Natural Language Processing In A Distributed Environment

A comparative performance analysis of Apache Spark and Hadoop MapReduce

Ludwig Andersson
Abstract

A big majority of the data hosted on the internet today is in natural text and therefore understanding natural language and how to effectively process and analyzing text has become a big part of data mining. Natural Language Processing has many applications in fields such as business intelligence and security purposes.

The problem with natural language text processing and analyzing is the computational power needed to perform the actual processing, performance of personal computer has not kept up with amounts of data that needs to be processed so another approach with good performance scaling potential is needed.

This study does a preliminary comparative performance analysis of processing natural language text in an distributed environment using two popular open-source frameworks, Hadoop MapReduce and Apache Spark.
Acknowledgements

I would like to express my gratitude towards Sogeti and Johan Burman who was the external supervisor for this thesis. I also would like to thank Johanna Björklund for being my supervisor for this thesis.

Special thanks to Ulrika Holmgren for accepting this thesis work to be done at the Sogeti office in Gävle.
Contents

1 Introduction 1
   1.1 Purpose of this thesis 1
      1.1.1 Goals 2
   1.2 Introduction to MapReduce 2
      1.2.1 Map and Reduce 2
   1.3 Hadoop 3
      1.3.1 Hadoop's File System 4
      1.3.2 YARN 4
   1.4 Apache Spark 4
      1.4.1 Resilient Distributed Datasets 5
   1.5 Natural Language Processing 5
      1.5.1 Stanford's Natural Language Processor 6
   1.6 Previous Work 6

2 Experiments 7
   2.1 Test Environment 7
      2.1.1 Tools and Resources 7
      2.1.2 Test Machines 8
   2.2 Test data 8
      2.2.1 JSON Structure 8
   2.3 MapReduce Natural Language Processing 9
   2.4 Spark Natural Language Processing 10
      2.4.1 Text File 10
      2.4.2 Flat Map 11
      2.4.3 Map 11
      2.4.4 Reduce By Key 11
   2.5 Java Natural Language Processing 11
1 Introduction

Data mining is an active industrial topic right now and being able to process and analyze natural languages is a vital part. One of the challenges is that processing natural languages is a time consuming and computationally heavy task. The need for faster processing of natural language text is much needed, the amount of user generated content in form of natural language on the internet has increased dramatically and a solution to process it in a reasonable amount of time is needed.

On the larger social media platforms, popular forums and websites there exists huge amounts of data that can be collected and analyzed for a deeper understanding of user behavior. Twitter as an example has an average of about 6,000 tweets sent per second to their service, each tweet containing natural language text. These tweets can be analyzed and processed using natural language processing to gain important insight in users behavior and interests. But to do process huge data-sets such as all the tweets that gets tweeted in one day would require more computational power than a single computer can provide.

In this thesis we will take a look into two distributed frameworks, Apache Spark and Hadoop MapReduce. Apache Spark and Hadoop MapReduce will be evaluated in the terms of their processing time when processing Natural Language texts using Stanford’s Natural Language Processor.

1.1 Purpose of this thesis

The purpose of this thesis is to present a comparative performance analysis of Apache Spark and Hadoop’s implementation of the MapReduce programming model. The performance analysis involves processing of natural language text for named-entity recognition. This is to give an indication on how the fundamental differences between MapReduce and Spark differentiates the two framework in terms of performance when processing natural language text.

https://twitter.com/
http://www.internetlivestats.com/twitter-statistics/
1.1.1 Goals

The goals for this thesis are:

- Present a preliminary performance analysis on Apache Spark and Hadoop MapReduce.
- Give an insight to the key differences between the MapReduce paradigm and Apache Spark and how they should help in the choice of framework.

1.2 Introduction to MapReduce

MapReduce was first introduced as a programming model in 2004 by a research paper published by Google and written by Jeffrey Dean and Sanjay Ghemawat. Jeffrey Dean and Sanjay Ghemawat describes MapReduce as a "programming model and an associated implementation for processing and generating large data-sets" [1].

Lin and Dyer [2] states that to overcome the problem with handling large data problems, Divide and Conquer is the only feasible approach that exists today. Divide and Conquer is a fundamental concept in Computing Science and is often used to divide larger problems into smaller independent sub-problems that can be solved individually/in parallel [2, Chapter 2].

Lin and Dyer [2] names that even though divide and conquer algorithms can be used on a wide array of problems in many different areas the difference in the implementation details of divide and conquer can be very complex due to the low-level programming problems that needs to be solved and therefore lots of attention is placed on solving low-level details such as synchronization between workers, avoiding deadlocks, fault tolerance and data sharing between workers. This requires the developers to explicitly pay attention on how the coordination and access between data is handled [2, Chapter 2].

1.2.1 Map and Reduce

The MapReduce programming model is built on an underlying implementation with two functions exposed to the programmer, a Map function, that generally performs filtering and sorting and a Reduce function that often perform some collection/summary from the data generated from the Map function. The Map and Reduce functions works with key-value pairs that is sent from the Map function to the Reduce function. The Map function often generates an intermediate key-value pairs from the input given and the reduce function often will perform some summation of the values generated from the Map function.

Figure [1] shows how the mappers, containing the user defined map function is applied to the input which generates another set of key-value pairs which is then used by the reducers which contain the user supplied function for Reduce to do
some operation on the key-value pairs received from the mappers.

**Figure 1**: A simplified illustration representing the pipeline for a basic MapReduce program that count character occurrences.

### 1.3 Hadoop

Hadoop is a MapReduce open-source framework for distributed computing which makes it great for handling computational heavy applications. It takes care of work and file distribution in a cluster environment. Hadoop is intended for large scale distributed data-processing with scaling potential, flexibility and fault tolerance.

The Hadoop project was firstly released in 2006 and has since then been growing into the project it is today. The Hadoop project now exists in plenty of different distributions from companies such as Cloudera[3] and Hortonworks[4] that offers its own software bundled with Hadoop.

Hadoop’s implementation of MapReduce is in Java and is one of the most popular today in the open-source world and is also the one that is going to be evaluated in this study.

---

4[http://hortonworks.com](http://hortonworks.com)
1.3.1 Hadoop’s File System

Hadoop comes with a distributed file-system called HDFS (Hadoop Distributed File System). It is a distributed file-system that spreads out across the worker nodes in the Hadoop cluster.

It is one of the core features in Hadoop’s implementation of MapReduce. It gives Hadoop as a framework fault tolerance by replication of the data blocks to scatter across the nodes in the cluster. Also HDFS makes it possible to move the code to the data and not the data to the code, this removes any bandwidth and networking overhead which may be a huge bottleneck in performance when dealing with large data-sets [2, Chapter 2.5].

In this thesis Hadoop’s distributed file-system will be the storage solution used for both MapReduce and Apache Spark. This because Hadoop’s implementation only supports the HDFS as input source [2, Chapter 2.5]. While Apache Spark supports a few other storage solutions (see chapter [1.4]) this is the storage system that will be used to give a fair comparison between the frameworks.

1.3.2 YARN

YARN (Yet Another Resource Negotiator) is a resource manager and a job scheduler that comes with the later versions of Hadoop’s MapReduce. The purpose of YARN is to separate the resource management from actual programming model in Hadoop MapReduce [3]. YARN’s purpose is to provide resource management in Hadoop clusters from work planning, dependency management, node management to keeping the cluster fault tolerant to node failures [3].

Together with HDFS replication ability, YARN provides the Hadoop framework with fault tolerance and error recovery for failing nodes. The distributed file system replicates data blocks across the computer cluster making sure that the data is available on at least another node. YARN provides the functionality of re-tasking work during run-time of a MapReduce job to keep the processing going without having to restart the job from the beginning. This is pointed out as a major advantage for the Hadoop framework by a performance evaluation between Hadoop and some parallel relational database management systems [4].

In this thesis YARN will be used as the resource manager for both Hadoop MapReduce and Apache Spark during the experiments.

1.4 Apache Spark

Apache Spark is a new project and is a viable "junior" competitor to MapReduce in the big data processing community. Apache Spark has one implementation that is portable over a number of platforms. Spark can be executed with Hadoop and HDFS but Spark can also be executed in a standalone mode with its own
When using Spark the programmer is free to construct the execution pipeline of the program so it fits the problem, this solves one of issues in MapReduce where the problem needs to be adjusted to work with the Map and Reduce model.

Spark does however have higher system requirements for effective processing due to its in-memory computation using resilient distributed datasets instead of writing output to the distributed filesystem, this requires that the cluster that Spark runs on has to have an equal or more amount of memory than the data-set that's being processed.

### 1.4.1 Resilient Distributed Datasets

Even though MapReduce provides an abstraction layer for programming against computer clusters it’s pointed out that MapReduce lacks an effective model to program against shared memory that is available on the slave machines through out the computer cluster.

In "Resilient distributed datasets" a RDD (Resilient Distributed Dataset) is defined as a collection of records that can only be created from a stable storage or through transformation from other RDD’s. The other features of an RDD is that its a read-only after its creation and partitioned.

In Apache Spark the programmer can access and create RDD’s through the in-built language API either in Scala or Java. As mentioned above, the programmer can choose to either create the RDD from storage, a file or several files that is stored in any of the supported storage options mentioned in 1.4. The other option is to create a RDD through transformations such as `map` and `filter`, their primary use is often to create new RDD’s from already existing RDD’s.

### 1.5 Natural Language Processing

Natural Language Processing, also known as NLP is a field in computer science that starts out in the 1950’s. It covers fields such as Machine Translation, Named Entity Recognition, Speech Processing and Information Retrieval and Extraction from Natural Language.

Named-entity recognition is a sub-task of information extracting in natural language processing. It goal is to find, classify tokens in a text into named entities such as persons, organizations dates and times.

---

6 [https://aws.amazon.com/s3/](https://aws.amazon.com/s3/)
A person named "John Doe" would be recognized as a person entity and "Thursday" would be recognized as a date entity.

1.5.1 Stanford’s Natural Language Processor

Stanford’s Natural Language Processor is an open-source project for natural language processing from Stanford University. It provides a number of language analysis tools and is written in Java. The current release requires at least Java 1.8 and you can use it either from the command-line, via the Java programmatic API, third party API or through a service.

Stanford’s NLP provides a framework (CoreNLP) which makes it possible to make use of the inbuilt language analysis tools on data sets containing natural language text. The basic distribution of the CoreNLP provides support for English but the developers also state that the engine has support for other language models as well.

For this thesis the Stanford’s Natural Language Processor is going to be used in a distributed environment with Apache Spark and Hadoop MapReduce using a proposed approach by Nigam and Sahu [6].

1.6 Previous Work

There has been some previous research done in this topic. The problem with slow natural language processing seems to be well known but many solutions are however often proprietary and therefore not released to the public.

However, Nigam and Sahu addressed this issue in their research paper "An Effective Text Processing Approach With MapReduce" [6] about this problem and proposed an algorithm for implementing natural language processing within a MapReduce program.

There has also been a comparative study between MapReduce and the Parallel database management systems Vertica and DBMS-X. They were compared to Hadoop MapReduce [4]. This study shows the difference in performance between SQL languages and Hadoop when testing different tasks in both frameworks. This study shows that parallel DBMS have superior performance to MapReduce in the past (Version 0.19.0) of Hadoop. In the conclusions the researchers of that research paper concluded that the parallel database management systems were faster in general than Hadoop but they lacked the extension possibilities and the excellent fault tolerance that Hadoop has.

[http://stanfordnlp.github.io/CoreNLP/]
2 Experiments

To evaluate Apache Spark and MapReduce frameworks, a simple NER (Named Entity Recognition) program was used. The tasks where to count the number of times the same NER tag was found in the text, similar to a "wordcount" program but with NER as well. The reason behind choosing NER was because it’s computationally heavy and is often used when analyzing natural language text.

2.1 Test Environment

The experiments were conducted under two different environments, one sequential environment with one CPU core was used and one distributed environment hosted Google Cloud\(^1\). The baseline program was executed on the sequential environment and is used to establish a baseline performance benchmark for the distributed performance tests to come between Hadoop MapReduce and Apache Spark.

2.1.1 Tools and Resources

The tools used for conducting these experiments were

- Hadoop
- Hadoop MapReduce
- Apache Spark
- Stanford’s Natural Language Processor

The tools mentioned above was used together with Java version 1.8 and together with Google Cloud\(^1\). Google Cloud provided a one-click deploy solution for Hadoop with MapReduce and Apache Spark together together with Google Dataproc\(^2\).

\(^1\) https://cloud.google.com
\(^2\) https://cloud.google.com/dataproc/
2.1.2 Test Machines

For running the experiments different machines were needed to test the different approaches to text processing. In total we tested in two different environments, one environment that was hosted on a single four core Intel i-7 with hyper-threading running Windows 10. When testing the scaling potential of the MapReduce jobs a Google Cloud Processing cluster was used with n1-standard-1 machines. The n1-standard-1 machines was specified with one virtual CPU and 3.75GB of RAM.

The Hadoop version that was used was 2.4.1 for both the cloud environment and the sequential environment. Also since Stanford’s Natural Language processor requires Java 1.8 as a minimum that was the Java version that was used both locally and on the Google Cloud platform.

2.2 Test data

For the performance analysis, JSON (JavaScript Object Notation) files were used and is a common file type for sending and receiving data on the internet today. The files where formatted using a forum like syntax with information about the user, which date it was posted and the body (text corpus of the post).

The test data is formatted in a way it would somewhat represent a realistic example of how a forum post could look like formatted in a JSON object.

2.2.1 JSON Structure

In listing 2.1 a sample JSON object is presented, this object will later be refereed in the results as a record and the processing will be on a variable number of these records.

```
Listing 2.1: JSON Record

1 {  
2   "parent_id": "",
3   "author": "",
4   "body": "",
5   "created_at": "",
6  }
```

Where parent_id represents an id to determine to which post the record belongs to, the author tag represents an id of an user, the body tag holds the text content which will be processed to search for NER tags and the created_at tag holds the date when the post or reply was posted.

For the experiments run only the content of the body tag will be taken in account when processing the natural language text. The body tag comes in a variable length of sentences and words.
2.3 MapReduce Natural Language Processing

For implementing the MapReduce version of text processing algorithm proposed by Nigam and Sahu [6] that suggested that the analysis should be made in the Mapper and the Reducer is to act as a summary function to sum together the intermediate results from the Mapper function of the MapReduce program. As the input data in for these experiments come is JSON format and that the text that we want natural language processing on is in the body tag, we use a simple JSON parser as well in the mappers to fetch the content of that particular JSON tag.

Listing 2.2: Map Psuedo Code

```
function map
  parseJSON
  extractSentences
  for each sentence
    extract nerTag
    write (nerTag, 1)
```

Listing 2.3: Reduce Psuedo Code

```
function reduce
  sum = 0
  for each input
    sum += value.val
  result.set(sum)
  write(key, result)
```

In the two listings above of the psuedo-code for the Map function and the Reduce function it is to be noted that the actual natural language processing is done in the Mapper and the Reducer function will simply sum together the intermediate output of the Mapper.
2.4 Spark Natural Language Processing

Since spark uses a Directed-Acyclic-Graph (DAG) based execution engine, it’s execution flow looks a bit different and is more flexible than the execution flow of a MapReduce program. In Figure 2 there is an overview on how the experiment is executed with Apache Spark.

![Figure 2: Flow of execution by the Spark Application](image)

The Spark job is divided into two stages by the DAG-based execution engine, stage 0 has three "sub-stages" that needs to be executed before the execution in stage 1 can begin. The components and arrows between them shows how which transformation methods that is used with the resilient datasets throughout the execution of the Spark application.

2.4.1 Text File

In this first stage, an initial RDD is created from input file or files that reside in the distributed file system. The input is read into an RDD as strings and each string contains a JSON record formatted as described in listing 2.1.
2.4.2 Flat Map

In this stage, the natural processing is applied using the Named-Entity-Recognition implementation given by Stanford’s Natural Language Processor. Using the RDD created in the "textFile" stage this stage iterates through the JSON objects and processes the body tags and uses RDD’s transformation method to create a new RDD containing the results from the FlatMap function containing the word and the named entity tag as a string in the RDD.

2.4.3 Map

This stage is when the key-value pairs is created, from the RDD of words and their NER tag as the key and an Integer as value.

2.4.4 Reduce By Key

And as similar to the MapReduce version of the Reduce this stage sums together all the key-value pairs generated from the Map stage into a new key-value pair list which holds the word and the NER tag as key and the number occurrences of that word as the value.

This is the final transformation method used to create the last RDD that contains the results that will be written to an output file in the distributed file system.

2.5 Java Natural Language Processing

The baseline Java NLP program consists of a standalone Java process that runs like a "normal" executable JAR file. The program is divided into three parts. The first part is where the body tag gets extracted from the JSON file, then we apply the natural text processing toolkit from Stanford[^3] and then we write the output into a file.

[^3]: [http://stanfordnlp.github.io/CoreNLP/](http://stanfordnlp.github.io/CoreNLP/)
2.6 Output Format

For equality purposes regarding output, all test applications have to write the same output in the same format. Since applications objective is to count the number of times a NER tag occurs in a text corpus the output is as following.

<table>
<thead>
<tr>
<th>Listing 2.4: Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (String,NERTAG) Count</td>
</tr>
</tbody>
</table>

The text output can in an future implementation be easily sorted and analyzed using a simple parser.

The output after parsing a series of words using named entity recognition the output list grows. As an example, "John visited his father Tuesday night. John usually visits every Tuesday." will give the following output

<table>
<thead>
<tr>
<th>Listing 2.5: Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (John, PERSON) 2</td>
</tr>
<tr>
<td>2 (father, PERSON) 1</td>
</tr>
<tr>
<td>3 (Tuesday, DATE) 2</td>
</tr>
<tr>
<td>4 (night, TIME) 1</td>
</tr>
</tbody>
</table>

In the experiments the non entity tags (tagged by Stanford’s NLP as "O") has been ignored and not to be used to limit the results only to named entities.
3 Results

In order to compare the approaches a performance analysis has to be made comparing the sequential and the distributed approaches to natural language processing. In the comparative analysis of the sequential approach MapReduce, Apache Spark and the baseline Java program will be evaluated in terms of execution time over a data-set of sample data. In the distributed environment the same comparative analysis of Apache Spark and MapReduce will be made using jobs running across a cluster. The distributed comparison will be between different sizes in data-sets over a different amount of virtual machines hosted on the Google Cloud platform.

3.1 Sequential Processing

By sequential processing it means that no effective measures has been taken to achieve parallelism on any of the experiments. All applications have been executed like a normal process under the JVM.

![Graph showing execution time vs JSON Records]

https://cloud.google.com/
3.2 Scaling up

Here the cluster is first set at 4 worker nodes with 1 virtual CPU per worker, and then we scale up the cluster to 6 worker nodes with 1 virtual CPU per worker.
### 3.3 Scaling Out

These results show the use of multicore processors per machine and how it’s possible to scale out the number of cores in relation to the number of machines.

![Graph showing execution time for different JSON records and machine configurations.](image-url)
4 Performance Analysis

As we can see from the first test between the baseline Java program versus a sequential MapReduce program running under Hadoop is that they performed almost the same. There were only a three minute difference in average on parsing 150,000 records. It is reasonable to assume that the read and writes to disk slowed the MapReduce jobs a bit as well taking into account the loading of the Hadoop framework. The baseline Java program held all the information it needed in memory before writing the output to a file which was its only I/O operation. Also the Apache Spark application that had similar execution time as the baseline Java application for the 150,000 records test in the sequential environment. The similarity between the baseline and the Apache Spark application was that both held the data it needed in memory before writing the output to a file which MapReduce does not.

The overall performance of the sequential processing for all three methods was the same, its safe to come to the conclusion that none of the distributed approaches had any advantages or major disadvantages being executed on a local computer on a single core.

Worth noting is that both Apache Spark and MapReduce had significantly longer start-up times before any processing actually began than the baseline program, this is due framework loading and startup times in the background for both Hadoop MapReduce and Apache Spark.
4.1 Spark and MapReduce

Both Apache Spark and MapReduce are popular frameworks today for cluster management and processing and analyzing big data. Both Apache Spark and Hadoop MapReduce provides an abstraction level that directs the focus towards solving the real problem and lets the framework handle the low-level implementation details.

Both frameworks where applicable to the purpose of processing natural language text and named-entity recognition. The key difference is that Spark is a more general framework with a flexible pipeline of execution due to its DAG based execution engine where as MapReduce requires the problem to be re-formed so it fits into the Map and Reduce paradigm.

The choice between Apache Spark and MapReduce should not entirely land in the performance measured. Some thought should be given if the problem fits the MapReduce paradigm, if not then maybe Apache Spark can be the choice due to its flexibility with its execution engine and use of resilient distributed datasets.

4.1.1 Performance

In terms of performance Spark and MapReduce where quite similar on the small data sets but when reaching larger amounts of data on multiple single core machines then MapReduce had a significantly shorter execution time. However on the corresponding test but with two CPU’s per machine then Spark performed better than MapReduce.

On a multicore machine Spark had less execution times than MapReduce, also in these experiments Spark had enough memory to compute its pipeline using its in-memory feature so Spark had almost no disk writes except writing the output to a file.

In general the biggest performance difference between Spark and MapReduce was impacted on if the machines had a single virtual CPU or multiple virtual CPU’s. In the tests of a scaling cluster of machines consisting only of only one virtual CPU then MapReduce performed better while in the tests of the same total amount of cluster cores but with fewer machines but with two virtual CPU’s per machines then Spark had shorter execution times than MapReduce.

4.1.2 Community and Licensing

Apache Spark is in comparison to MapReduce a very new framework that has gained a lot of traction in the Big-Data world. According to Wikipedia, Spark had its initial release the 30 of May 2014 from Apache, where Hadoop MapReduce was released December 10, 2011 as an Apache Project.

Both Spark and MapReduce have large communities and are broadly supported commercially by companies providing big data solutions for both frameworks. Major application service providers such as Microsoft, Google, and Amazon provide MapReduce and Spark with Hadoop as one-click-deploy clusters with either framework pre-installed and configured.

Figure 3: Showing the search trends on Google for Hadoop (Blue), MapReduce (Yellow) and Apache Spark (Red)

In Figure 3 it can be observed that Apache Spark is a newcomer compared to Hadoop but has gained a lot of traction since its release. Hadoop MapReduce and Apache Spark are Open-Source projects and licensed under the Apache 2.0 license.

4.1.3 Flexibility and Requirements

Both MapReduce and Spark are cross-platform compatible which means that they can run pretty much under all machines that have an appropriate JVM. Hadoop (2.4.1) which is implemented in Java requires at least Java 6 and the later versions of Hadoop (version 2.7+) require Java 7. Apache Spark is implemented using Scala that runs also under the JVM as well which should make the system compatibility requirements quite similar for both Spark and Hadoop MapReduce.

One of the main features of Spark is its ability to perform in-memory computation without the need to read/write to disk after every stage in the job, its often marketed as much faster than MapReduce. This feature does however come at a cost of having equal or larger amounts of memory than the data processing.

Flexibility is not great in MapReduce as mentioned earlier in the introduction to this thesis, the Map and Reduce paradigm must be thought of. Spark however offers a more flexible solution but is much less refined and mature than MapReduce in terms of being a much newer framework solution.

http://www.apache.org/licenses/LICENSE-2.0
4.2 Limitations

During these experiments the trial account on Google Cloud had a limitation of the number of eight CPU cores per cluster. Also since we were using a cloud platform where multi-tenant machines is used, each of the test runs performed where executed five times each to provide an accurate execution time and to normalize the possible load provided by other tenants using the same physical server.

4.3 Further Optimization

As this was a preliminary study there where no in-depth tuning of parameters for either the MapReduce jobs nor the Apache Spark jobs.

To optimize further a multithreaded mapper could be used for MapReduce to support concurrency within each Map and also specifying the amount of Map/Reduce tasks for the MapReduce jobs to match the number of file splits and cpus’s available could increase performance for the MapReduce jobs.
5 Conclusion

This thesis gives an preliminary study of the performance between Apache Spark and MapReduce and how they can be applied for processing natural language text with Java based frameworks.

Both Spark and MapReduce are giants in the open-source big-data analytic world. At first glance we have the old and refined MapReduce framework that has many implementations in different languages but on the other side we have a new but very promising framework Apache Spark that has one implementation but can be executed almost everywhere and can be seen as more flexible in that approach.

Both Apache Spark and Hadoop MapReduce shows promising potential to scale across more machines than was tested during this study much better than a sequential processing on a local machine. Even if your local machine has several cores it can’t scale up like a distributed environment can. Even if unproven in this thesis on a larger scale on thousands of machines, Apache Spark and Hadoop MapReduce has great scaling potential. Even in clusters with a few number of machines the speedup of using either of these two frameworks was substantial.

Natural language processing in a distributed environment using open-source frameworks like Apache Spark or Hadoop MapReduce is proven in this thesis as a feasible approach to natural language text processing using large data sets.
6 Further Research

As this is a preliminary study and the resources where limited a further point to investigate is the scaling potential of enterprise sizes of 100’s of machines running in a cluster and with even more data-sets. A follow up study could also show the potential of enterprise distributions of Hadoop such as those from Cloudera\(^1\) or Hortonworks\(^2\) and the performance of their Hadoop distributions compared to Apaches distribution of Hadoop.

Another further point of research is to take a look into Stanford Natural Language Processor it self to see if its possible to rewrite the source code to fit into a distributed environment.

One future point of research is to investigate storage solutions that exists for Apache Spark and MapReduce and compare their scaling potential against parallel database management systems in a Natural Language Processing approach, such research could possible be based on a previous research paper that compares an old version of Hadoop MapReduce against a set of popular parallel database management systems \(^4\).

Also one interesting topic to take further is the other implementations of MapReduce that exists today. Implementations as MapReduce-MPI\(^3\) that is written in C++ or HPC adapted implementations such as MARIANE.

As this is a preliminary study of investigating the use of cluster environment frameworks for shortening the execution times for natural language processing, its use is to point the way for further more in-depth research in the application for MapReduce and Apache Spark and how they can be utilized for natural language processing.

\(^1\)http://www.cloudera.com/
\(^2\)http://hortonworks.com/
\(^3\)http://mapreduce.sandia.gov
References


