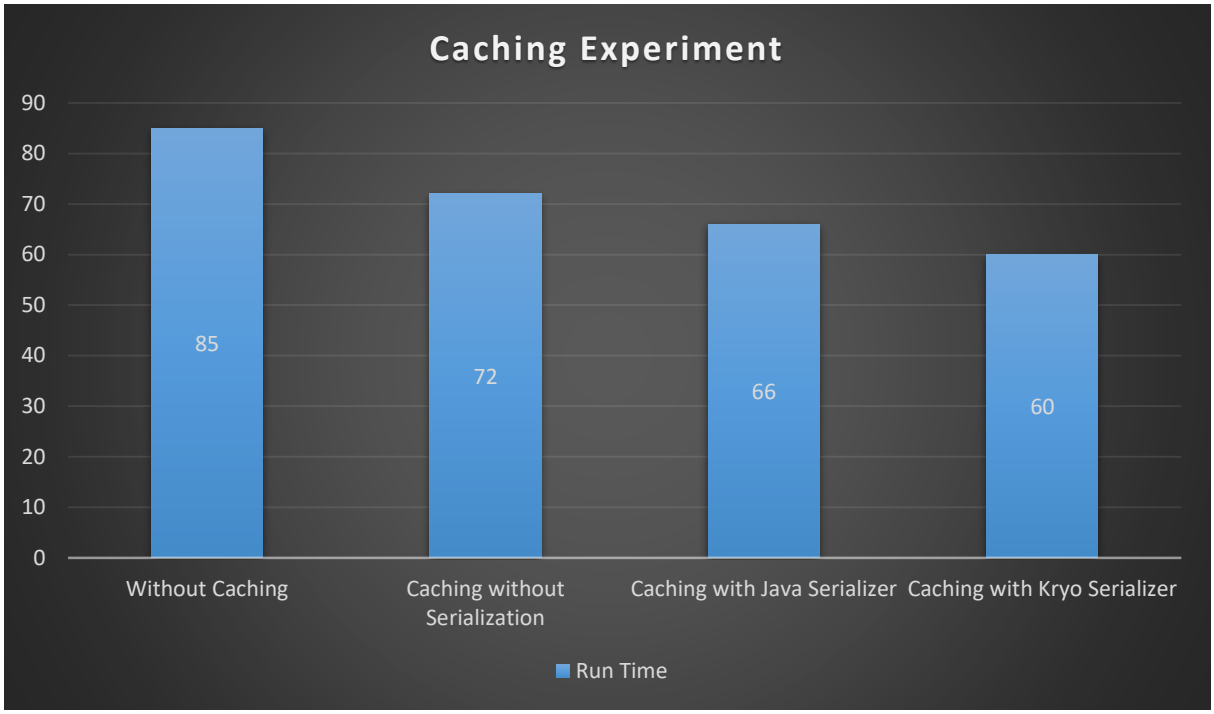


Figure 60 Page Rank Caching Experiment

Conclusion

By storing the intermediate transformation results in memory in a serialised form I achieved a great improvement in execution time because of:

- Not re-calculating intermediate steps.
- Better compactness for storing caching results in memory (because of Kryo serialization).

EXPERIMENT 2: SERIALIZATION

Description

Run the Page Rank algorithm on the RHUL Big Data Cluster with and without Serialization.

Goal

I wanted to see a performance improvement when using serialization.

Explanation

As we have seen in the Introduction to Spark: Serialization chapter, serialization can improve the performance of your Spark programs because it shrinks to objects when they are being transmitted over the cluster, therefore improving I/O speed. Also, the if you are using serialization with Kryo, the benefits are even greater since it can vastly outperform Java serialization.

Results

Configuration	Run Time
Without serialization (or caching)	85 min
Best Caching with Kryo	60 min
Caching + Java Serialization	62 min
Kryo Caching + Kryo Serialization	55 min

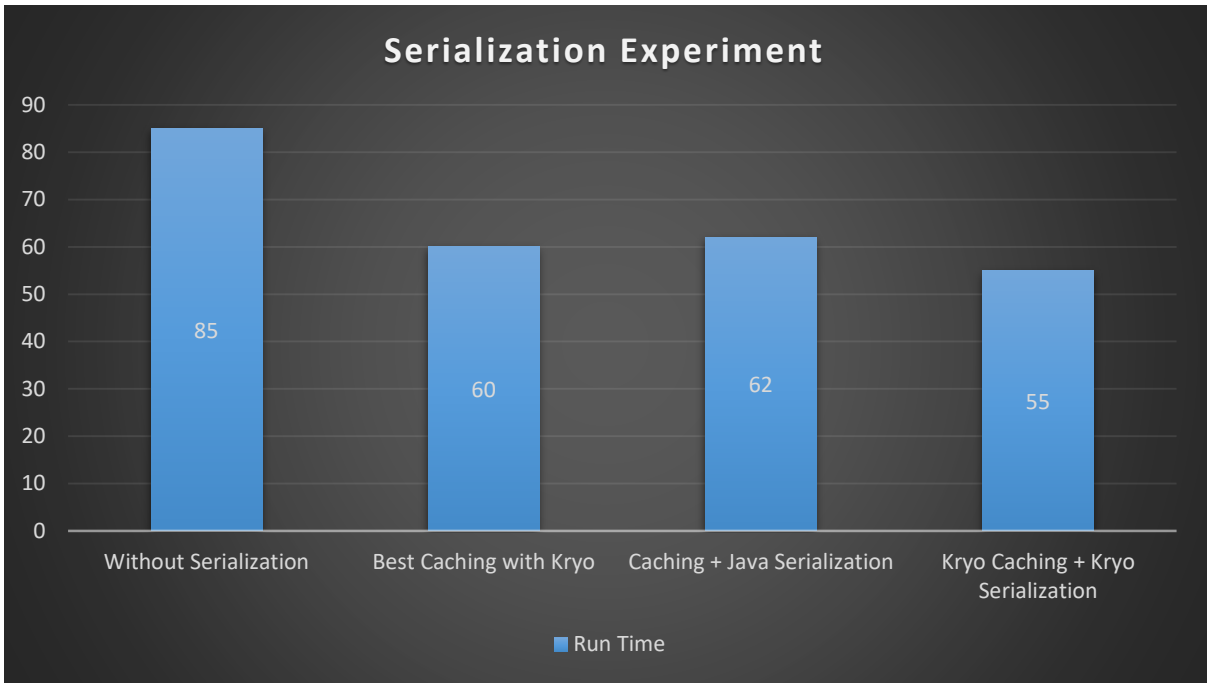


Figure 61 Page Rank Serialization Experiment

Conclusion

By using a good Serializer like Kryo we greatly increase I/O speeds (which are by default costly) and can greatly improve the performance of the programs.

2.3.3.7. PERFORMANCE ANALYSIS

Description

After optimizing the program with caching and serialization, I wanted to optimize it even further by fine-tuning some job parameters. The ones I was interested in the most were default parallelism, shuffle partitions and the number of executors and cores.

Goal

Compare and analyse the performance of the Page Rank job on the RHUL Big Data Cluster in terms of execution time.

Explanation

For the number of executors and cores, I chose by default the greatest values possible on the cluster (10 nodes with 3 cores).

Regarding the number of shuffle partitions, the default is 200. For my data size, this is quite bad since it will result in a large number of very small partitions (some even empty!). This automatically means worse execution times. As a rule of thumb, for medium-sized clusters (like RHUL Big Data Cluster) with medium-sized data like the Wikipedia one, the number of shuffle partitions should be 1x or 2x the number of cores. So in my case either 3 or 6 should be the ideal number.

Regarding the default parallelism, it is by default equal to the total number of cores on all executor nodes or 2, whichever is larger. After analysing Spark UI partition size with this setting, I saw that partition size is relatively constant around 120MB. I was happy with this size as this is exactly what a partition size should be so I left the setting to default (in my case this will be 10 nodes x 3 nodes/executors = 30).

Results

Experiment Number	Executor Cores	Executors Number	Default Parallelism	Shuffle Partitions	Best Caching	Run Time
1	10	3	30	200	Yes	70min
2	10	3	30	200	No	90 min
3	10	3	30	3	Yes	50 min
4	10	3	30	3	No	65 min
5	10	3	30	6	Yes	54 min
6	10	3	30	6	No	69 min

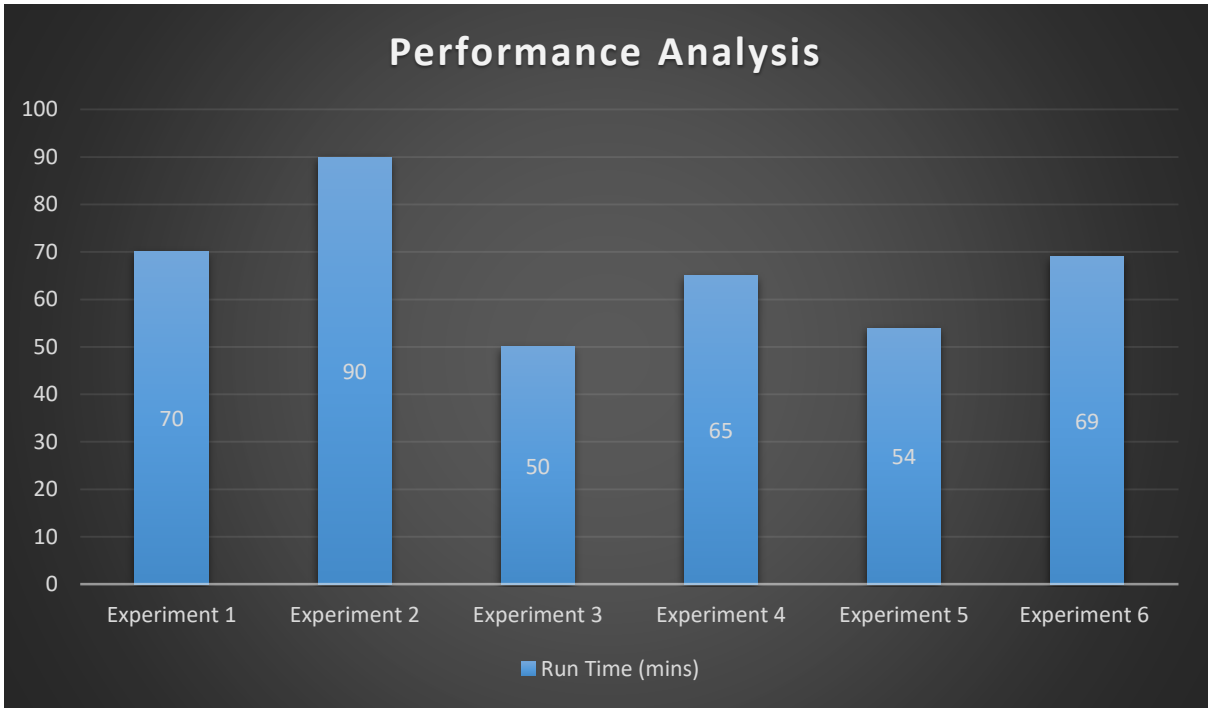


Figure 62 Page Rank Performance Analysis

Conclusion

In this program, we have seen the importance of adequate parameter tuning, caching and serialization in the performance of Spark programs.

- Spark is a truly powerful enterprise-grade tool. We have seen that Hadoop can perform relatively well given the right configuration but it doesn't come close to Spark's performance, ease of use and scalability.
- For Big Data problems it's crucial to think about the impact of every small operation on the overall job performance. I was required to constantly adapt my strategy, consult existing research and experiment to obtain the best performance.

2.3.3.8. OPTIMISED SPARK PARAMETERS

In the following table, there are my proposed parameters for an optimal Page Rank job execution using Spark. Please recall the Parameters Section to understand these settings better.

Parameter	Parameter Value	Explanation
Default Parallelism	30	The default value of 30 worked just fine.
Spark Serializer	Kryo Serializer	As we have seen this gives the best object size and speed.
Kryo Registration	Not Required	The difference between manual registration and auto was very small.
Driver Memory	2GB	The driver doesn't require much memory.
Executor Memory	20GB	With that much caching going on and the large data size, having a value smaller than 20GB might be problematic
The number of executors	10	The maximum number of executors available.
Executor Cores	3	Maximum number of cores available
The number of shuffle partitions	3	Between 3 and 6, 3 proved to be giving slightly better results.
Extra archived files	None	N/A
Log Level	ERROR	Avoiding printing a large number of INFO logs increased performance.
Jars Packages	None	N/A

2.3.4. NATURAL LANGAUGE PROCESSING MODEL FOR TEXT SEMANTICS CATEGORIZATION

2.3.4.1. OVERVIEW

The purpose of this program is to take a biographical Wikipedia article and accurately predict its semantic category. For the scope of this project, I was interested in broad categories such as arts, science, sports, etc. This functionality is not provided by Wikipedia and I will next briefly discuss why in the next section.

Wikipedia Category Graph

My initial approach was to use existing Wikipedia article categories and create a knowledge base (KB) from which I could infer broad categories for my biography articles. Wikipedia has a way of internally structuring its articles through the use of a very complex category system. This system however is unsuitable to use for two reasons.

The first reason is that the categories that are assigned to articles are not relevant enough, as they are too granular. As an example, for Julius Caesar, some of the categories assigned to his page are:

- Pontifices maximi of the Roman Republic
- Populares
- Roman governors of Hispania
- Roman military writers

All these categories are indeed relevant to the famous roman emperor. However, they are not an adequate category for my use case, as this would have been “politics”.

The second reason is that Wikipedia structures its categories in a directed acyclic graph (DAG), going from very broad categories to very granular ones. A DAG is a type of graph where connections between nodes are directional and it doesn’t have cycles (no matter what path you are taking while traversing the graph, you can’t go from a node to itself).

This was initially looking very promising because that meant that a broad category could have been inferred from a granular one by finding the initial parent. Theoretically, that meant that from the Roman governors of Hispania, the category “politics” could have been inferred. The problem with this approach is that the Wikipedia category graph is non-hierarchical. Therefore, a category could have multiple broad categories of parents, not useful for my use case. For example, the category Roman governors of Hispania can fall under the category politics but also history. This means I couldn’t properly create a knowledge base.

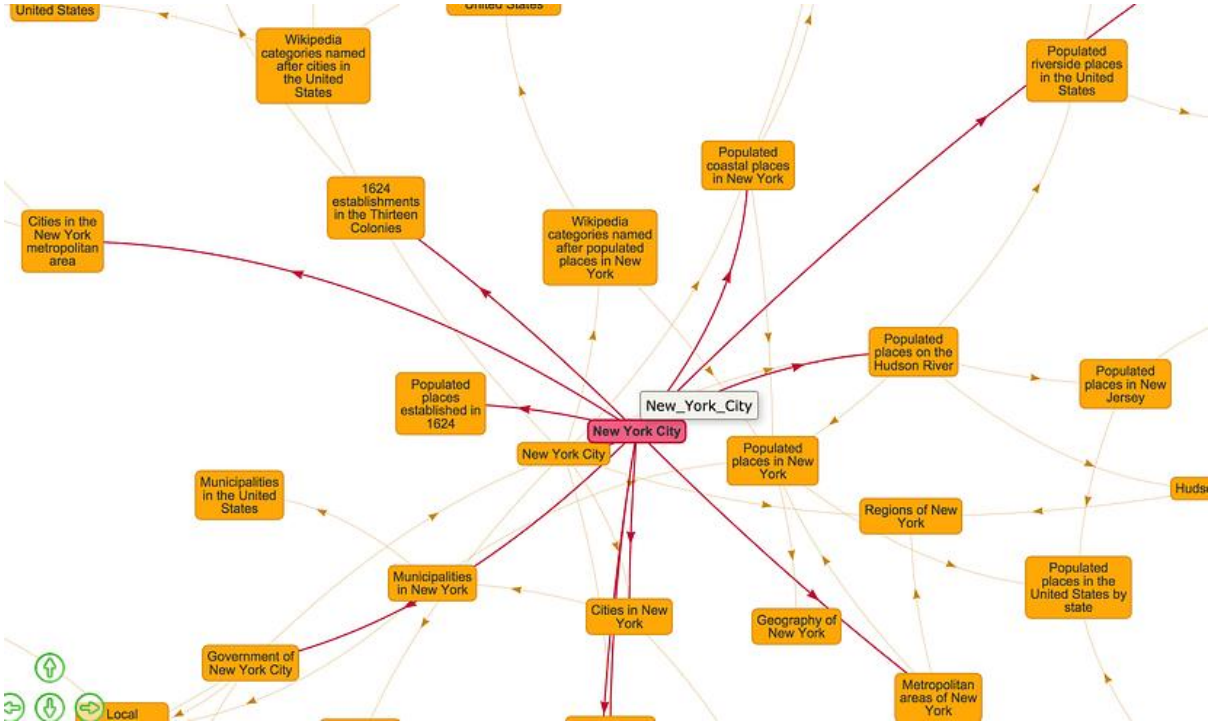


Figure 63 Wikipedia Category Graph [36]

2.3.4.2. IMPLEMENTATION

In the development of the Natural Language Processing Model, I iteratively developed two versions of the implementation:

- Linear Model: developed with Scikitlearn, Pandas, nltk and python. It was created for a linear (non-distributed) execution.
- Spark Model: developed with Spark, SparkML and nltk. It was created for a distributed execution on the RHUL Big Data Cluster.

The process of creating the model is very similar between the two versions. What will be important here is discussing the scalability and performance benefits of the Spark version.

2.3.4.3. LINEAR MODEL

In the following section, I am going to explain my approach to creating the model, going from one step of the pipeline to the next (illustrated below).

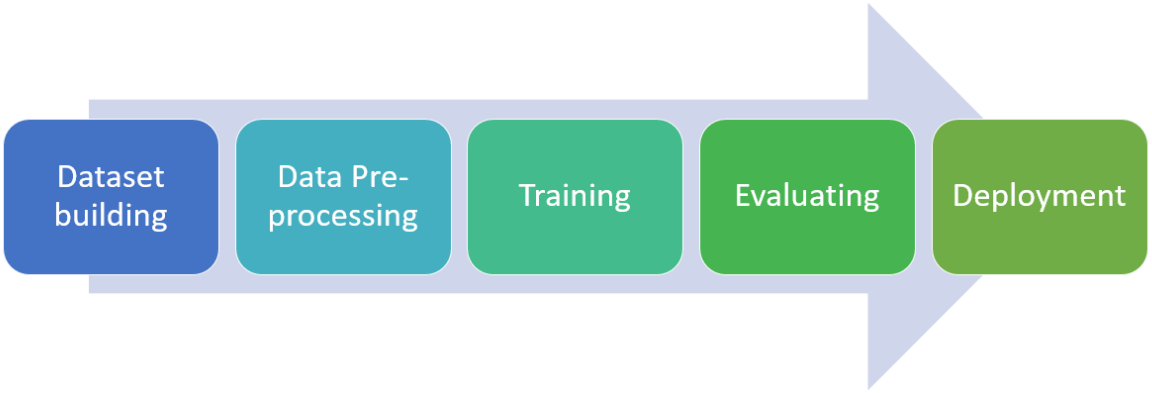


Figure 64 Model Creation Pipeline

DATASET BUILDING

As was explained in the theory section, data gathering is a crucial part of the model's success. I hand-crafted the training dataset by collecting, processing and labelling a large number of Wikipedia articles. For each category, I generated 5000 entries of 60 words in length. The dataset-building pipeline is illustrated below.

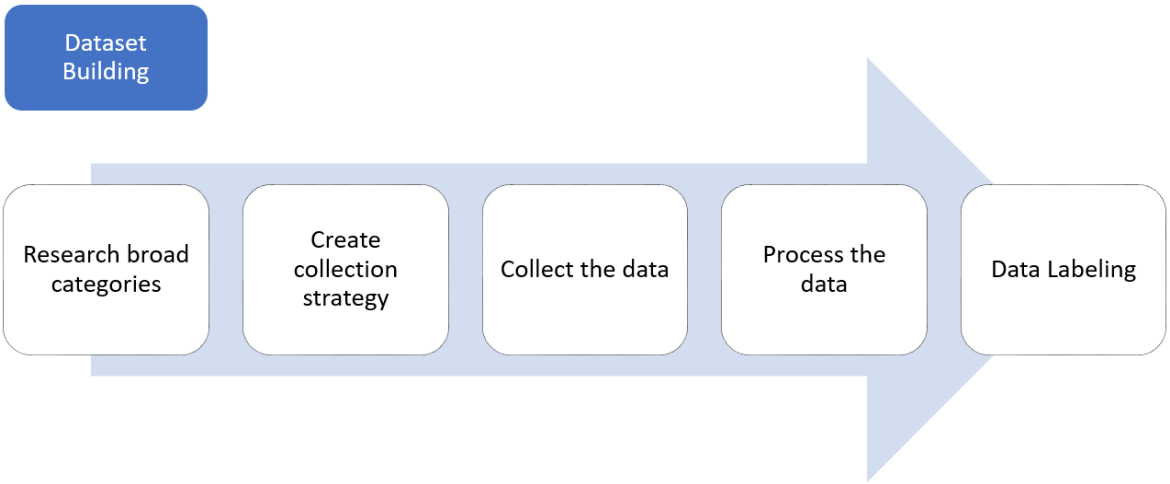


Figure 65 Dataset Building Pipeline

1. Research broad categories

After an extensive analysis of human occupations from ancient times to modern days, I boiled down the category classes to the following:

Culture & Arts

Includes all of the arts: painting, sculpture, literature, cinema, music, etc… It also includes scholars from ancient times or modern occupations such as digital artists, modern singers and painters, rock stars, etc…

Entertainment

Includes mainly modern-day comedians, TV stars, you-tubers, influencers and so on.

Military

Includes everything regarding human conflict, warfare, law enforcement, revolutionaries, and other related areas from across human history.

Philosophy

Includes all the great thinkers, philosophies, and movements from across human history, from Plato to Kant and feminism.

Politics & Leaders

Includes everything regarding politics and state ruling or leadership from across history.

Religion

Includes religions articles from across the main religious groups and movements from across the world and history.

Science & Technology

Includes everything regarding science, knowledge, and modern technology.

Sport

Includes sports articles from a great variety of backgrounds, from football to boxing and quidditch.

2. Create Collection Strategy

For each of the categories above, I researched their main topics and periods, trying to be as unbiased as possible towards an era or culture. Using my research, I wrote down the main topics for each category. I aimed to collect the data from a broad range of sources, semantic backgrounds, cultures and historical periods to achieve a good generalization that would ensure that my model is easily extensible to real-world data.

3. Collect the Data

Using my collection strategy, for each category I browsed and copied Wikipedia articles [10] according to the researched topics. I then saved all these articles in separate text files.

4. Process the Data

To parse the raw Wikipedia data from the text file, I created a Spark program in Java. This program takes in a raw text file, parses it to remove any non-alphabetic characters, normalizes spaces and splits the file into paragraphs of 60 words.

```
1 usage  Cosmin Sirbu +1 *
private static void parseWikiText(SparkSession spark, String inputPath, String outputPath) {
  //delete output path
  FileUtils.deleteQuietly(new File(outputPath));

  JavaRDD<String> lines = spark.sparkContext() SparkContext
                              .wholeTextFiles(inputPath, minPartitions: 1) RDD<Tuple2<String, String>>
                              .toJavaRDD() JavaRDD<Tuple2<String, String>>
                              .mapToPair(v -> v) JavaPairRDD<String, String>
                              .reduceByKey((a, b) -> a + " " + b)
                              .values() JavaRDD<String>
                              .coalesce( numPartitions: 1);


  JavaRDD<String> cleanedLines = lines.flatMap(TextParer::breakIntoParagraphs);

  cleanedLines.saveAsTextFile(outputPath);
}
```

Figure 66 Data Gathering processing

The first part of the program reads the file as a whole file and combines all rows in the file into a string. This is done by mapping each row to the file name and then reducing all the values by key (the file name), combining all rows into one.

```
1 usage  Cosmin Sirbu *
private static Iterator<String> breakIntoParagraphs(String line) {
  line = line.toLowerCase()
             .replaceAll( regex: "[^a-zA-Z]",  replacement: " ");

  String[] words = StringUtils.normalizeSpace(line)
                              .split( regex: "\\W+");

  ArrayList<String> newLines = new ArrayList<>();

  StringBuilder newLine = new StringBuilder();
  for (int i = 0; i < words.length; i++) {
    newLine.append(words[i])
           .append(" ");
    if ((i % paragraphLength == 0 && i != 0) || i == words.length - 1) {
      newLines.add(newLine + ",");
      newLine = new StringBuilder();
    }
  }

  return newLines.iterator();
}
```

Figure 67 Data Gathering Processing

The second part of the program cleans the text and breaks it into a set of 60 words lines. This part essentially "explodes" the string and re-maps with the flat map to then save it as a text file.

5. Data Labelling

Finally, I took all the generated text files, labelled them with their according category and saved them in a CSV file.

	A	B	C	D	E	F	G	H	I	J
1	category	text								
2	sport	of the bar the lifter shall face the front of the platform on completion of the lift the knees shall be								
3	science & technology	be associated with severe bicycling injuries to date the majority of injury risk studies have focused								
4	politics & leaders	debate in australia of course the conservatives are in the liberal party jupp writes that the decline i								
5	military	cable to stalin emphasizing the difficulties faced by chinese forces and the need for air cover espec								
6	religion	yet having a profound influence on the fabric of japanese society overall during the edo period the								
7	religion	by the romans there were also many deities that existed in the roman religion before its interactior								
8	religion	composite deity this process was a recognition of the presence of one god in another when the sec								
9	military	land both military orders were accumulating holdings in the kingdom and crusader states with the h								
10	culture & arts	d film that are unnatural for human vision crosstalk between the eyes caused by imperfect image s								
11	philosophy	philosophy spinoza logique du sens the logic of sense spinoza paris puf dialogues nd exp ed with cla								
12	science & technology	ongoing access to privilege for those with relative power and the reduction of the other to a lesser								
13	sport	in skiing can lead to death or permanent brain damage in alpine skiing for every people skiing in a d								
14	religion	of judging being against something that might not exist led many medieval philosophers approach t								
15	entertainment	channel and extend discussion outside of class time censorship by governments main article interne								
16	military	terrain that was shell torn and often impassable to traffic following operation michael germany lau								
17	culture & arts	decline of painting in his period as for painting greek painting was utterly lost neoclassicist painters								

Figure 38 Final training data

DATA PRE-PROCESSING

After the data is gathered, the next is pre-processing. At the end of this step, an input dataset will be produced which will be usable for the model training. The pre-processing pipeline can be seen below.

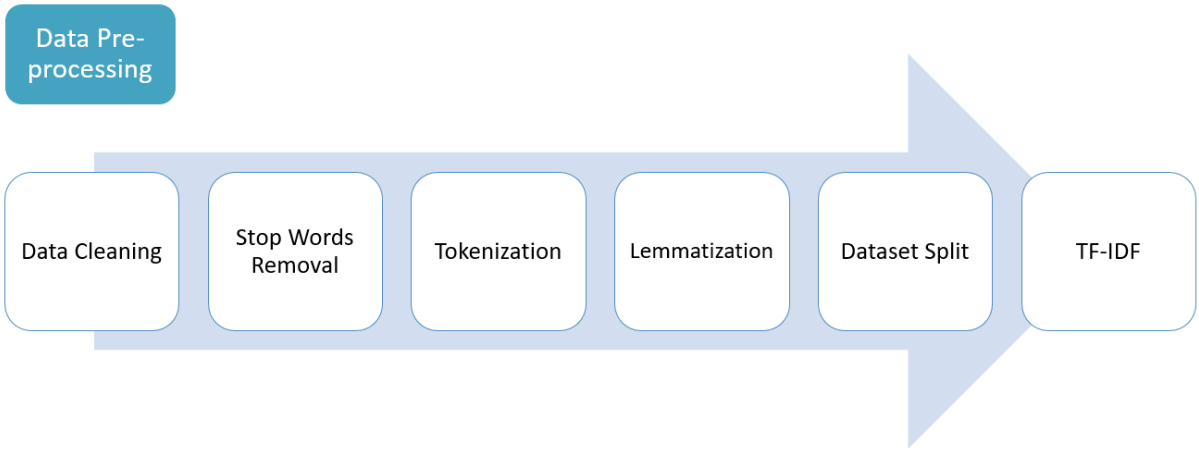


Figure 69 Data Pre-processing Pipeline

1. Data Cleaning

In this step, all non-alphabetic characters, links and tags are removed and the string space is normalised and converted to lowercase.

```
Cosmin Sirbu *
def clean_text(text):
    wiki_clean_regex = re.compile(r'http\S+|<.*?>*?</.*?>|{{.*?}}|[^a-zA-Z]')
    text = re.sub(wiki_clean_regex, " ", text)
    text = ' '.join(text.split())
    text = text.lower()
    return text
```

Figure 70 Data Cleaning method

2. Stop Words Removal

In this step, the Natural Language Toolkit Library (nltk) was used to remove the text stop words. After downloading the nltk English stop words dataset, I converted it into a set. I then removed all words in the text which were part of the stop words set.

```
Cosmin Sirbu *
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    no_stopword_text = [w for w in text.split() if w not in stop_words]
    return ' '.join(no_stopword_text)
```

Figure 71 Stop Words Removal

3. Tokenization

In this step, I used nltk and its tokenization library to convert the text into a list of 1-gram tokens. I then remove all non-alphabetic tokens.

```
tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
filtered_tokens = []
for token in tokens:
    if re.search(alphabetic_regex, token):
        filtered_tokens.append(token)
```

Figure 72 Tokenization method

4. Lemmatization

In this step, I used the word net lemmatizer from nltk to convert the filtered tokens into lemmas.

```
lemmatizer = WordNetLemmatizer()
lem = [lemmatizer.lemmatize(t) for t in filtered_tokens]
return lem
```

Figure 73 Lemmatization method

5. Dataset Split

In this step, I randomly split the dataset into training and testing datasets using the scikitlearn dataset split method. The training dataset has 80% of the initial dataset and the testing one 20%. This is a normal split ration in Machine Learning and it ensures the training part and testing parts are completely separated.

```
Cosmin Cristian Sirbu
def _dataset_split(self):
    x = self._data.loc[:, 'clean_text']
    y = self._data.loc[:, 'category_id']
    self._x_train, self._x_test, self._y_train, self._y_test = train_test_split(x, y, test_size=0.2,
                                                                                 random_state=55)
```

Figure 74 Dataset split method

6. TF-IDF

In this step, I used the TF-IDF vectorizer module from Scikitlearn to convert the lemmatized dataset into a TF-IDF. Firstly, I created the TF-IDF model and trained it only on the training dataset.

```
Cosmin Cristian Sirbu
def _train_tfidf_model(self):
    self._tfidf_vec = TfidfVectorizer(
        stop_words='english',
        ngram_range=(1, 2),
        tokenizer=tokenize_and_lemmatize,
        use_idf=True) \
        .fit(self._x_train)
```

Figure 75 TF-IDF training

Then I used the trained TF-IDF model to transform both the training and testing datasets.

```
self._x_train_tfidf = self._tfidf_vec.transform(self._x_train)
self._x_test_tfidf = self._tfidf_vec.transform(self._x_test)
```

Figure 76 TF-IDF transformation

TRAINING

Training the model is a straightforward process illustrated below.

Figure 77 Training Pipeline

For this task, I used the modules SVC and OneVsRestClassifier from scikitlearn. SVC is scikitlearn's implementation of an SVM (see section) and OneVsRestClassifier is a heuristic method for using binary classifiers like SVM for multi-class classification [27]. This method splits the multi-class dataset into multiple smaller binary classification problems [27].

```
Cosmin Cristian Sirbu *
def _train_model(self):
    self._model = OneVsRestClassifier(LinearSVC(C=0.5, max_iter=10))
    self._model.fit(self._x_train_tfidf, self._y_train)
```

Figure 78 Model Training

EVALUATING

After the model is trained on the training set, I evaluate its performance on the testing set (which is previously unseen data). The process is illustrated below.

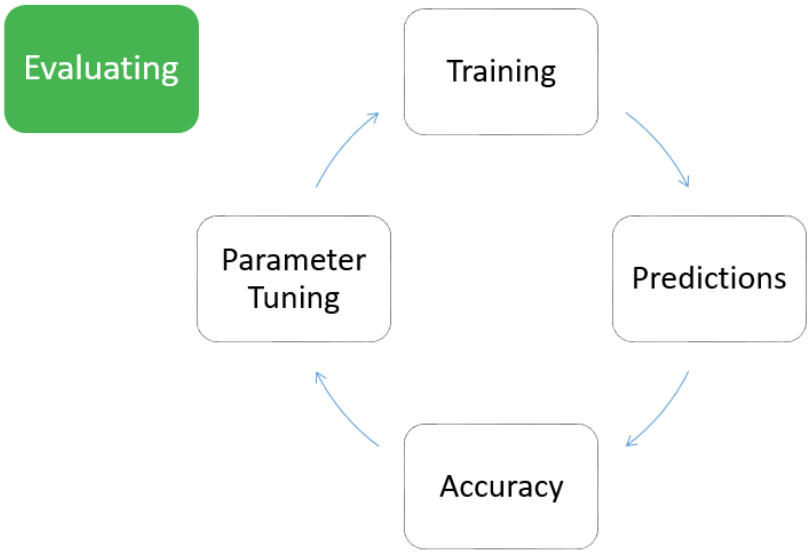


Figure 79 Evaluating

As mentioned in section x, fine-tuning the regularization parameter and the number of iterations can greatly improve the accuracy of the model. For this purpose, I used the grid search cv module from Scikitlearn. Given a list of parameter values and a model, grid search cv trains your model with all the possible parameter combinations and returns the ones that give the best accuracy.

The best accuracy of 95% for my model is obtained with C=0.5 and max_iter=10.

```
new *
def _grid_search(self):
    self._model = OneVsRestClassifier(LinearSVC())
    parameters = {
        "estimator__C": [0.0001, 0.001, 0.01, 0.1, 0.5, 1, 10, 100, 100],
        "estimator__max_iter": [1, 5, 10, 50, 100, 500]
    }
    model_tunning = GridSearchCV(self._model, param_grid=parameters)

    model_tunning.fit(self._x_train_tfidf, self._y_train)

    print(model_tunning.best_score_)
    print(model_tunning.best_params_)
```

Figure 40 Parameter tuning

DEPLOYMENT

After the model is fined tuned and trained, the last step is to test it on real-world Wikipedia data. The process is illustrated below.

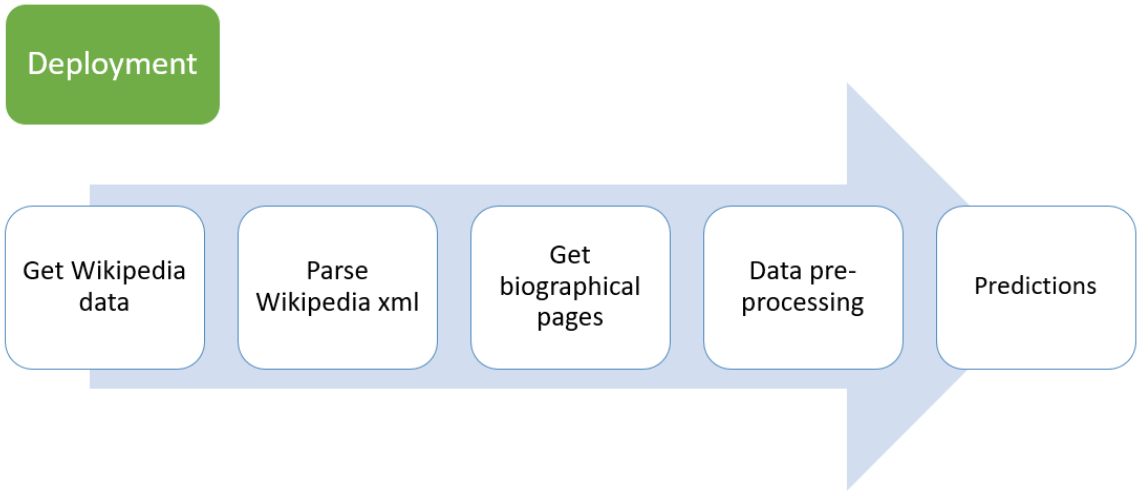


Figure 81 Deployment Pipeline

To do so, I downloaded the 2020 English Wikipedia from [13]. This contains an XML with all the English Wikipedia articles. It is about 100 GB in size. Because this model is created in plain python with Scikitlearn, applying my model to such a huge amount of data is very problematic and impractical. This is because the Scikitlearn model is not designed to handle massive amounts of data like this. Therefore, for some preliminary results, I selected a smaller 100MB file, parsed it, pre-processed it again (with the trained TF-IDF) and ran my model on it. Some results can be seen below. As you can see, the model is accurate on real-world data but lacks scalability. This issue is solved by Spark. In the next section, I will describe how I used Spark to scale my model to efficiently predict the categories of all people on Wikipedia.

Person	Predicted Semantic Category
Abraham Lincoln	politics & leaders
Aristotle	philosophy
Ayn Rand	philosophy
Alain Connes	science & technology
Ande Agassi	sport
Abel	religion
Arthur Schopenhauer	philosophy
Alfred Nobel	science & technology
Alexander the Great	military
Attila	military

Figure 82 Preliminary model results

2.3.4.4. SPARK MODEL

We have already seen the benefits of using spark instead of Scikitlearn. The question is how to do it. Scikitlearn or other Machine Learning libraries cannot be properly used with Spark because they are not part of its API and therefore cannot be parallelised. Therefore, for this task, I used Spark's own Machine Learning library, SparkML. This library contains the most popular machine-learning tools and programs and allows for efficient processing through parallelism during training and testing. Furthermore, the pipeline is the same and the machine learning approach and API are similar. What needs adapting is the usage of Spark data frames instead of the Pandas data frames I used for the normal model.

DATA PRE-PROCESSING

Pre-processing is very similar to Spark. The same transformations are done, but this time cleaning, stop word removal, tokenization and lemmatization methods are user-defined functions (UDF)s which are applied to a column of the data frame. A UDF is a kind of spark transformation which is done in parallel on the whole column of the data frame (see below).

```
Cosmin Cristian Sirbu +1
def _pre_process_data(self):
    self._data = self._data \
        .withColumn('pre_processed_data', clean_text(col('text'))) \
        .withColumn('pre_processed_data', remove_stopwords(col('pre_processed_data'))) \
        .withColumn('pre_processed_data', tokenize_and_lemmatize('pre_processed_data')) \
        .cache()
```

Figure 83 Pre-processing with Spark

```
Cosmin Cristian Sirbu *
@udf(returnType=StringType())
def clean_text(text):
    text = re.sub(wiki_clean_regex, " ", text)
    text = ' '.join([word for word in text.split() if len(word) > 1])
    text = text.lower()
    return text
```

Figure 84 Clean text UDF

TF-IDF

SparkML doesn't directly provide a TF-IDF module. Instead, it provides the TF and IDF modules separately through its HashingTF and IDF modules. Therefore, the TF-IDF step becomes slightly longer.

Firstly, the HashingTF is used to hash the input.

```
Cosmin Cristian Sirbu
def _tf_transformation(self):
    hashing_tf = HashingTF(
        inputCol='pre_processed_data',
        outputCol='tf')
    self._data = hashing_tf \
        .transform(self._data) \
        .cache()
```

Figure 85 Spark TF transformation

Secondly, the IDF is created and trained on the training data frame.

```
Cosmin Cristian Sirbu
def _train_tf_idf_model(self):
    idf = IDF(
        inputCol='tf',
        outputCol='tfidf')
    self._idf_model = idf.fit(self._train_df)
```

Figure 86 Spark TF-IDF training

Thirdly, the trained IDF is used to create the TF-IDF for the training and testing data frames.

```
self._train_df = self._idf_model \
    .transform(self._train_df) \
    .cache()
self._test_df = self._idf_model \
    .transform(self._test_df) \
    .cache()
```

Figure 87 Spark TF-IDF transformation

TRAINING

As before I am using an SVM with an OneVsRestClassifier. Since it is the same kind of model and I already performed the parameter optimization with GridSearchCv with the Scikitlearn model, I am using the same parameters.

```
Cosmin Cristian Sirbu *
def _train_model(self):
    lsvc = LinearSVC(maxIter=10, regParam=0.5)
    ovr = OneVsRest(
        classifier=lsvc,
        predictionCol="prediction",
        featuresCol="tfidf",
        labelCol="category_id",
        parallelism=2)
    self._model = ovr.fit(self._train_df)
```

Figure 88 Spark Model Training

EVALUATING

Evaluating the model performance with Spark is done with its MulticlassClassificationEvalutor module that takes in the predictions and the correct column and computes the model accuracy. Unsurprisingly, the Spark model has a slightly different, but very similar **accuracy of 94%.**

```
Cosmin Cristian Sirbu
def _check_accuracy(self):
    evaluator = MulticlassClassificationEvaluator(
        metricName="accuracy",
        predictionCol="prediction",
        labelCol="category_id")
    predictions = self._model.transform(self._test_df)
    print("Accuracy: {}".format(evaluator.evaluate(predictions)))
```

Figure 89 Evaluating with Spark

2.3.4.5. DEPLOYMENT

Here is the part where Spark shines. Because of the Spark architecture, the trained model can now predict the categories of every biographical page on Wikipedia. The input to the Spark job is the 20GB archived file with the entire Wikipedia. The prediction system on the RHUL Big Data Cluster looks like this:

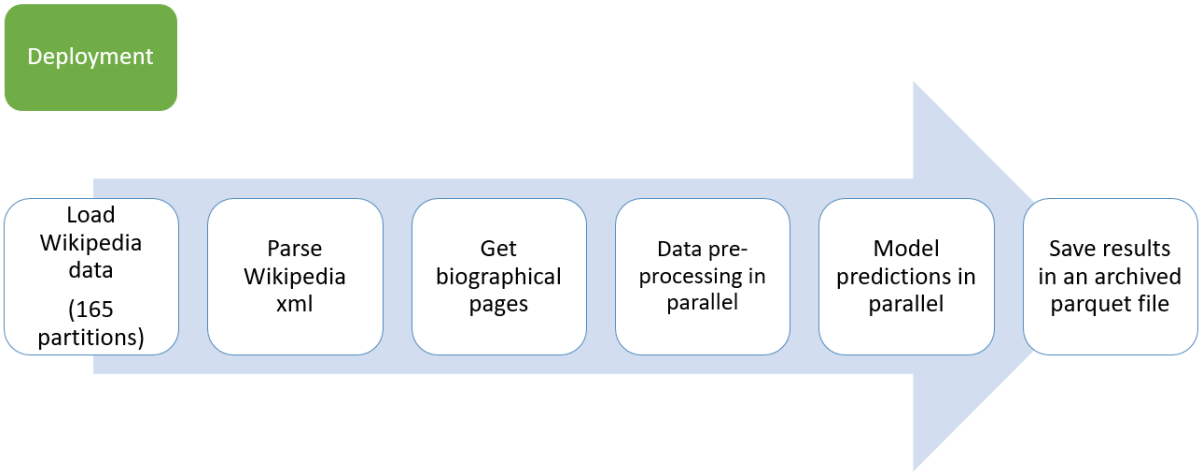


Figure 90 Spark deployment pipeline

Since the model was created in python using several external packages and dependencies, distributing the program becomes problematic. This is because python programs cannot be easily archived and distributed (like Java jars). This is important because all the required packages shouldn't be downloaded on each of the executor and master nodes. The solution is to create a virtual environment and ship it with the spark python file via spark-submit. To do so, I used anaconda to create a virtual environment with all my packages. At run time, I programmatically set the PYSPARK-PYTHON environment variable to use my virtual environment instead of the executor file system and pass in the archived anaconda environment.

```
conda create -y -n pyspark_env -c conda-forge pyarrow pandas conda-pack nltk
```

Figure 51 Creating the anaconda environment

Finally, here is a partition of the machine learning model result executed on the whole of Wikipedia. For full results check the output/SemnaticsClassifier.csv directory of my project.

Person	Predicted Semantic Category
Paul McCartney	entertainment
Mikhail Gorbachev	politics & leaders
Pablo Picasso	culture & arts
Catherine the Great	politics & leaders
The Rolling Stones	entertainment
Madonna	entertainment
Isaac Newton	philosophy
Frank Sinatra	entertainment
J. R. R. Tolkien	culture & arts
Sigmund Freud	philosophy

Figure 92 Wiki Category predictions

2.3.4.6. EXPERIMENTS

For the Natural Processing Model, I performed a series of experiments with different parameter settings and code modifications to achieve the best performance.

EXPERIMENT: CACHING

Description

Run the Natural Language Processing Model on the RHUL Big Data Cluster with and without caching intermediate results.

Goal

I want to see an improvement in execution when the problem is executed with and without caching.

Explanation

Same explanation as for Page Rank Experiment 1. The only difference is that I used PySpark, so Kryo serialization will not be useful here since PySpark stores data as byte objects.

Results

Configuration	Run Time
Without Caching	**30 min**
With Caching	**19 min**

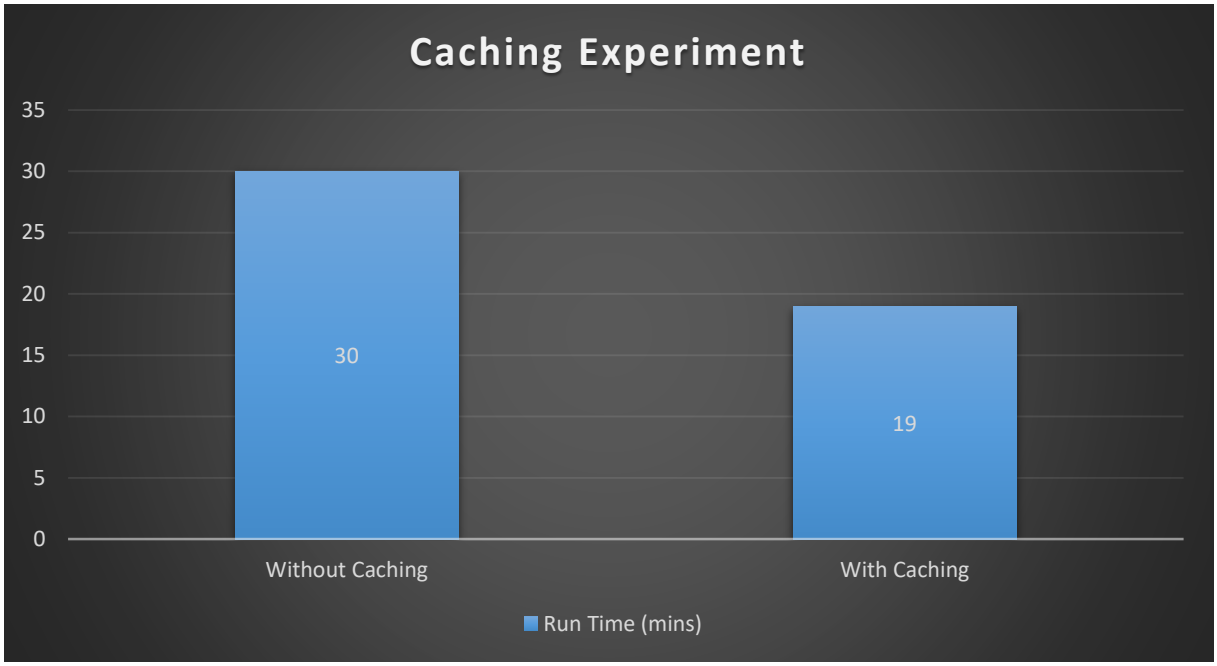


Figure 93 Semantic Classifier Caching Experiment

2.3.4.7. PERFORMANCE ANALYSIS

Description

After optimizing the program with caching, I wanted to optimize it even further by fine-tuning some job parameters. The ones I was interested in the most were default parallelism, shuffle partitions and the number of executors and cores.

Goal

Compare and analyse the performance of the Natural Language Processing Model job on the RHUL Big Data Cluster in terms of execution time.

Explanation

Same explanations as for Page Rank Performance Analysis. However, since I used Dataframes, the default parallelism won't have any effect. Dataframes are more optimised than RDDs so modifying the number of

partitions during the job wouldn't have a great positive effect since Spark splits Dataframes efficiently in 120MB blocks.

Results

Experiment Number	Executor Cores	Executors Number	Shuffle Partitions	Caching	Run Time
1	10	3	200	No	39 min
2	10	3	200	Yes	35 min
3	10	3	3	No	30 min
4	10	3	3	Yes	19 min

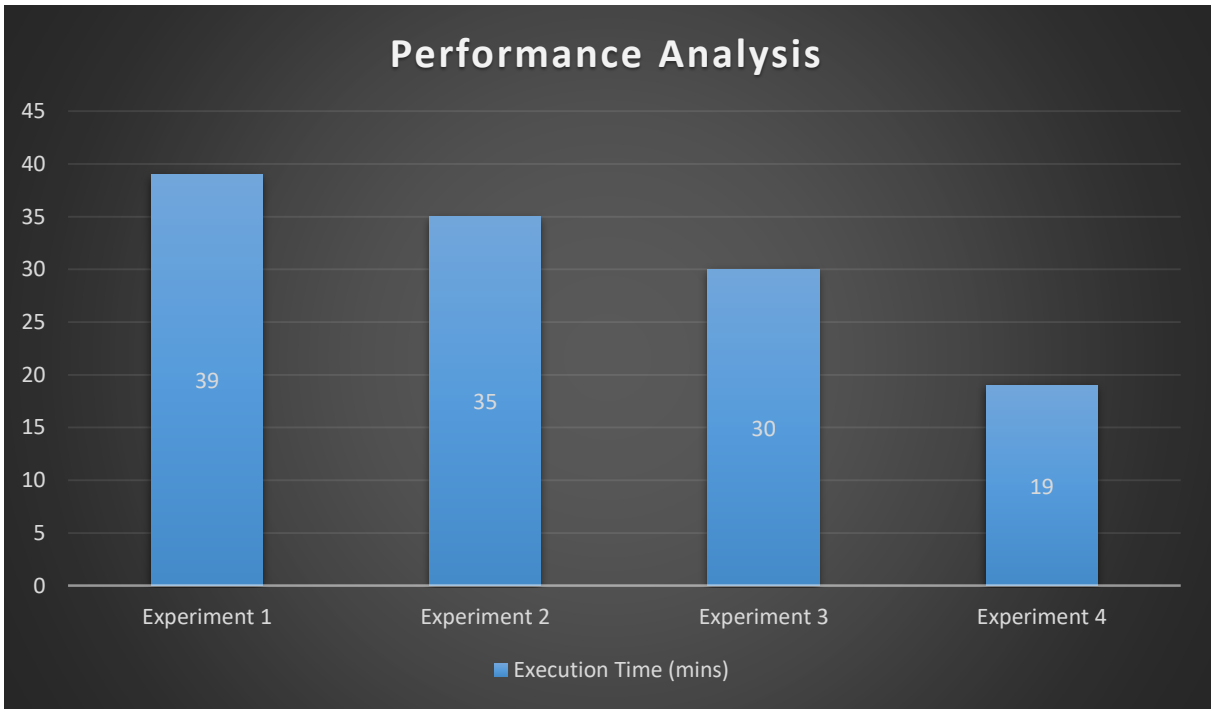


Figure 94 Semantic Classifier Performance Analysis

Conclusion

In this program, we have seen the importance of a good Machine Learning pipeline and how easy is to scale Machine Learning Models with Spark, benefiting from its powerful computational power.

- Machine Learning models require careful analysis of their parameters, performance and input data. Constructing the training set myself, I can say I now see the importance of carefully choosing high-quality data. High-quality data will later transform into high performance.

- Machine Learning models can be easily scaled with spark because of its straightforward and powerful API and libraries.

2.3.4.8. OPTIMISED SPARK PARAMETERS

In the following table, there are my proposed parameters for an optimal Natural Language Processing Model job execution using Spark. Please recall the Parameters Section to understand these settings better.

Parameter	Parameter Value	Explanation
Default Parallelism	30	The default value of 30 worked just fine.
Spark Serializer	None	N/A
Kryo Registration	None	N/A
Driver Memory	2GB	The driver doesn't require much memory.
Executor Memory	20GB	With that much caching going on and the large data size, having a value smaller than 20GB might be problematic
The number of executors	10	The maximum number of executors available.
Executor Cores	3	Maximum number of cores available
The number of shuffle partitions	3	Between 3 and 6, 3 proved to be giving slightly better results.
Extra archived files	The anaconda environment	Having multiple required dependencies, shipping them all as an environment is a must.
Log Level	ERROR	Avoiding to print a large number of INFO logs increased performance.
Jars Packages	Databricks XML package	Had to pass this package at run time to be able to parse the Wikipedia XMLs.

2.3.5. HISTORICAL SIGNIFICANCE SCORE PROGRAM

2.3.5.1. OVERVIEW

The last step in the pipeline is to now put everything together and get the final historical significance score. The process is pretty straightforward. I take the wiki pages with their predicted categories and their page rank score and apply a formula using the two to get the final historical significance scores. To do so I assign using a python dictionary to each category a category factor in the range of 0.85 to 1.20. These category factors are subjective and created by me after studying human occupations across history.

```
category_factors_dict = {
    "entertainment": 0.85,
    "sport": 0.90,
    "military": 0.95,
    "culture & arts": 1.00,
    "philosophy": 1.05,
    "religion": 1.10,
    "politics & leaders": 1.15,
    "science & technology": 1.20
}
```

Figure 95 Categories factors dictionary

To get the final scores I apply this simple formula:

$$Historical\ Score = Page\ Rank\ Score\ \times Category\ Factor$$

2.3.5.2. IMPLEMENTATION

After combining the results of the Page Rank and Natural Language Processing Model into a single dataframe, I multiply the two columns.

```
def _calculate_historical_significance(self):
    self._historical_significance_df = self._historical_significance_df \
        .withColumn("historical_significance_score",
                    col("category_factor") * col("rank")) \
        .select("page", "historical_significance_score") \
        .sort(col("historical_significance_score").desc())

    self._historical_significance_df \
        .write \
        .csv(path=self._output_path, mode="overwrite")
```

Figure 96 Historical Score Function

Scaling Results

To create more easily understandable results, I scale the calculate historical significance scores with a Min-Max Scaler. This is a type of normalization method which scales a distribution of values in the range of 0-1, keeping the proportions in the process. For this task, I used the SparkML Min Max scaler.

```
assembler = VectorAssembler(inputCols=['historical_significance_score'],
                            outputCol='historical_significance_score_vec')

scaler = MinMaxScaler(inputCol='historical_significance_score_vec',
                      outputCol='historical_significance_score_scaled')

pipeline = Pipeline(stages=[assembler, scaler])

self._historical_significance_df = pipeline \
    .fit(self._historical_significance_df) \
    .transform(self._historical_significance_df) \
    .withColumn("historical_significance_score", vector_to_values("historical_significance_score_scaled")) \
    .drop("historical_significance_score_vec", "historical_significance_score_scaled", "category_factor") \
    .orderBy('historical_significance_score', ascending=False)
```

Figure 97 Scaling Significance Scores

2.3.5.3. DEPLOYMENT

As input, there are two files with **1.2 million entries** each**.** The first one is the result of the Page Rank Program and is a CSV file with a list of all biographical Wikipedia page titles and their page rank score. The second one is the result of the Text Semantics Classifier Program and is a CSV with a list of all biographical Wikipedia page titles and their predicted semantic category.

The output is a sorted list of the most important historical personalities. In the table below you can see the top 10 most important personalities of all time. For the full results, please check the output/HistoricalSignificance.csv directory of my project.

Person	Page Rank	Semantic Category	Historical Significance Score
Elizabeth II	1016.94385	politics & leaders	1
Napoleon	902.2612	politics & leaders	0.887
George W. Bush	884.37286	politics & leaders	0.87
Barack Obama	1126.394	entertainment	0.819
Franklin D. Roosevelt	795.71655	politics & leaders	0.782
Jesus	774.4456	religion	0.728

Carl Linnaeus	795.9622	philosophy	0.715
Albert Einstein	695.306	science & technology	0.665
Adolf Hitler	675.9967	politics & leaders	0.665
Donald Trump	902.34924	entertainment	0.656

2.3.5.4. PERFORMANCE ANALYSIS

Initially, I thought that because of the huge number of records (page rank and semantic classifier combined to **2.4 million records**), I would need to deploy the Historical Significance Program on the cluster. However, this was not the case. Because of Spark's native optimizations, this program takes no longer than 10 seconds to run in local mode.

Running the program on the cluster would have taken longer than running it in local mode (around 20 seconds). This is because a lot of time is normally spent by Spark, HDFS and Yarn to just schedule the job and distribute the data to the executor nodes. For larger jobs that qualify for cluster deployment, this prolonged scheduling time is necessary to guarantee the program's efficiency and stability.

Even through the Historical Significance Program doesn't require a performance analysis or parameter tuning, we can however perform a useful analysis. That is that distributed computing is not always the solution. One might have the wrong impression that feeding any program into Spark and deploying it on a cluster is going to make it several times faster. As we just have seen, this is not true. To put it short, each specific programming problem needs its specific solution and only Big Data problems truly need Big Data tools.

2.3.6. DISCUSSION

As you can see, the results of the historical significance are more or less predictable. However, there are some surprises in this Historical Significance Ranking and some interesting trends.

The top ranking is dominated by politicians and leaders. This was to be expected because in this category we have individuals that shaped history the most at a given time from my point of view. Notice that I mentioned, “at a given time”. This is because, at a given point in time, politicians and leaders indeed have the most power and influence over the events of the world. However, if we look at the bigger picture, the lives of lesser-known individuals that were not popular during their lifetime might have a bigger resonance in the history of the world. The best example of this is Vincent Van Gogh, who was poor during his lifetime but it’s regarded as one of the most important impressionist painters. This bias towards politicians highlights a possible point of improvement for my project. In the Next Steps section, I will later describe some measures I have in mind in order to further expand my project and create a more accurate ranking.

The most historically significant person according to my ranking is Elisabeth II. Undeniably, the former Queen of the United Kingdom was one of the most important figures from our time. However, her huge popularity according to page rank can also explained by her recent passing and the media attention the event has gathered. This lead to a high increase in mentions of her majesty in other Wikipedia pages, artificially increasing her page rank score, and automatically her historical significance score. Therefore, arguably she is not the most historically significant individual.

Jesus, as expected, is in the top ten. Compared to the other individuals on the list, Jesus is the “eldest”, as he was born 2000 years ago. Consequentially, he should be the most disadvantaged by the page rank. However, he manages to be on the sixth position. A critical analysis of these facts suggests that his influence over the ages was so great and constant that he attained a different kind of historical signifance status. In other words, his influence is so engrained in the global culture that temporary popularity ups and downs become insignificant. Ergo, considering this historical period argument, the historical significance metric shows that Jesus Christ is in fact the most historically significant person.

2.4. SOFTWARE ENGINEERING

Throughout the project development, I closely followed SE principles regarding testing, version control, testing, design, and management. I introduced CI/CD into my development and professional software management with the use of Jira. This ensured that my implementation closely followed industry standards throughout the software development lifecycle.

2.4.1. WORKFLOW

1. Planned the new release using Jira, creating issues, bugs, checklists, and timelines.
2. Import Jira data into Gitlab to create corresponding issues.
3. Create branches corresponding to the Jira issues keys.
4. Work in the branches using TDD.
5. At each git push, the CI/CD pipeline is triggered executing app building, packaging, testing and check style verification.
6. When all the issues are integrated, I perform integration testing.
7. After all release issues are solved, I merge them to master, close the Jira issues, cut a new tag, and release my app in Jira as well.

2.4.2. JIRA

I used Jira to manage and plan my final year project. Jira is a suite of agile work management solutions that powers collaboration across all teams from concept to the customer. It is the de facto enterprise work management tool for all kinds of use cases, from requirements and test case management to agile software development. [13].

2.4.3. KANBAN BOARD

For the development of my project, I used Kanban because I was mostly focused on work visualisation and flow. The Kanban board is organised as follows:

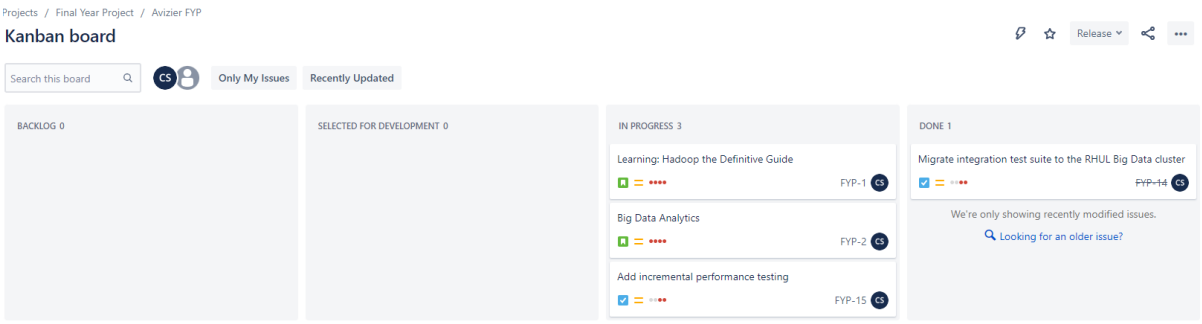


Figure 98 Kanban Board [13,16]

2.4.4. ISSUE OVERVIEW

Jira lets you customize issues in very extensive ways with the use of many plugins, but the most important customizations are:

- **Description**: general overview of the view.
- **Linked issues**: issues which relate to the current one.
- **Subtasks:** other issues which are subtasks of the main issue.
- **Checklist**: a list of tasks that need to be done.
- **Commit History**: Because I connected GitLab and Jira, all GitLab commits corresponding to the issue appear in Jira as well. The GitLab commits are links that redirect to the actual GitLab commit.
- **Priority:** how important is the issue?
- **Release:** release which corresponds to the current issue.
- **Development:** with Jira you can directly create a branch, commit, or release in GitLab
- **Status:** Backlog / Selected for development / Done

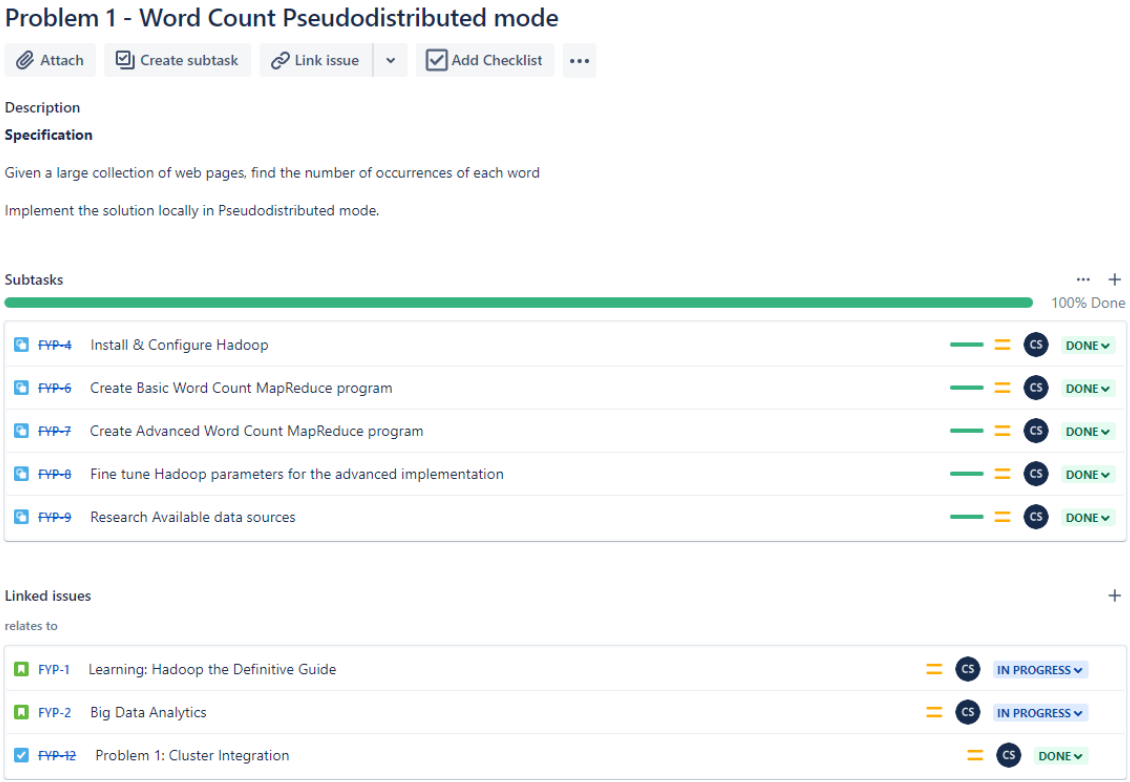


Figure 99 Jira Issue Overview [13, 16]

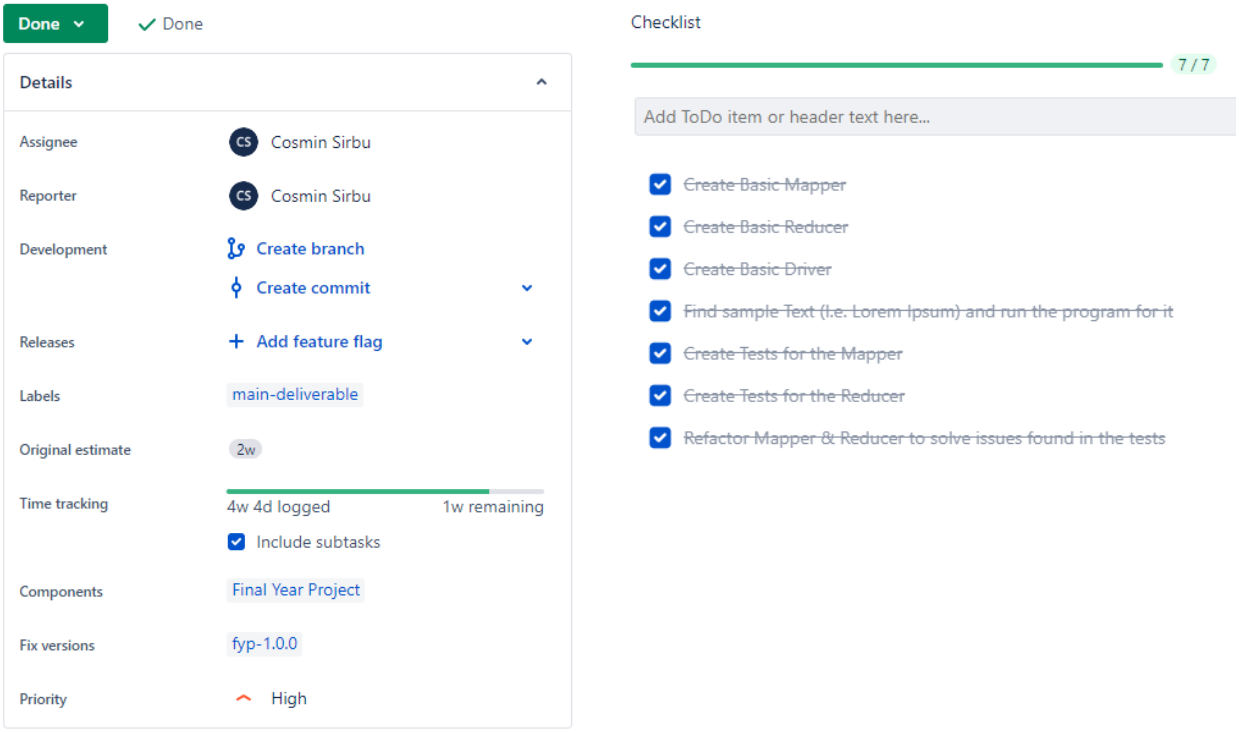


Figure 100 Jira Issue details [13,16]

Figure 101 Jira Issue Commit History [13,16]

2.4.5. RELEASES

In the releases view, I can easily see all past releases and the progress of my current ones. I can also see all assigned issues of a release and its completion status depending on the issue status.

Figure 102 Jira Releases [13,16]

2.4.6. GITLAB

I used GitLab with Git to manage my final year project software development. GitLab provides a central server that manages Git repositories and is used to simplify the administration tasks of many corporations worldwide. Like Jira, GitLab is the de facto enterprise solution for version control system management. GitLab uses Git, which is a version control system used to track changes in computer files. Git's primary purpose is to manage any changes made in one or more projects over a given period. [14,15]

In GitLab, after creating an issue in Jira, I created a branch with the Jira issue key as the name. While committing, if I specified the Jira issue key in the commit, Jira would pick it up as well and log it in its commit history view. After I finished developing the issues in Jira assigned to a fix version, I release it in GitLab which automatically releases it in Jira as well.

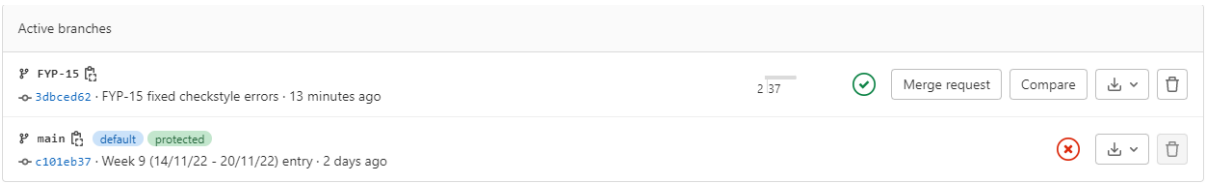


Figure 103 GitLab branches

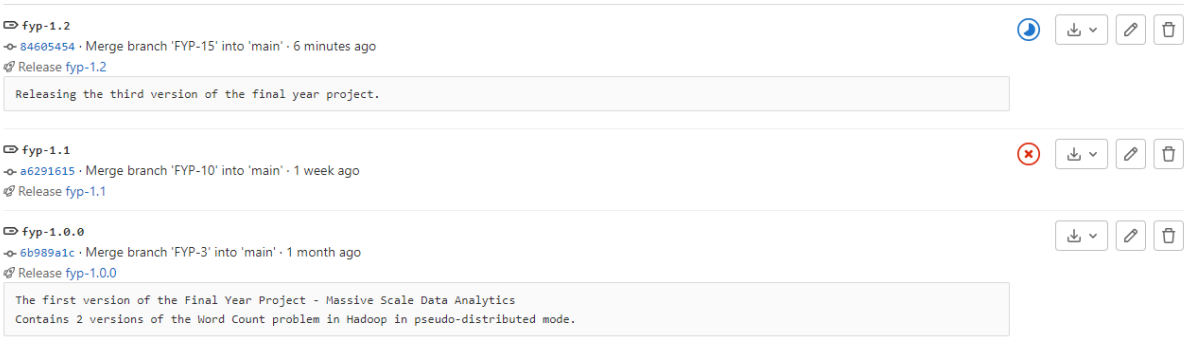


Figure 104 GitLab tags

2.4.7. CONTINOUS IMPLEMENTATION / CONTINUOUS DEVELOPMENT (CI/CD)

To guarantee constant software development standards, I set up CI/CD early in the project progression, with the help of GitLab CI/CD. CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts of CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD introduces automation and monitoring throughout the lifecycle of projects, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline". [17]

On my machine, I set up a GitLab runner which uses a Docker image to execute the pipeline runs. Then I plugged in this runner with GitLab so that at each push my pipeline handles

- Building
- Packaging
- Unit testing

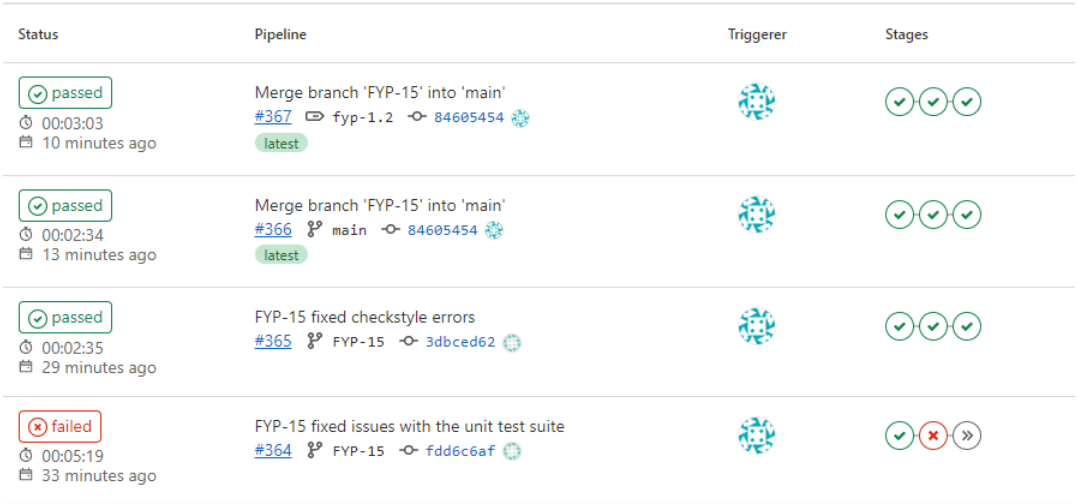


Figure 105 Pipeline Runs

2.4.8. TEST DRIVEN DEVELOPMENT (TDD)

I developed my programs using TDD, with the help of Junit and Mockito. TDD is a methodology whose aim is to iteratively build together code and tests one test case at a time. Therefore, the focus shifts from reaching the optimal solution in the first pass to gradually building it, ensuring that the software is properly built, and all (or most) edge cases are considered. JUnit is a unit-testing framework for Java and Mockito is a mocking framework that is used for unit testing of Java applications. [18]

My unit testing suite closely reflects this approach. For example, for developing the Word Count Mapper using TDD, I started by faking a real map (by just returning the value) and then progressively add more complex tests as I am refactoring and adding more features. For the Word Count map example, my tests start with checking the map functionality for a single word and finishing up with testing multiple words of different types of characters and languages. In the example below, using Mockito to mock the Context interface and Junit I verify the order in which output is printed by the write method.

```
@BeforeEach
public void setUp() {
  mapper = new WordMapper();
  context = mock(Context.class);
  input = new Text();
  one = new IntWritable( value: 1);
}
```

```
@Test
public void foreignWords() throws IOException, InterruptedException {
  input.set("ｋａｎｅｂｏケイトパウダレ");
  mapper.map(new LongWritable( value: 1L), input, context);
  InOrder inOrder = inOrder(context);
  inOrder.verify(context).write(new Text( string: "ｋａｎｅｂｏケイトパウダレ"), one);
  inOrder.verify(context).write(new Text( string: "التفاف"), one);
  inOrder.verify(context).write(new Text( string: "후기"), one);
}
```

Figure 106 Unit Tests

2.4.9. INTEGRATION TESTING

After finalising the development and unit testing of a new feature, I had to assure that all the implemented and tested components work correctly together. Therefore, I created an integration testing suite for each of the 3 problems in which the actual driver execution is tested. Later I also included the integration testing suite in the pipeline to ensure constant compliance with my requirements.

2.4.10 SOFTWARE DESIGN

2.4.10.1. FAÇADE PATTERN

The facade pattern abstracts a complex subprogram behind a simpler interface, like a facade. By hiding much of the complexity, it makes the subprogram easier to understand and use.

Both WordCountV2 and WordGrepMap have quite complex implementations which can be efficiently hidden under a Façade. I did this with the help of the 2 Façade classes: WordCountV2MapFacade & WordGrepMapFacade which abstract all their functionalities under their mapRecord() method.

2.4.10.2. REFACTORING

Polymorphism: WarcMap

Polymorphism is the ability of a program to perform in multiple ways, taking on more than one form.

WordCountV2MapFacade and WordGrepMapFacade are using the same Common Crawl data, the map is done similarly, with multiple methods of the same type. Therefore, I decided to refactor this functionality into its superclass: WarcMap, applying the polymorphism object-oriented programming principle. Then the Word Grep and Word Count V2 Map facades extend this base superclass and implement their functionalities by overriding existing methods or using the WarcMap ones.

Polymorphism: MapReduceDriver

All 3 problem drivers have very similar structures and configurations. Therefore, I have created a base superclass, MapReduceDriver that handles the basic functionalities of a MapReduce driver. Then the 3 specialised drivers extend this base superclass and implement their functionalities by overriding existing methods or using the MapReduceDriver ones.

2.4.10.3. UML CLASS DIAGRAMS

In the following figures, you can see all the UML diagrams of my programs. They show all the relationships between the classes and their scope.

WORD COUNT V1 UML DIAGRAM

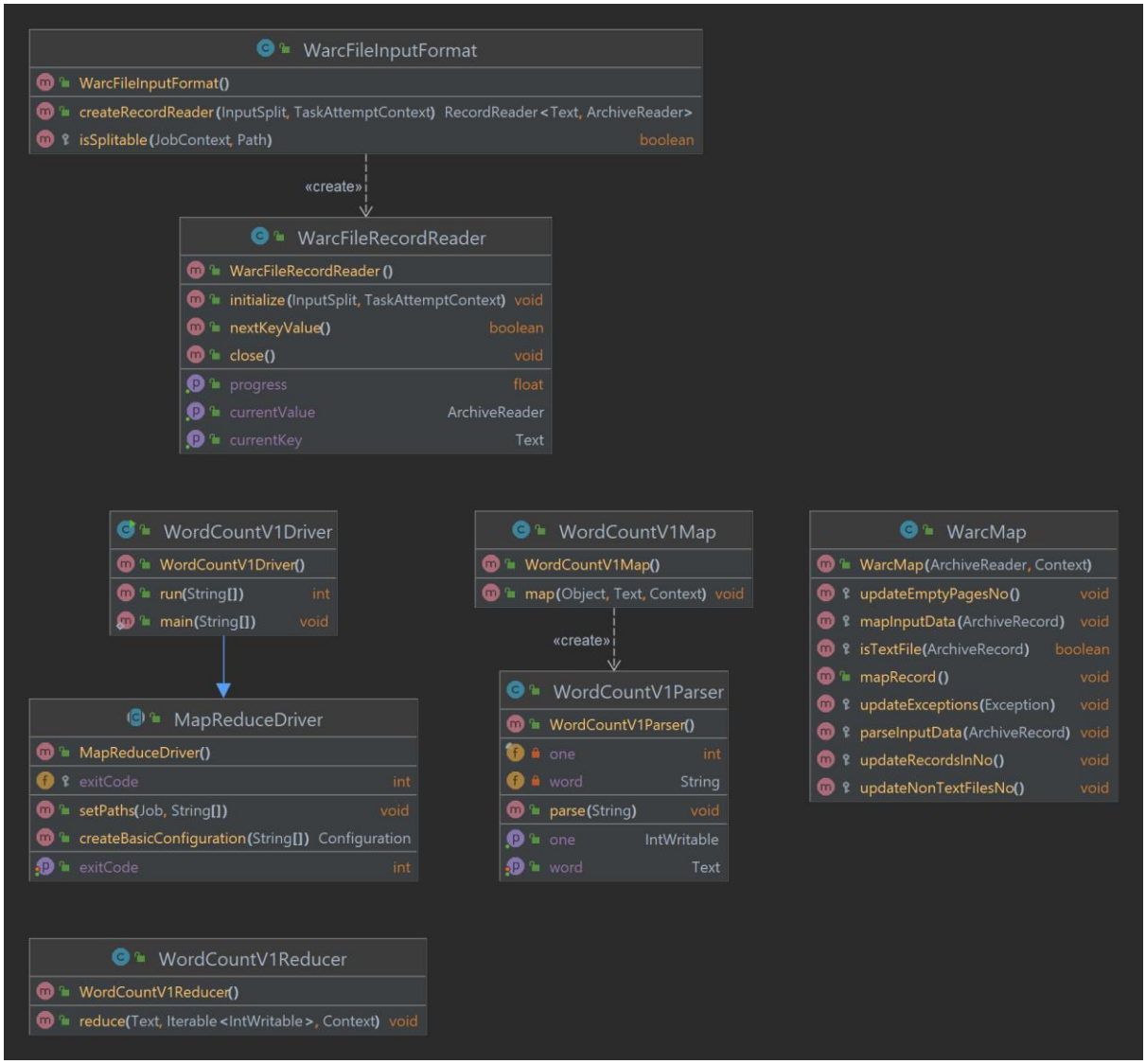


Figure 107 Word Count V1 UML Diagram

WORD COUNT V2 UML DIAGRAM

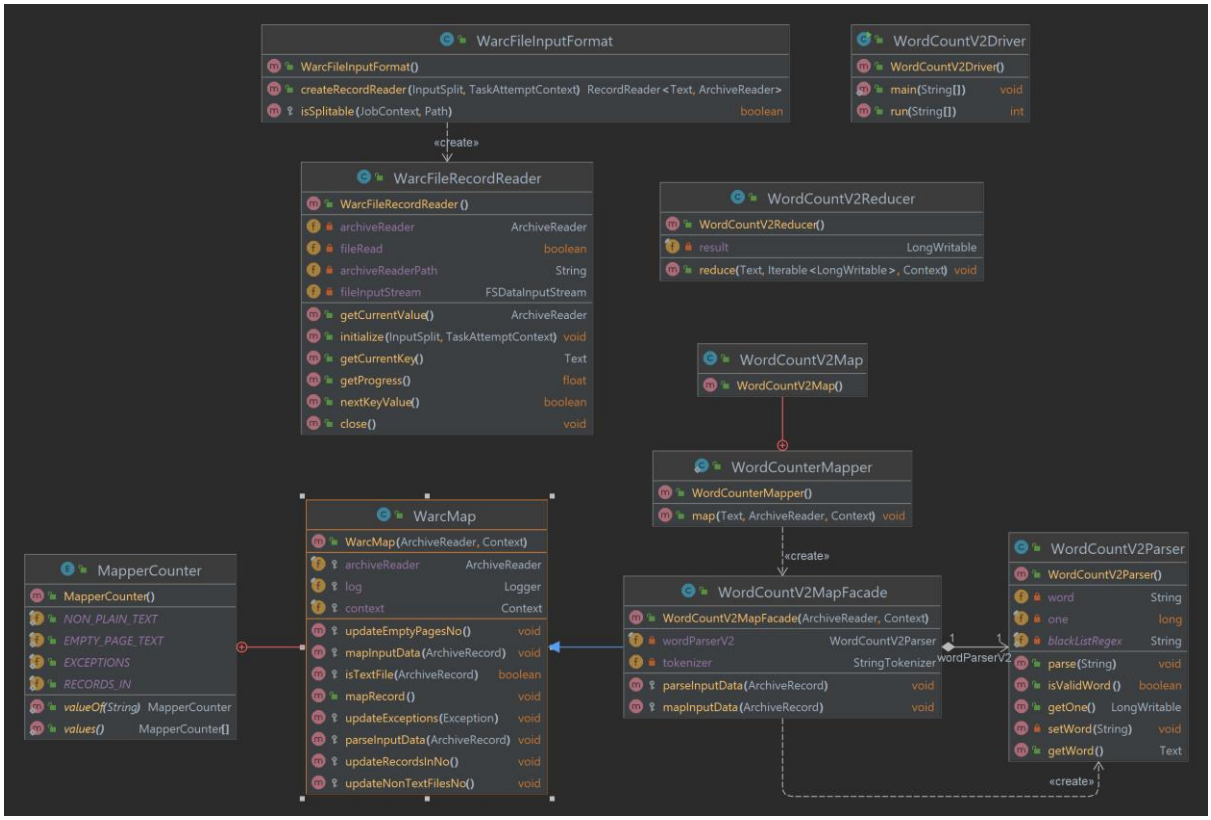


Figure 108 Word Count V2 UML Diagram

WORD GREP UML DIAGRAM

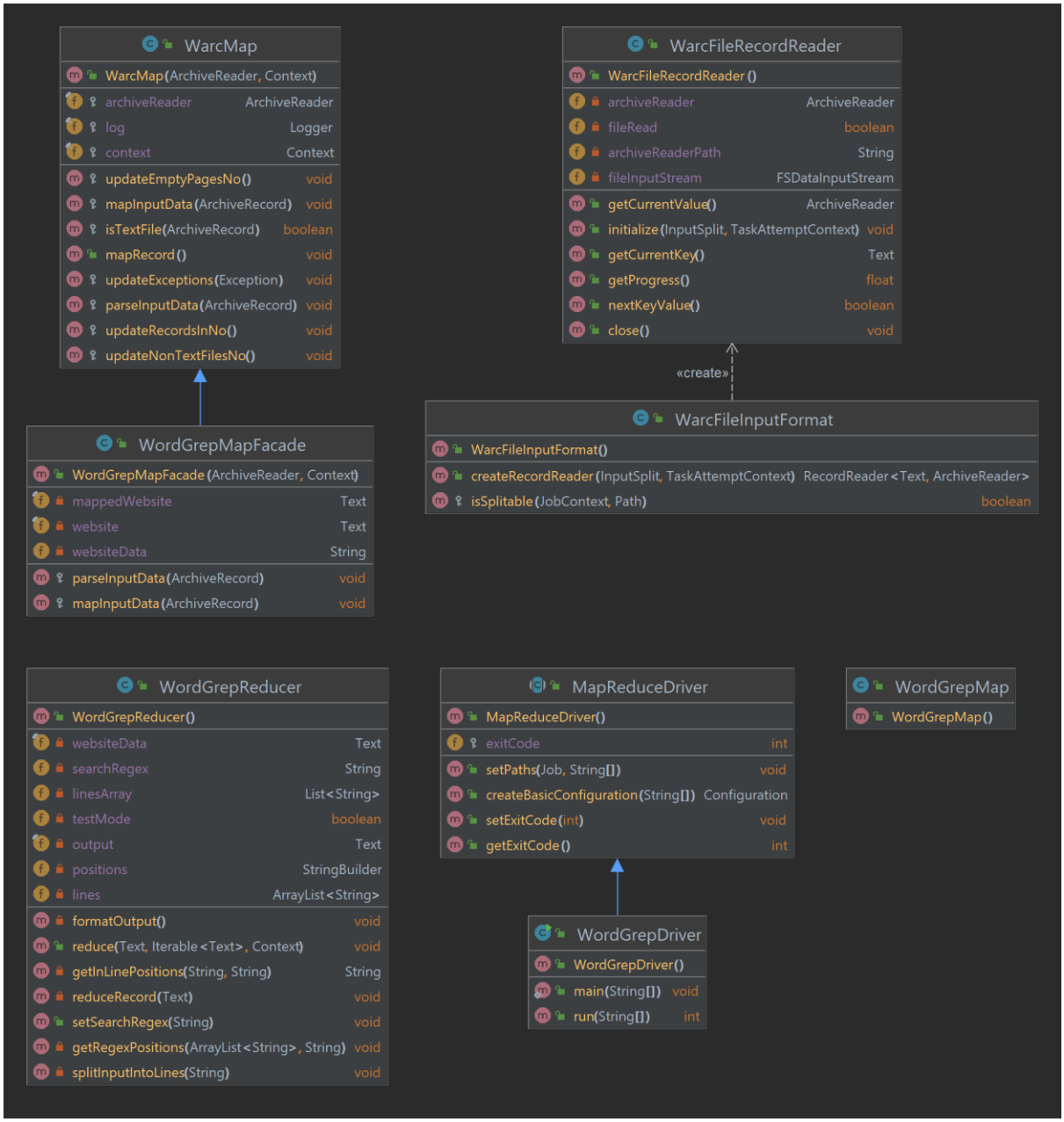


Figure 109 Word Grep UML Diagram

PAGE RANK UML DIAGRAM

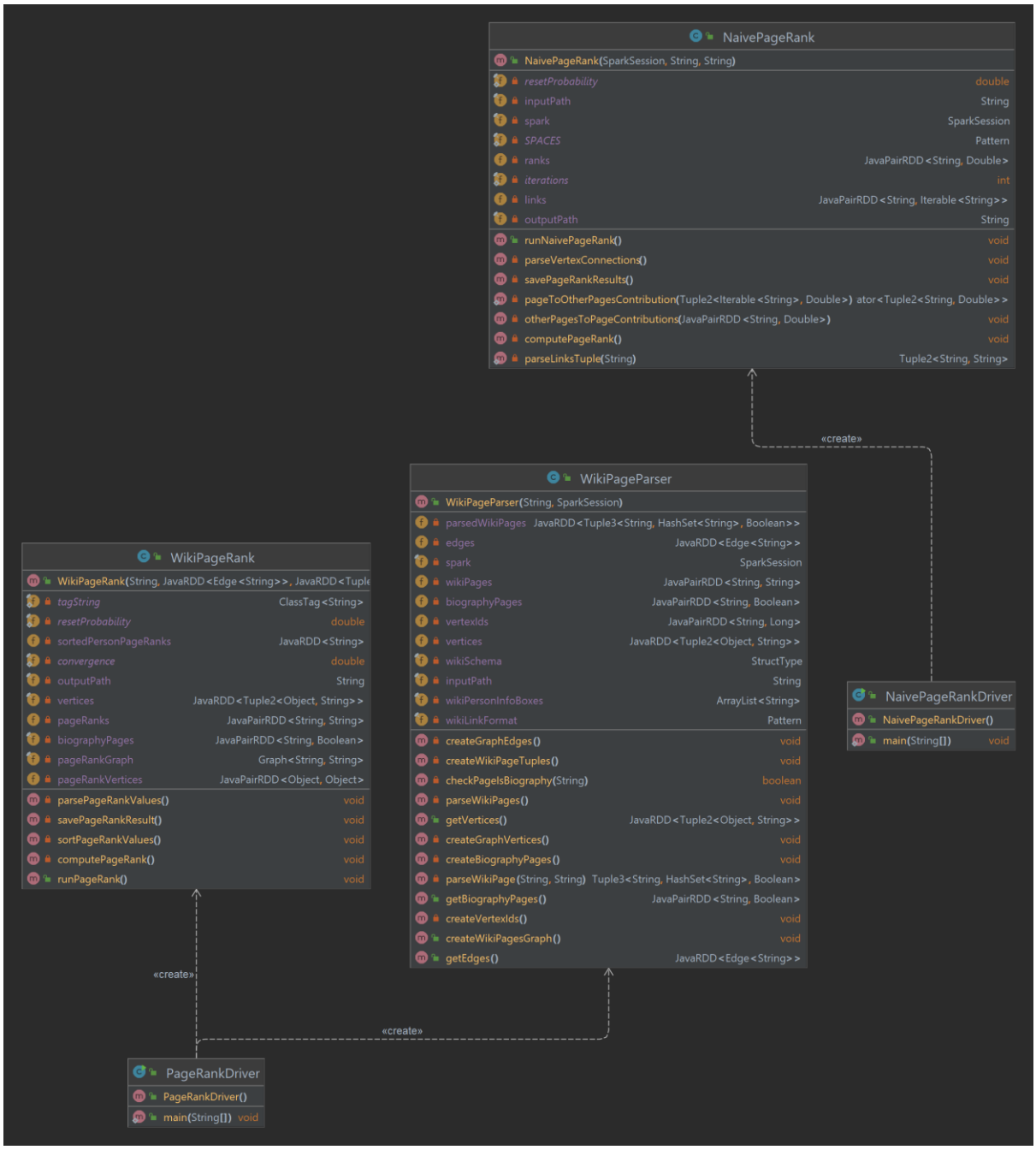


Figure 610 Page Rank UML Diagram

CATEGORY CLASSIFIER UML DIAGRAM

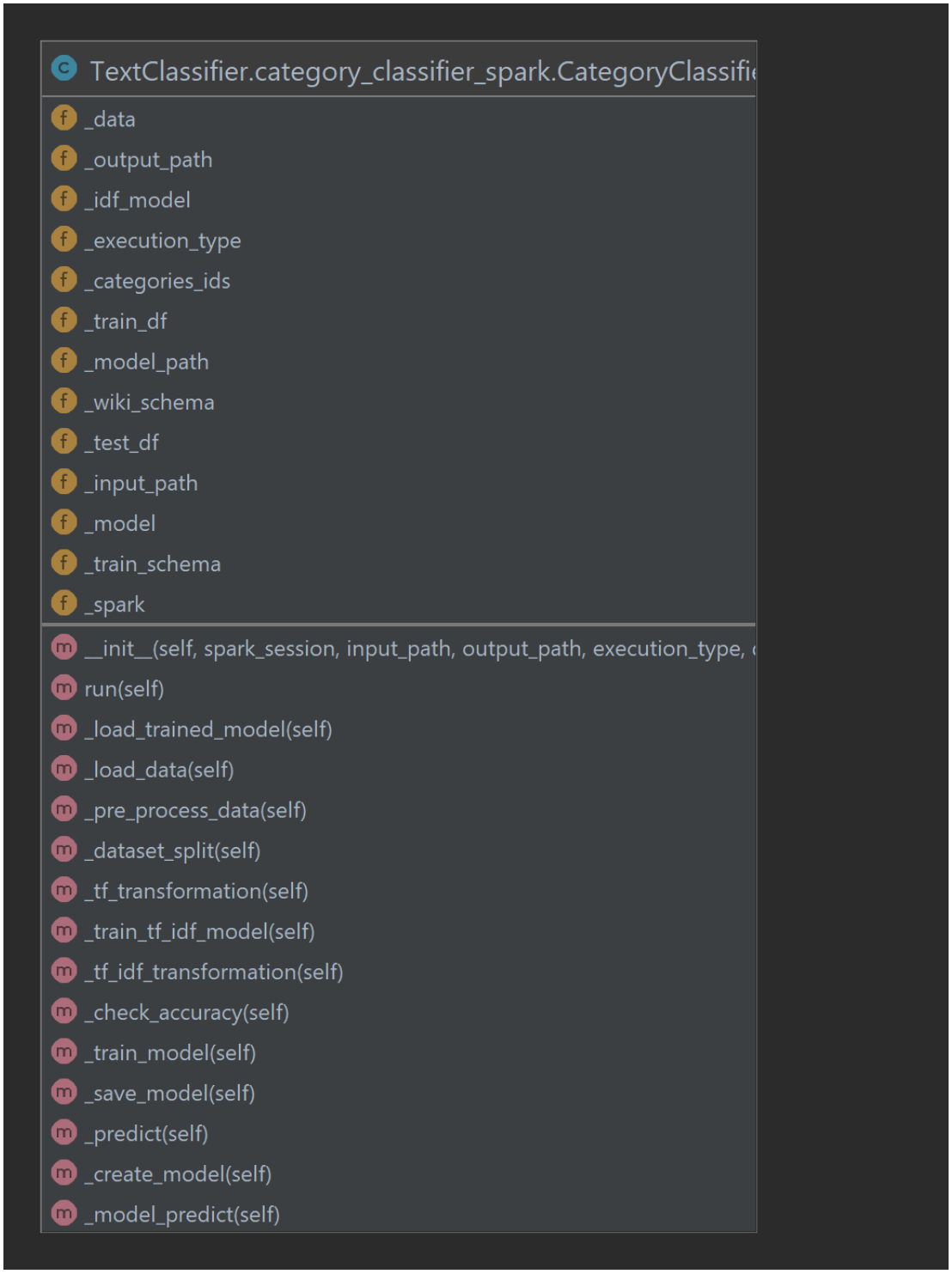


Figure 111 Semantic Category Classifier UML Diagram

2.4.10.4. TECHNICAL DECISION MAKING

In the following section, I am going to discuss my technical decision-making for the main tools and algorithms I used.

Jira

The reason why I chose Jira for my project management tool is that it's an enterprise-level software with a lot of great functionalities, as it was described above. Furthermore, I already had one year of professional experience using it thanks to my placement so the choice was natural.

GitLab

The reason why I choose GitLab for my project is that, like Jira, it is the de facto enterprise solution for version control system management and I had 1 year of professional experience with it.

Continuous Implementation/ Continuous Development

The reason why I chose to add CI/CD to my project was because of the stability and quality assurance I know it brings to the project. Despite being more difficult to set up and use for this type of project, I managed to do it nonetheless and automate testing and quality assurance for my final year project.

Test Driven Development

The reason why I chose to develop my code in a TDD fashion was that I know the benefits TDD brings to projects in the long term. TDD guarantees code quality through iterative development alongside testing and can save you from numerous problems in the future.

Hadoop

It wasn't necessary for me to also learn and use Hadoop for this project as I could have just started from the beginning to use Spark. The reason to do so was that I wanted to start learning about Big Data and being as close as possible to its underlying mechanisms. In Hadoop, you have to define your reducers and mappers and think thoroughly about the potential impact of your design decisions on the performance of your programs. If I just started learning Spark, the development would have been easier but I wouldn't have had the opportunity to properly understand the underpinnings of Big Data jobs.

Spark

Despite of my academic desire to also learn Hadoop, as was already discussed, Spark is superior to Hadoop MR in almost every way and is a truly modern solution to Big Data problems. Therefore, my choice to use Spark for my second part which contained complex and hard-to-implement programs was natural. Furthermore, creating a

machine learning model or a graph of the magnitude of the page would have also been either very hard or highly impractical with Hadoop.

Word Count

The reason why I chose to first implement this problem is that it is the “hello world” problem of Big Data, being simple but including both a map and a reduce task. I used this problem as a learning tool for Hadoop and performance analysis of Big Data problems. As it was presented, I created two alternatives for this problem, a version which just worked with simple plain text and one which worked well with large amounts of archived data.

Word Grep

The reason why I chose to implement this problem is that it could be done in the same MapReduce paradigm as Word Count but it was more complex, allowing me to better think and analyse performance requirements of MapReduce problems.

Page Rank

The reason I chose this algorithm to attribute scores to Wikipedia biographical pages was that it is a very well-known and behaved algorithm which works very well in this kind of graph structure like Wikipedia’s connections graph. As it was presented, I created two alternatives for this problem, a version from scratch which ran for a certain amount of iterations and an improved version which ran until a convergence point was reached.

Natural Language Processing Model

The reason I chose to create a model to predict semantic categories of biographic Wikipedia articles is that it is the best and easiest solution to the problem of attributing sematic meaning to texts. The alternative which was already discussed was to create a Knowledge Base from the Wikipedia Category Tree. However, we have seen this was highly impractical because of the non-hierarchical nature of the category graph.

3. ISSUES

In the following section, I am going to discuss the professional issues which helped me shape my project's process as well as present some main technical issues I encountered throughout the project development.

3.1. PROFESSIONAL ISSUES

This project has been developed in the area of Big Data and Machine Learning, two fields which are increasing in popularity every day. With the raise of tools like Chat GPT, the importance of addressing the concerns for creating proper standards and regulations for artificial intelligence and big data has become increasingly important. With that in mind, I have developed my project with the following professional issues in mind:

Privacy and data protection

Because my project handled hundreds of GB of data, one issue that I followed is privacy and data protection. Even through my project was not handling sensitive data, I am well aware of the potential implications of not handling data responsively and securely. In my case, I always made sure to store the data I used securely on the HDFS. However, in the case of big institutions and corporations which may handle personal or sensitive data, this aspect is even more crucial.

A good case study for this issue is the General Data Protection Regulation (GDPR) act which is a data protection law that came into effect for all EU countries in 2018 [33]. This law applies to all companies in the EU that process personal data [33]. At the heart of GDPR lay the following principles: transparency, fairness, lawfulness, purpose limitation, data minimization, accuracy, storage limitation, integrity and confidentiality [33]. GDPR besides being a law should be a standard for companies to follow while handling data. The only problem is that amount of data we generate and consume is increasing exponentially. Therefore, for companies and institutions, the big data privacy problem automatically translates into increasing costs and complicated internal regulations that have to be followed and constantly updated. To tackle this issue, companies and institutions should take this issue more seriously and create better long-term strategies for this.

Bias and Fairness

With the raise in popularity of many Natural Language Processing tools, the need for ensuring that they are not biased and they are fair in their interactions with the users is a must [34]. As machine learning models become more sophisticated, people start to view them more like real entities that can have views and can be discriminatory or offensive [34]. Therefore, there needs to be some kind of regulation and protective measures against this kind of issue in the development of machine learning models [34].

Being trained on large amounts of data which may contain innate biases, the machine learning models will end up with the same kind of biases themselves since they are just a reflection of the data they are trained on [34]. Regarding this issue, organizations and institutions must be very careful with the contents of the data they feed the models in the first place and constantly monitor the activity of the models after their deployment and constantly adjust them in case any bias slips through the training process [34].

This issue was highly important to me when creating my natural language processing model for text semantics classification. This is because I wanted my model to have a good generalization on the entire Wikipedia and accurately predict the occupational category of any person, despite their geographic, historic or ethnic

background. To achieve this good generalization and remove my model biases, I extensively researched the sub-genres and aspects of each of the categories from my dataset and created a list of topics of interest that aimed to be as broad as possible. Therefore, the entries for each category were selected from a vast domain of backgrounds, cultures and historical periods. By doing this, not only was I able to remove biases from my model, but also highly increase the accuracy.

Transparency

As was mentioned before, machine learning models are just a reflection of the data they are trained on. Their training process involves the bottom-up approach which was earlier discussed in this report. This means a single machine learning model type can be used for a large variety of problems. All it needs is data. This is a great advantage but also at the same time can be a curse. This is because this approach can transform the model into a black box [35]. Not even specialists can know in most cases why certain results appear or how the decision-making process is made [35]. Therefore, if a model does something wrong, it is almost impossible to backtrack the source of the problem in the algorithm itself as you would in a normal top-down approach [35].

This lack of transparency can be problematic, especially in areas like healthcare of finance where decisions of the models can have significant implications [35]. I view this as more of a human fault than the model's fault. Because we discussed biases, we humans have our own biological biases. These biases apply to machine learning models as well. Why? Because a human will always appear more trustworthy than a program. Even if the model has very high accuracy, better even than the human, if asked, the majority of people would want to have their financial or medical decisions to be assessed by a fellow human. How to get past this bias then? The best solution is to treat machine learning models as assistants [35]. The human specialists will consult with it but the final decision will belong to the human as well. In this way, we take the best from both worlds [35].

I was faced with this transparency problem as well. In many cases, even after I already thought I made my data general enough, there were still some issues with my model's predictions and I couldn't figure out why. Furthermore, I couldn't go and analyse the learning process of my model to pin point the error. Therefore, what I had to tediously do was to carefully analyse my training data and look for any samples which didn't really belong to that category or whose semantic meaning was ambiguous, remove them and then re-train the model and re-access the model's behaviour on real-world data. Going back to the bigger picture, I was able to do this tedious work because the volume of data I had was large but manageable. However, this is not the case for big corporations and institutions which are using immense volumes of data. Therefore, they have to be careful with their data quality from the beginning to avoid hard-to-solve transparency issues in the future.

Conclusion

To conclude, assessing professional issues have shaped my project from the start to the end. This enabled me to not only develop my code better but critically analyse my project's process and the technologies behind it better. I believe this crucial analysis has made me a better software engineer and will prove very useful in my professional career.

3.2. TECHNICAL ISSUES

In the development of this project, I encountered many technical issues. In this section, I will briefly illustrate the main ones I encountered and how I solved them.

Setting Up Hadoop

To properly set up my development environment on my laptop, I couldn't just run my MapReduce programs as a local job. Therefore, I had to install and set-up Hadoop on my laptop. This was highly important because, at the beginning of the project, I didn't even know of the existence of the Royal Holloway Cluster. To set-up Hadoop, I decided to try two different approaches.

My first approach was to set it up locally in pseudo-distributed mode. As it turned out, this was a long and complicated process and one which is very poorly documented by Apache. The main issue I encountered here was that after downloading Hadoop, the bin folder, the most important folder in the app and you have to delete the downloaded one and download the correct one separately. This seemed like a very bad design problem from Apache, which combined with their lack of tutorials and documentation on the issue, made the problem even worse. However, after extensive trial and error, I managed to make the cluster work and used it for development for the rest of the project.

My second approach which turned out to be even more problematic than the first was to emulate a cluster using Cloudera. Cloudera is an enterprise-grade tool which can be used to virtualise the resources of your machine to create a virtual Hadoop cluster on your machine. This, in theory, sounds great as this means you will have the benefits of parallelism without actually having the infrastructure for it. The downside of the Cloudera solution is that is requires a large amount of RAM to function properly, 8GB minimum to be exact. On my development laptop, I had 8GB in total so it can easily be understood why this was problematic. After I tried to open the Cluster, my laptop quickly crashed. To avoid this issue, I decreased the amount of RAM available to the Cloudera cluster but this then lead to poor performance.

In the end, I decided to just the pseudo-distributed cluster for development and the Royal Holloway Big Data cluster for deployment, testing and parameter tuning.

Spark RDD Stack Overflow errors.

For the Naïve Page Rank model, I had to re-accumulate iteratively the page ranks and store my updated computations in an RDD. This type of approach combined with RDD's design caused an interesting but well-known bug, Stack Overflow errors. For RDDs, Spark keeps in memory all previous transformations done on the RDD. This is beneficial because this approach allows Spark to quickly identify and re-compute RDD operations if a certain task or executor node failed. However, when you update an RDD 100 times let's say as you have to calculate the page rank, then all of this extra memory of all past operations will become too large and potentially cause a Stack Overflow Error. In other words, RDDs will have an increasingly large lineage, that will cause Stack Overflow errors at large iterations sizes. This was a hard problem to diagnose but the solution was quite simple. At a certain amount of iterations (for example 10 in my case) you have to save the current RDD computations by calling checkpoint().

```
if (i % 10 == 0) {
  contributions.checkpoint();
}
```

Figure 7 Avoiding Stack Overflow errors

Cluster Compatibility Errors

The Royal Holloway Big Data cluster uses an old version of Spark and Scala, Spark 2.3 and Scala 11. This has caused a couple of computability issues with the software I was using.

Firstly, one interesting software I couldn't use at all was Spark Pandas. This is a new library added in Spark 3.3. which is Pandas plugin for Spark. Developers in this was can easily move their machine learning models which usually use Pandas to Spark. The Spark Pandas API is the same as normal Pandas, Spark creating another level of abstraction over Spark Dataframes, without dropping performance. After I moved my model from python to Spark on my development environment, I tried to test it on the cluster, but without any luck, because Pandas Spark API which I was using wasn't available. Therefore, I had to convert my model again, this time using Spark Dataframes.

Secondly, I had issues with multiple other maven dependencies and I had to downgrade back to a compatible version of Spark and Scala. This made my deployment process way more complicated than it should have been. Furthermore, Spark doesn't integrate with containerization engines like Docker since HDFS can only work with its own Spark API and can't use Docker containers.

Creating a Machine Learning Training Set

As mentioned in the professional issues part, creating a good Machine Learning dataset can be problematic. This is because you need high-quality data from the start. Otherwise, because of the black-box nature of machine learning models, it is quite impossible to backtrack poor accuracy to specific input samples. When this issue happened to me I discovered I had a poor accuracy for my model of around 80%, I re-did parts of my training set, this time trying to create my input sample in a normalised form from better-curated sources. Just by adjusting this step, my accuracy jumped from 80% to 95% (on the testing set) and managed to also achieve a very good generalization on unseen Wikipedia articles as well.

Cluster Problems

Because I had to work with very large amounts of data, I was quite reliant on the Royal Holloway Big Data cluster. Therefore, any kind if issues arising from the cluster side greatly impacted my progress. The cluster has been mostly stable but towards the end of the term I encountered a couple of problems. Once, for a couple of days, the cluster would not schedule any pending jobs. However, this was easily resolved by the IT team by just restarting the cluster. The second problem was much more severe and hard to diagnose as well. Because of a hardware issue, for a large input size, the cluster would crash. Restarting the cluster would not have any effect. This problem took 3 days to properly be diagnosed and solved by the IT team by solving the underlying hardware problems. However, this slowed down my progress a little bit as the timing was not great as well, as this happened towards the end of the term.

4. CONCLUSION

Big data undeniably play an increasingly important role in our everyday life. The most effective way to tackle the Big Data issue is through parallelization with the help of cluster computing and frameworks like Hadoop and Spark. In this report, we have seen what Big Data and Machine Learning frameworks are, how they work and their implementation in some complex programs.

For each of the problems we have observed the importance of cluster computing, parameter engineering and performance analysis and drew useful conclusions about the strengths and weaknesses of Hadoop and Spark as well as issues and solutions to Big Data problems.

From the Word Count problem, we saw that Hadoop is ideal for batch processing and easily scalable but costly to maintain and operate for enterprise requirements. This is because, despite of the ease with which Hadoop architecture can be scaled up or down, it can guarantee a good program performance only with many Hardware resources. The difficulty of this part was to properly put all the learned theory into practice and deploy my first program into a distributed environment.

From the Word Grep problem, we saw the importance of carefully analysing edge cases in Big Data. This is because some problems have highly variable performance depending on the job parameters. Therefore, edge cases must be thoroughly handled so the programs don't get lost in a large amount of data. The difficulty of this part was to further expand what I learned from developing the Word Count problem and analyse the performance of my programs efficiently to properly fine-tune the parameters of the problem.

From the Page Rank program, we have seen the importance of caching, serialization and code optimizations in the execution time of Spark programs. We dealt with massive amounts of data (the entire Wikipedia). We managed to parse all this data, construct the connections graph on the entire Wikipedia from it and run a powerful algorithm, Page Rank to obtain some interesting results. The difficulty of this part was understanding how to handle massive amounts of data efficiently given the problem's high computational requirements and the massive data input.

From the Natural Language Processing Model, we have seen the importance of carefully designing and accessing all steps in the pipeline. We understood the importance of gathering qualitative data to construct a good training dataset and saw the benefits of fine-tuning machine learning meta-parameters through grid search to obtain the best accuracy. Finally, we saw the power and scalability of Spark by implementing the model with SparkML and scaling it to be able to predict the categories for every biographical page on Wikipedia. The difficulty of this part was creating an accurate and efficient machine learning model, especially the crucial and complicated task of creating a good training dataset for the model.

Overall the project was a success. I have achieved all the goals I set up at the start of the project and managed to finish an ambitious project that used the entire Wikipedia data source and create an interesting ranking of the most important historical personalities in history. I aim to apply all the learned lessons about Big Data, Machine Learning and software engineering in my professional career.

PROJECT NEXT STEPS

The historical signifance program worked very well but as we have seen, it was not perfect. It was a bit biased towards politicians and people from more modern times. So how to improve the program to have a better, more well-rounded approach? These are some proposal I have for a future extension of this project.

Firstly, many individuals don't strictly belong to one semantic category. This is because many individuals have multiple occupations. I designed my semantic categories with this in mind, making categories as different semantically as possible. However, some individuals, like Donald Trump belong to multiple semantic categories like politics & leaders and entertainment. This is why his predicted semantic category "entertainment" is understandable, but doesn't reflect the whole picture. A solution for this is to implement multi-class regression instead of classification, in which a text would belong to multiple semantic categories. Then a person's category score would be the average of all its predicted category scores.

Secondly, my category factors are created corresponding to my subjective ranking after I researched human occupations throughout history. However, this might not be the most accurate representation of them. To achieve a fairer human occupation ranking, an extensive sociological and historical study of human occupations throughout history would be needed. After this study is performed, I might have to add or remove categories and re-make the category factor ranking.

Thirdly, only page rank and occupation category metrics might not be relevant enough to create a fair ranking. To account for this, I could add more metrics to the historical significance calculation such as the length of the article [12], the frequency of the person appearing in the news [12], and user activity on each article [12]. I could also add a historical period discounting metric that adjusts the score of the individuals based on the historical period they lived. Automatically, more modern historical figures would be more popular according to page rank, this is why this time metric is important. All these metrics put together will form a more well-rounded picture of the historical significance of the individuals.

To conclude, I believe all these correctional measures put together will form a more accurate picture of the historical significance ranking and will help me develop the project even further.

BIBLIOGRAPHY

[1] White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012

I finished the book and did all the examples which gave an in-depth understanding of Hadoop, its usage & internal structure, MapReduce applications and cluster configuration.

[2] Ankam, Venkat. Big data analytics. Packt Publishing Ltd, 2016

I read chapters 1-6 and did all the examples in the book which gave a good understanding of the differences between Hadoop and Spark, their pros and cons and a deep dive into the Spark ecosystem.

[3] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), 2010.

Read this research paper to better understand Spark and its uniqueness compared to other tools.

[4] Russom, Philip. "Big data analytics." TDWI best practices report, fourth quarter 19.4, 2011

Read this report to better understand TDWI and its role within Big Data Analytics

[5] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008)

Thoroughly read about the MapReduce framework and its importance to big data processing.

[6] "Hadoop Installation for Windows." Brain Mentors, brain-mentors.com/hadoopinstallation/.

Used this guide to install Hadoop for pseudo-distributed mode on my machine.

[7] "Common Crawl." Common Crawl, commoncrawl.org/.

A very useful site with a rich repo of petabytes of data scraped from the web all the way back to 2012. I used this repository for data gathering for part one.

[8] "Common Crawl Warc Examples." GitHub, github.com/commoncrawl/cc-warc-examples.

A repo provided by Common Crawl (8) that uses some of the scrapped web data and uses Spark and Hadoop to solve a couple of problems like Word Count. I used it to help me develop my programs.

[9] "How Big is the internet?" ipxo, ipxo.com/blog/how-big-is-the-internet/.

An interesting overview of the size and growth of the internet.

[10]"What is DFS (Distributed File System)?" geeksforgeeks, geeksforgeeks.org/what-is-dfsdistributed-file-system/.

Good general overview of distributed file systems.

[11] Shvachko, Konstantin, et al. "The Hadoop distributed file system.", 2010

Useful and detailed information on the HDFS.

[12] Skiena, Steven, and Charles B. Ward. Who's bigger? Where historical figures rank. Cambridge University Press, 2014.

The book I got my inspiration for my second term program. I will use it as a guide while developing my programs in the second term.

[13] "Atlassian." Atlassian, www.atlassian.com/.

The work management tool, Jira by Atlassian which I used for my project.

[14] "About Gitlab." Gitlab, about.gitlab.com/.

The version control system and Git server I used for my project.

[15] "What is GitLab and How to Use It?" simplilearn, simplilearn.com/tutorials/git-tutorial/what-is-Gitlab.

Good overview of Gitlab and its usage.

[16]"Final Year Project board." Jira, cosminsirbu-finalyearproject.atlassian.net/jira/software/c/projects/FYP/boards/1.

My Jira board I used to manage the project.

[17] "What is CI/CD?" Redhat, redhat.com/en/topics/devops/what-is-ci-cd.

This a good article offering an overview of CI/CD and its importance in software development.

[18] "Test-driven development." IBM, ibm.com/garage/method/practices/code/practice_test_driven_development/.

A complete guide by IBM to TDD.

[19] Page, Lawrence, et al. The PageRank citation ranking: Bringing order to the web. Stanford InfoLab, 1999.

The original Page Rank paper by Google co-founder Page Lawrence. Short, but very condensed and easy to understand despite the high level of technicality.

[20] Hirschberg, Julia, and Christopher D. Manning. "Advances in natural language processing." Science 349.6245

Comprehensive guide to natural language processing and approaches.

[21] Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." Science 349.6245

The main source of knowledge for my machine learning models and approach. Learned about training, validating, data leaking, overfitting, underfitting and other key concepts in machine learning.

[22] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20 (1995)

The original paper from the creator of Support Vector Machines (SVM)s, Vladimir Vapnik.

[23] Webster, Jonathan J., and Chunyu Kit. "Tokenization as the initial phase in NLP." COLING 1992 volume 4: The 14th international conference on computational linguistics. 1992.

Good read about the importance of tokenization if natural language processing, how it's done and its benefits.

[24] Plisson, Joël, Nada Lavrac, and Dunja Mladenic. "A rule-based approach to word lemmatization." Proceedings of IS. Vol. 3. 2004.

An instructive guide about lemmatization, its role and importance in natural language processing pipelines.

[25] Trstenjak, Bruno, Sasa Mikac, and Dzenana Donko. "KNN with TF-IDF based framework for text categorization." Procedia Engineering 69 (2014): 1356-1364.

Applied read about the use of a term frequency-inverse density frequency vector with a K Nearest Neighbours machine learning model for text categorization. Used this book as a guide for my semantic categorization, despite using a different model and framework.

[26] RelcyEngineering. "Using Wikipedia Category Graph for Semantic Understanding." Medium, Medium, 1 Mar. 2016, https://medium.com/@RelcyEngineering/using-wikipedia-category-graph-for-semantic-understanding-5638c9897f8b.

An article about how Wikipedia can be used for deriving semantic understanding in blocks of text.

[27] Andreas C. Mueller and Sarah Guido. Introduction to Machine Learning with Python. O'Reilly Media

Comprehensive read about Machine Learning that completed my knowledge given by [21].

[28] "Deep Learning System Improves Breast Cancer Detection." NVIDIA Technical Blog, 21 Aug. 2022, https://developer.nvidia.com/blog/deep-learning-system-improves-breast-cancer-detection/.

Sample resource used for exemplifying machine learning concepts.

[29] “Wikipedia”, Wikipedia, https://www.wikipedia.org/

The main Wikipedia page. Used Wikipedia extensively in the second part of the project.

[30] “3d Order Tensor”, Research Gate, https://www.researchgate.net/figure/3rd-order-Tensor-representation-of-a-color-image_fig2_307091456

Sample resource used for exemplifying tensors.

[31] Silva, Catarina, and Bernardete Ribeiro. "The importance of stop word removal on recall values in text categorization." Proceedings of the International Joint Conference on Neural Networks, 2003. Vol. 3. IEEE, 2003.

Very useful read about the importance of the stop words removal step in a natural language processing pipeline.

[32] “2022 English Wikipedia”, Internet Archives, https://archive.org/details/enwiki-20220501

My Wikipedia Internet Archives source for downloading the entire English Wikipedia.

[33] “Official Legal Text.” General Data Protection Regulation (GDPR), 27 Sept. 2022, https://gdpr-info.eu/.

The Official GDPR article I studied for my professional issues section.

[34] Mehrabi, Ninareh, et al. "A survey on bias and fairness in machine learning." ACM Computing Surveys (CSUR) 54.6 (2021): 1-35.

Study about the biases and fairness in machine learning that helped me in my professional issues section.

[35] Vollmer, Sebastian, et al. "Machine learning and artificial intelligence research for patient benefit: 20 critical questions on transparency, replicability, ethics, and effectiveness." bmj 368 (2020).

Article about the importance of transparency in machine learning that helped me in professional issues section.

APPENDIX

DEPLOYMENT VIDEO

Please follow the following link to see my deployment video (with the sound on) on YouTube.

https://youtu.be/ZoWg4LrowgA

DIARY

Week	Tasks Performed	Problems Encountered	Next Steps
Week 1 (19/09/22 - 25/09/22)	1. Read Chapters 5-8 of Hadoop the Definitive Guide 2. Understood the anatomy of MapReduce Jobs and how they work within Hadoop 3. Had a look at configuration tuning for different types of requirements 4. Understood different file types of formats and how to manipulate them in Hadoop 5. Performed testing on different MapReduce classes using Junit & Mockito 6. Had a look at different MapReduce features 7. Started writing the project plan	1. Using Junit & Mockito for mocking objects 2. Understanding advanced Hadoop configuration settings 3. Documenting for writing the Project Plan	1. Further research on Hadoop, Spark, and Big Data Analytics 2. Finish the first draft of the project plan 3. Set up Jira and GitLab and hook them together
Week 2 (26/09/22 - 02/10/22)	1. Read Chapters 9-10 of Hadoop the Definitive Guide 2. Read Chapters 1-3 of Big Data Analytics	1. Finding a good data source for web scraping data	1. Research different data sources for problems 1 and 2

	3. Gained insight into the Spark ecosystem 4. Understood the use of Big Data Analytics tools compared to the classic ETL approach 5. Understood differences between Hadoop and Spark and the advantages of each 6. Finished writing the first draft of the project plan 7. Researched repositories and API-s to use for web scraping for problems 1 & 2 8. Set up Jira & GitLab and hooked them together 9. Started working on problem 1 in a feature branch	2. Setting up Spark in pseudo-distributed mode on Windows 3. Quickly understanding the Spark API and ecosystem	2. Finish the first version of the word count problem 3. Perform SE refactoring
Week 3 (03/10/22 - 09/10/22)	1. Researched different data sources for Big Data Analytics for problems 1 & 2 2. Found Common Crawl data sources website - a massive database of web scrapped data of petabytes of data 3. Finished version 1 of the Word Count Problem which works with normal, plain text and files 4. Tested Word Count v1 on data sources of various sizes	1. Parsing the .wet.gz files efficiently 2. Creating a good word parser system to clean up undesirable input (I.e., words with special chars (emojis, commas, dots, etc.)) 3. Adapt to the new Hadoop API using Tool and Configurable	1. Research cloud solutions or other distributed solutions to deploy my MR programs 2. Set up CI/CD with Jenkins 3. Fine-tune the existing problems

	5. Finished version 2 of the Word Count Problem which parses .wet files archived with gzip .gz from the Common Crawl database 6. Tested Word Count v2 with a massive Common Crawl scrapped web data source from 2013 that has 55000 websites 7. Refactored Word Count v1 and v2 based on SE principles and implemented an efficient clean-up method for the scrapped data		
Week 4 (10/10/22 - 16/10/22)	1. Fine-tuned the word count implementation 2. Researched distributed solutions for both cloud and on-prem 3. Reached out to the CIM team to use the RHUL cluster 4. Set up a Cloudera VM with 1 master and 3 workers on my PC and ran the word count problem 5. Researched setting up CI/CD using Jenkins and set up a local server on my laptop 6. Tried to create Azure and Amazon cloud clusters	1. Efficiently running the Cloudera VM on my machine was difficult because of the VM's high memory requirements. Ended up not being efficient. 2. Was unsuccessful in creating Cloud clusters with free accounts because there was a service reserved for paid subscriptions 3. Was unsuccessful in hooking up GitLab with my local Jenkins server because GitLab only allows Web servers	1. Looking up into setting up a Gitlab CI/CD 2. Discussing with the CIM team for using their cluster 3. Finish problem 2 and start writing the other reports

Week 5 (17/10/22 - 23/10/22)	1. Finished Problem 2 - Distributed Word Grep and applied to the 60k websites Common Crawl repo 2. Was granted access to the RHUL Big Data cluster and made an initial attempt to connect to it 3. Implemented CI/CD using a locally installed Gitlab Runner on my machine that uses a Docker agent 4. Every git push now successfully triggers a pipeline run for quality checking, building, and testing 5. Made an initial attempt at containerising the application using Docker and deploying it as an image on my personal Docker Hub 6. Started the MapReduce & Problems Encountered reports 7. Planned the second meeting with the advisor	1. Docker consumes a lot of resources while running either the build job or the push job, so I must close my IDE while running Docker Jobs 2. Connecting to the RHUL cluster proved problematic as I encountered config issues with the ssh	1. Add more tasks to the pipeline so it does comprehensive testing, packaging, building, and deploying to the Docker Hub as a tag 2. Add a more comprehensive testing suite for both Problems 3. Finish the MR, Problems Encountered & Experience running programs on cluster reports 4. Start testing and experimenting with the cluster and make observations of performance
Week 6 (24/10/22 - 30/10/22)	1. Refactored the Reducer for the Word Grep problem to consider end-line words when splitting the website into phrases	1. Writing relevant tests for the problems 2. Creating performance tests for the problems	1. Finish the pipeline configuration 2. Finish Docker adoption

	2. Added an extensive testing suite for the Word Grep and Word Count problems 3. Added performance testing for the Word Count & Word Grep Problems 4. Finished initial drafts for MR, Problems Encountered & Experience running programs on cluster reports	3. Writing the Experience running programs on cluster reports because of the ambiguity of the tasks	3. Start experimenting with the RHUL cluster
Week 7 (31/10/22 - 06/11/22)	1. Added Performance testing in the testing suite to measure execution time 2. Separated the integration & unit tests into 2 testing suites (handled by a specialised class) 3. Configured the maven test to run the testing suite and output the results to an XML file 4. Added integration & unit test jobs into the pipeline 5. Finished Docker adoption & pipeline configuration	1. Creating efficient performance tests 2. Creating a testing suite file with JUnit5 3. Integrating the tests into the pipeline	1. Start experimenting with the RHUL cluster 2. Start writing the experience with performance analysis, and parameter tuning; recommendation for optimal parameter setting for each of the analysed problem reports
Week 8 (07/11/22 - 13/11/22)	1. Migrated the test suite to use the RHUL Big Data cluster using ssh	1. Connecting with ssh to the RHUL cluster	1. Perform extensive performance testing & parameter tuning

	2. Fixed the CI/CD pipeline 3. Added a linting task in the CI/CD pipeline 4. Integrated the programs to work on the RHUL cluster using ssh	2. For many clusters runs, the MR job was executed locally because of an issue with the maven run config, the mitigation was to launch an ssh terminal from IntelliJ, package the drivers' classes into a jar and use that to run the programs on the cluster 3. Creating a proper linting configuration (I finally used Google Checks) and configuring my IntelliJ code formatting to accommodate Google Checks requirements	2. Start writing the last 2 reports
Week 9 (14/11/22 - 20/11/22)	1. Performed extensive performance testing using all the common crawl file partitions (from 120MB to 1800MB). 2. Plotted the found data on graphs showing the number of nodes vs data size vs execution time. 3. Compiled all the reports into one. Wrote up until the Word Grep implementation (approx. 5000 words). 4. Performed several experiments on the cluster to optimize parameters and performance.	1. Performing adequate performance testing 2. Refactoring the Driver into its class because I had to account for multiple runs time-specific settings 3. Correctly extracting the programs into jars to work both on local and distributed mode	1. Finish the Interim report 2. Add more unit & integration testing

	5. Refactored the Driver into its separate component to be used by all the specialised Drivers.		
Week 10 (21/11/22 – 27/11/22)	1. Finished the interim report 2. Refactored the word count v2 and word grep into a façade 3. Added more unit and integration testing 4. Packaged the programs into jars 5. Updated READ.me with all the required info about the project and how to run the programs.	1. Efficiently package the jars using the maven shade plugin 2. Combining all the required information into the interim report	1. Document on Natural Language processor classification for text categories 2. Start working on the Wikipedia page rank algorithm implementation with Spark
Week 17 (09/01/23 – 15/01/23)	1. Finished first version of the Page Rank algorithm that works will all wiki pages 2. Learned the basics of Spark 3. Created a parser for wiki page dumps that is processed into an rdd and then transformed into a graph 4. Created version 2 of the page rank algorithm with the use of GraphX	1. While updating an rdd multiple times, the maximum cache limit is Quickly reached. A solution for this is using checkpoints once a couple of iterations while updating the rdd. 2. Creating an efficient parser for the wiki dumps was complicated because of the sometimes poorly formatted dump structure. The solution included using multiple transformations and trying different approaches, in the end,	1. Improve the efficiency of the page rank algorithm 2. Modify the program so that after running the page rank on the entire wiki dump, it filters the rdd to output only biography pages

		choosing the databricks XML parser with a predefined Struct type.	
Week 18 (16/01/23 - 22/01/23)	1. Improved the performance of the page rank by eliminating redundant variables, using hashing and with proper use caching the rdd in memory in a serialised format. 2. Researched the way caching and serialization works and implemented memory serialization with the use of a powerful serialization library called Kryo. 3. Modified the page rank program so it figures out based on info-boxes if it's a biography or not. I collected a list of all the biography info-boxes format and then during dump processing, the infobox is checked if it is part of that list. 4. Researched ways to group pages in broader categories. Normal wiki categories are way too granular. Found out that the wiki categories graph is an acyclic direct graph with no strict hierarchical struct. This unfortunately means that a granular category can belong to multiple broader categories	1. Kryo 5x version causes some errors with the Spark 3x version which result in weird, non-existent errors at run time. Solved this by downgrading to Kryo 4x. 2. Finding a way to determine if a page is biographic was not straightforward and I had to get creative with the info-boxes approach. 3. Trying to find a way of using the wiki categories was time-consuming and not worth doing.	1. Research NLPs to group pages into categories based on their text. 2. Find a good labelled text dataset (or construct one if non-suitable exists). 3. Train the model and test it on the wiki dump

	so this approach is not doable. The alternative solution to explore is using an NLP.		
Week 19 (23/01/23 - 29/01/23)	1. Created the first version of the NLP classifier and trained it on a large amount of data to classify text into 5 categories. It has 99% accuracy on test data. 2. Researched the Wikipedia Knowledge Base and realised that since it is a directed acyclic graph with no hierarchy, I can't use it for my category classification (to infer the main category from the granular category since a granular category could have in theory multiple main categories as parents). 3. Researched other Knowledge Bases created from Wikipedia but again, it was too complicated and computationally expensive to use them.	1. Researching the KB for Wiki categories was tedious and I initially thought it was possible by reconstructing the graph. 2. Finding good quality datasets and transforming them into the same format (.csv) was complicated.	1. Improve the NLP by adding more classes (through more datasets). 2. Refactor the Jupyter Notebook into a standalone python file. 3. After training, save the model and its configuration into a file so it can be easily just loaded and used for prediction in the future.

	4. Researched main personal occupational categories areas for my NLP and created an initial list.		
Week 20 (30/01/23 - 05/02/23)	1. Refactored the model into its python file. 2. Added 4 more classes of 1000 entries each using several data sources 3. Refactored the python file to use pickle to save and load the model 4. Tested the model on preliminary Wikipedia data with good results	1. Saving and loading the model was more problematic than I through 2. There is a significant performance overhead when running the model on large amounts of data	1. Add even more categories and aim to have 20k entries for the model 2. Create some spark scripts to load and parse large datasets to generate data for the model

Week 21 (06/02/23 - 12/02/23)	1. Created some java spark classes to parse CSV, JSON and text data into my desired CSV format 2. Collected numerous online datasets and started to put them together 3. Refactored and improved the performance and accuracy of the model	1. Fining useful datasets online for my semantic requirements proved to be a challenge 2. Creating fast and useful spark file parsers was complicated	1. Aim to have 9 categories for the model 2. Finalize the data collection part
Week 22 (13/02/23 - 19/02/23)	1. Finished the data collection part, now I have 9 classes and 20k entries for each 2. Started proving the quality of my data by hand collecting Wikipedia articles from each topic and combining them with my existing data 3. Despite the large number of samples that are sometimes semantically close, managed to get a 91% accuracy	1. Collecting the Wikipedia articles took a lot of effort and time and required quite a bit of thought 2. Initially, because of some poor-quality data, my performance started to decrease significantly to about 80%	1. Create the spark version of the model. 2. Collect more Wikipedia data for training.

Week 23(20/02 - 26/02/23)	1. Created a spark version of my python scikitlearn model 2. Collected more data from Wikipedia to increase the quality of my data	1. Converting from python to spark was tedious as the pandas transformations and spark transformations are different and 2. Scikit-learn is not available in a distributed environment so I had to re-do my methods using the Spark Mlib	1. Deploy the model on the RHUL cluster for training to compare its performance and accuracy to the one running on my laptop kernel 2. Run the model on Wikipedia data and check preliminary results
Week 24(27/02/23 - 05/03/23)	1. Found out that the RHUL Cluster has an old version of Spark on it so I couldn't use pandas spark API and had to move to spark dataframes. 2. Experimented with deploying the model on the cluster by creating the virtual environment and shipping it with the model	1. Converting from spark pandas to spark dataframes was tedious 2. It was complicated running the python model on the cluster because of the virtual environment that had to be shipped with it. Furthermore, nltk had some packages that had to be downloaded programmatically that I had to ship with the environment as well.	1. Put both programs together and create the final historical score program 2. Work on the Final Report

Week 25 (06/03/23 - 12/03/23)	1. Ran the programs together and got the final significance scores. 2. Made good progress on the report.	1. Experimenting on the cluster with different run time configurations was tedious 2. The cluster was down for 2 days, not scheduling any jobs	1. Finalise final year report. 2. Finalize programs and refactor them.
Week 26 (13/03/23 - 19/03/23)	1. Finalised the report. 2. Finalised programs and performed extra refactoring. 3. Registered the deployment video.	1. Cluster was down for 3 days when high loads were being passed on. In the end, it was fixed, but with 2 nodes being temporally removed from the cluster. 2. Putting all the reports together in a final format was tedious.	N/A project finished