PARALLELISING NLP TASKS USING MAP REDUCE PARADIGM

PROJECT REPORT

Submitted by:

FRENY CLARA DAVIS PUTHUR [EPALECS015]

SHALINI M [EPALECS048]

SHIBHIN MOHAN [EPALECS049]

NIDHIN M MOHAN [EPALECS069]

for the award of the degree

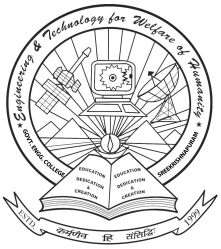
of

Bachelor of Technology

in

Computer Science and Engineering

(University of Calicut)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT ENGINEERING COLLEGE

SREEKRISHNAPURAM

PALAKKAD-678 633

May 2015

Declaration

We hereby declare that the project report entitled as **"PARALLELISING NLP TASKS USING MAP REDUCE PARADIGM"** submitted by us to the Department of **Computer Science and Engineering, Government Engineering College,** Sreekrishnapuram, is a bonafide work undertaken by us and it is not submitted to any other University or Institution for the award of any degree or published any time before.

Place: Sreekrishnapuram

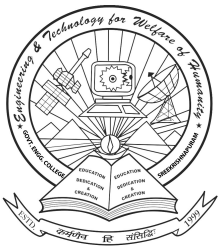
Date :

FRENY CLARA DAVIS	EPALECS015
SHALINI M	EPALECS048
SHIBHIN MOHAN	EPALECS049
NIDHIN M MOHAN	EPALECS069

GOVERNMENT ENGINEERING COLLEGE

SREEKRISHNAPURAM

PALAKKAD-678 633



Certificate

This is to certify that the project report entitled **PARALLELISING NLP TASKS USING MAP REDUCE PARADIGM** *submitted by* **FRENY CLARA DAVIS PUTHUR, SHALINI M, SHIBHIN MOHAN and NIDHIN M MOHAN** *to the Department Of Computer Science and Engineering, Government Engineering College, Sreekrishnapuram, Palakkad, in partial fulfillment of the requirement for the award of B.Tech Degree in Computer Science and Engineering under Calicut University is a bonafide record of the work carried out by them.*

Dr.P C REGHURAJ
Guide

Asst. Prof. BINU R
Co-ordinator

Dr.P C REGHURAJ
Head Of Department

Place : Sreekrishnapuram

Date :

Acknowledgement

It stands to reason that the completion of main project needs the support of many people. We take this opportunity to express our boundless thanks and commitment to each and every one, who helped us in successful completion of our project.

First and foremost we wish to express wholehearted indebtedness to **God Almighty** for his gracious constant care and magnanimity showered blissfully over us during this endeavour.

We would also like to acknowledge to Principal **Dr. S. Radhakrishnan**, Govt. Engineering College, Sreekrishnapuram, for his advice and encouragement in our project.

We are thankful to **Dr.P C REGHURAJ**, our Internal Guide and Head of Department, Computer Science and Engineering , Govt. Engineering College Sreekrishnapuram, for providing and availing us of all the required facilities to prepare this report and also inspiring and providing sincere guidance throughout our project.

We express our heartfelt gratitude to **Asst. Prof. BINU R** for working as our project co-ordinator.

Gratitude is extended to all teaching and non teaching staffs of Department of Information technology, Govt Engineering College Sreekrishnapuram for their cooperation to complete this report.

We are also thankful to our parents who constantly supported us. Gratitude may be extended to all well-wishers and our friends who supported us to complete this report in time.

Contents

List of Figures

Abstract

Natural Language Processing(NLP) refers to the applications that deal with natural language in a way or other. The proposed system tries to parallelise NLP tasks using map-reduce paradigm. MapReduce is a framework for performing data-intensive computations in parallel on commodity computers. The main NLP tasks that the proposed system performs are POS tagging, Stemming, Stopword Removal, Named-entity recognition and Sentimental analysis. The MapReduce algorithm is implemented in Hadoop which is an open source framework for writing and running distributed applications that process large amounts of data. If tasks like POS tagging and Stemming are done sequentially, it may take a lot of time. This is where Hadoop is relevant. Hadoop scales up linearly to handle larger data sets by adding more nodes to the cluster. It also allows users to quickly write efficient parallel code.

Keywords: ***Map-Reduce, NLP, Hadoop, POS Tagging, Stemming, Stopword Removal, Named-Entity Recognition, Sentimental Analysis***

Chapter 1

Introduction

Modern information societies are defined by vast repositories of data, both public and private. Therefore, any practical application must be able to scale up to datasets of interest. Big data problems are often complex to analyze and solve. The sheer volume, velocity, and variety of the data make it difficult to extract information and business insight. The MapReduce computing paradigm is an architectural and programming model for efficiently processing massive amount of raw data in a parallel and distributed manner. With the recent proliferation of cloud computing services, the MapReduce framework has become a very popular approach to process Big Data in distributed environments. It provides seamless distribution of computing tasks among computing nodes, in a transparent way to programmers. Its current design allows users' data to be efficiently processed in multiple parallel tasks, with little control from the end users. Considering its benefits, MapReduce has been widely adopted by a number of large companies, such as Google, Yahoo, Amazon, Facebook and AOL. It was originally developed by Google and built on well-known principles in parallel and distributed processing dating back several decades. MapReduce has since enjoyed widespread adoption via

an open-source implementation called Hadoop, whose development was led by Yahoo (now an Apache project). Today, a vibrant software ecosystem has sprung up around Hadoop, with significant activity in both industry and academia. With this achieved flexibility and great MapReduce degree of scalable computing offered to its clients, we explore parallelising of NLP tasks using map-reduce paradigm. The main NLP tasks that the system tries to implement are POS tagging, Stemming, Stopword Removal, Named-Entity recognition and Sentiment Analysis. The input to this application is Big data which may range from gigabytes to terabytes. These NLP tasks can further be used in many areas especially for information retreival. If these NLP tasks like POS tagging and Stemming are done sequentially, it may take a lot of time. This is where Hadoop is relevant. Hadoop scales up linearly to handle larger data sets by adding more nodes to the cluster. It also allows users to quickly write efficient parallel code. Here we show the difference in time taken to execute the map reduce program in a single node as well as a cluster of nodes.

Chapter 2

Literature Survey

2.1 NLP Tasks

Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP involve natural language understanding, that is, enabling computers to derive meaning from human or natural language input, and others involve natural language generation. The history of NLP generally starts in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence.

The Main NLP tasks that the system implements are :

- Part-of-Speech Tagging
- Stemming

- Stopword Removal
- Named-Entity Recognition
- Sentiment Analysis

2.2 POS Tagging

Part of speech tagging is the process of assigning a part- of-speech or other lexical class marker to each word in a corpus. Taggers play an important role in speech recognition, natural language parsing and information retrieval. The input to a tagging algorithm is a string of words and the specified tagset. The output is the best tag for each word.

2.2.1 Example

An example of POS Tagging is given:

- text = "And now for something completely different"
- pos tag = [('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')]

2.2.2 Applications

POS Tagging is useful in

- Information Retrieval
- Text to Speech Conversion

- Word Sense Disambiguation
- Useful as a preprocessing step of parsing

2.3 Stemming

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form. Stemming is the term used in linguistic morphology and information retrieval to describe the process for reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem needs not to be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.

Stemming programs are commonly referred to as stemming algorithms or stemmers.

2.3.1 Example

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root) but "argument" and "arguments" reduce to the stem "argument".

2.3.2 Algorithms

1. Lookup algorithm

   A simple stemmer looks up the inflected form in a lookup table. The advantages of this approach is that it is simple, fast, and easily handles exceptions. The disadvantages are that all inflected forms must be explicitly listed in the table: new or unfamiliar words are not handled, even if they are perfectly regular (e.g. iPads iPad), and the table may be large. For languages with simple morphology, like English, table sizes are modest, but highly inflected languages like Turkish may have hundreds of potential inflected forms for each root.

   A lookup approach may use preliminary part-of-speech tagging to avoid overstemming.

2. Suffix-stripping algorithms

   Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm, given an input word form, to find its root form. Some examples of the rules include:

   - if the word ends in 'ed', remove the 'ed'
   - if the word ends in 'ing', remove the 'ing'
   - if the word ends in 'ly', remove the 'ly'

   Suffix stripping approaches enjoy the benefit of being much simpler to maintain than brute force algorithms, assuming the maintainer

is sufficiently knowledgeable in the challenges of linguistics and morphology and encoding suffix stripping rules. Suffix stripping algorithms are sometimes regarded as crude given the poor performance when dealing with exceptional relations (like 'ran' and 'run'). The solutions produced by suffix stripping algorithms are limited to those lexical categories which have well known suffixes with few exceptions. This, however, is a problem, as not all parts of speech have such a well formulated set of rules. Lemmatisation attempts to improve upon this challenge.

3. Lemmatisation algorithms

   A more complex approach to the problem of determining a stem of a word is lemmatisation. This process involves first determining the part of speech of a word, and applying different normalization rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on a word's part of speech.

   This approach is highly conditional upon obtaining the correct lexical category (part of speech). While there is overlap between the normalization rules for certain categories, identifying the wrong category or being unable to produce the right category limits the added benefit of this approach over suffix stripping algorithms. The basic idea is that, if the stemmer is able to grasp more information about the word being stemmed, then it can apply more accurate normalization rules (which unlike suffix stripping rules can also modify the stem).

4. Porter Stemmer

   Porter's algorithm was developed for the stemming of English-

language texts but the increasing importance of information retrieval in the 1990s led to a proliferation of interest in the development of conflation techniques that would enhance the searching of texts written in other languages. These stemmers are described in a high-le vel computer progra mming language, called Snowball (Porter, 2006) that has been developed to provide a concise but unambiguous description of the rules for a stemmer. Snowball provides a concise but powerful description that can then be processed by a compiler to give a C or Java implementation of the algorithm for the chosen language (Porter, 2001). In passing, it is worth noting that this paper by Porter contains an extremely illuminating discussion of stemming and the structures of words that is very well worth reading, even if one does not wish to obtain any of the downloadable programs.

2.3.3 Applications

Stemming is used as an approximate method for grouping words with a similar basic meaning together. Some of its area of application include:

- Information retrieval
- Domain Analysis
- Use in commercial products

2.4 Stopword Removal

In computing, stop words are words which are filtered out before or after processing of natural language data (text). There is no single

universal list of stop words used by all processing of natural language tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as 'The Who', 'The The', or 'Take That'. Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance. Hans Peter Luhn, one of the pioneers in information retrieval, is credited with coining the phrase and using the concept.

Many times, it makes sense to not index "stopwords" duing the indexing process. Stopwords are words which have very little informational content. These are words such as:and, the, of, it, as, may, that, a, an, of, off, etc.

Studies have shown that by removing stopwords from the index, you may benifit with reduced index size without significantly affecting the accuracy of a user's query. Care must be taken however to take into account the user's needs. For example, the phrase "to be or not to be" from Hamlet is composed entirely of stopwords. Most of the internet's search engines eliminate all the stopwords from their indexes. By eliminating stopwords from the index, the index size is typically reduced by about 33% for a word level index. For a record level index or IDF level index, then eliminating stopwords is not typically done as

they will not add significantly to the index size.

2.4.1 Applications

The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead. Stop words can be used in a whole range of tasks and these are just a few:

- Supervised machine learning - removing stop words from the feature space
- Clustering - removing stop words prior to generating clusters
- Information retrieval - preventing stop words from being indexed
- Text summarization - excluding stop words from contributing to summarization scores and removing stop words when computing ROUGE scores

2.5 Named-Entity Recognition

Named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

2.5.1 Applications

- Information Retrieval

- Named entities can be indexed, linked off, etc
- For question answering, answers are often named entities.
- Many web pages tag various entities, with links to bio or topic pages, etc.

2.6 Sentiment Analysis

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document.

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level — whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry," "sad," and "happy."

2.6.1 Applications

Some applications of Sentiment Analysis are in areas like:

- Business Intelligence
- Cross-Domain applications like Sociology, Psychology etc
- Evaluation of Public voter's opinion

Chapter 3

MapReduce

MapReduce is a functional programming paradigm. It enables parallel programming of large data efficiently using multiple nodes. Its programming model is built upon a distributed file system (DFS) which provides distributed storage.The main idea of the MapReduce model is to hide details of parallel execution and allow users to focus only on data processing strategies.

The model is inspired by the map and reduce functions commonly used in functional programming.

3.1 Overview

MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Processing can occur on data stored either in a filesystem (unstructured)

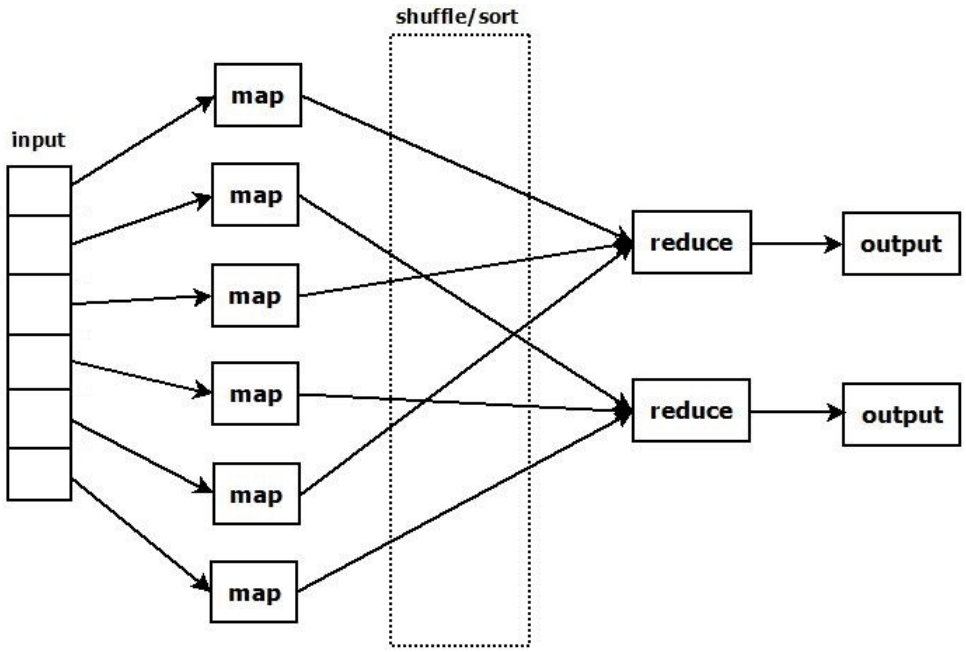


Figure 3.1: MapReduce Architecture

or in a database (structured). MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.

The steps performed in the map-reduce program are:

1. *Map step* : Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.

2. *Shuffle step* : Worker nodes redistribute data based on the output keys produced by the map() function, such that all data belonging to one key is located on the same worker node.

3. *Reduce step* : Worker nodes now process each group of output data, per key, in parallel.

3.2 Logical view

The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

- Map(k1,v1) -¿ list(k2,v2)

The Map function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

- Reduce(k2, list (v2)) -¿ list(v3)

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values. This behavior is different from the typical functional programming map and reduce combination, which accepts a list of arbitrary

values and returns one single value that combines all the values returned by map.

It is necessary but not sufficient to have implementations of the map and reduce abstractions in order to implement MapReduce. Distributed implementations of MapReduce require a means of connecting the processes performing the Map and Reduce phases. This may be a distributed file system. Other options are possible, such as direct streaming from mappers to reducers, or for the mapping processors to serve up their results to reducers that query them.

Chapter 4

Hadoop

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) inparallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. Hadoop was created by Doug Cutting who named it after his son's stuffed elephant. Hadoop is a top-level Apache project, being built and used by a community of contributors from all over the world. Yahoo! has been the largest contributor to the project and uses Hadoop extensively in its web search and advertising businesses. IBM and Google have announced a major initiative to use Hadoop to support university courses in distributed computer programming.

4.1 Architecture

Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop File System, or HDFS.

The user is required to tell the framework the following:

- The location(s) in the distributed file system of the job input
- The location(s) in the distributed file system for the job output
- The input format
- The output format
- The class containing the map function
- Optionally, the class containing the reduce function
- The JAR file containing the map and reduce functions and any support classes

On a fully configured cluster, "running Hadoop" means running a set of daemons, or resident programs. These daemons have specific roles; some exist only on one server, some exist across multiple servers. The daemons include:

- NameNode
- DataNode
- Secondary NameNode
- JobTracker
- TaskTracker

4.1.1 Namenode

The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks. The NameNode is the bookkeeper

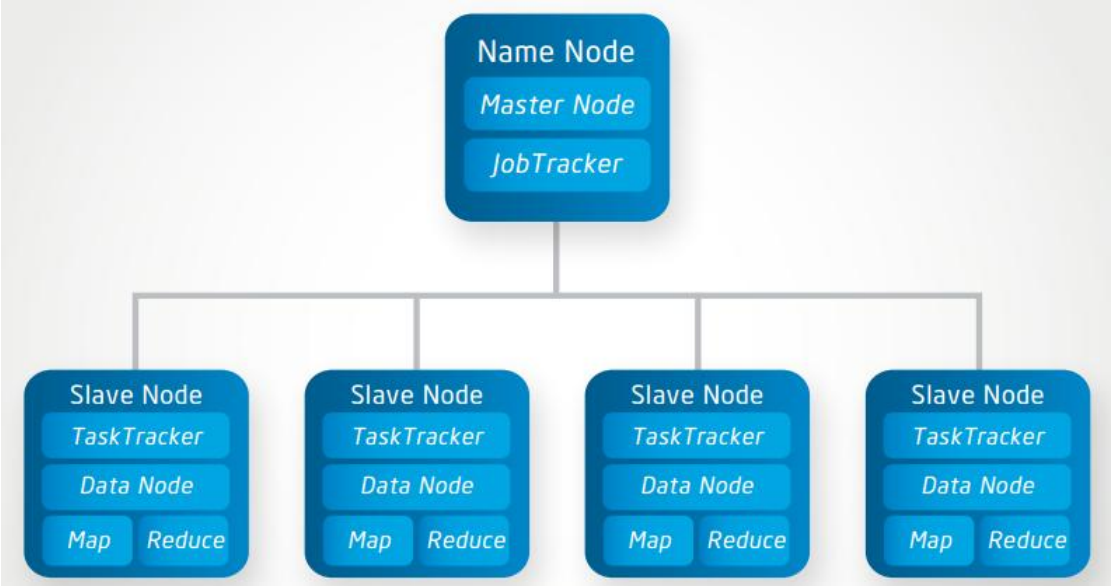


Figure 4.1: Hadoop Architecture

of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem.

4.1.2 Datanode

Each slave machine in the cluster will host a DataNode daemon to perform the grunt work of the distributed filesystem reading and writing HDFS blocks to actual files on the local filesystem. When it is required to read or write a HDFS file, the file is broken into blocks and the Namenode will tell the client which DataNode each block resides in. The client communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

4.1.3 Secondary NameNode

The Secondary NameNode (SNN) is an assistant daemon for monitoring the state of the cluster HDFS. Like the NameNode, each cluster has one SNN, and it typically resides on its own machine as well. The SNN differs from the NameNode in that this process doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration. The NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data.

4.1.4 JobTracker

The JobTracker daemon is the liaison between the application and Hadoop. Once the code is submitted to the cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster.

4.1.5 TaskTracker

TaskTrackers manage the execution of individual tasks on each slave node. Each TaskTracker is re- sponsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple JVMs to handle many map or reduce tasks in parallel. One responsibility of the TaskTracker is to constantly communicate with the JobTracker. If the JobTracker fails to receive a heartbeat

from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster.

4.1.6 HDFS

Hadoop can work directly with any mountable distributed file system, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS). It is a fault-tolerant distributed file system that is designed for commonly available hardware. It is well-suited for large data sets due to its high throughput access to application data.
HDFS has the following features :

1. Hadoop is designed to run on clusters of machines.
2. HDFS can handle large data sets.
3. Since HDFS deals with large scale data, it supports a multitude of machines.
4. HDFS provides a write-once-read-many access model.
5. HDFS is built using the Java language making it portable across various platforms.

4.2 Cluster Modes

There are three different modes of operation for Hadoop. They are differentiated on the basis of how the job is handled. The three different modes are given below:

- Single Mode Operation

- Pseudo-Distributed Mode Operation
- Multi-Node Operation

4.2.1 Single Mode Operation

Here each operation is considered as a single Java process. This mode is mainly used for demonstration of working of Hadoop using a single system. This mode involves just executing the Hadoop command with the arguments as needed for operation. This mode takes more time and consumes large space if used for running task involving huge amount of data.

4.2.2 Pseudo-Distributed Mode Operation

This mode runs on a single node or system. And here each Hadoop daemon runs in a separate Java process. This mode simulates the distributed processing in Hadoop with the help of a single system. But this mode takes much more time compared to fully distributed mode.

4.2.3 Multi-Node Operation

This mode of operation runs on a cluster of nodes and thus takes the advantage of distributed environment for processing massive amount of data at a very rapid rate. Here each node has its own memory and computation power. In this mode the master runs jobtracker, namenode and datanodes, while slaves runs only datanode and tasktracker.

4.3 Flow in Hadoop

A typical flow in Hadoop is a follows:

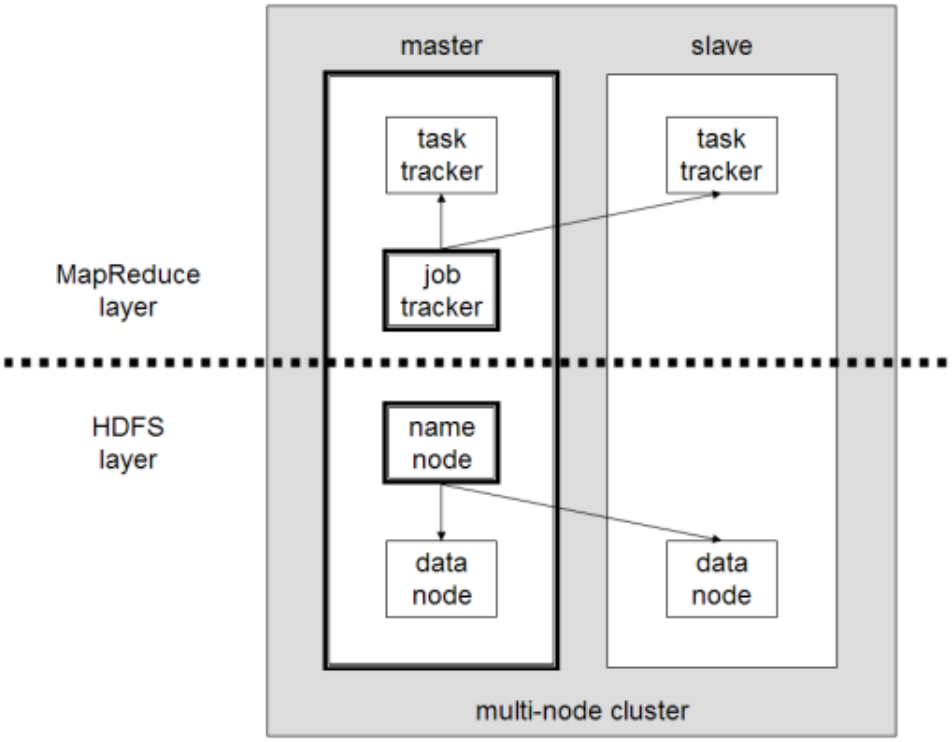


Figure 4.2: Multi-node Architecture

1. A Client (usually a MapReduce program) submits a job to the JobTracker.

2. The JobTracker get information from the NameNode on the location of the data within the DataNodes. The JobTracker places the client program (usually a jar file along with the configuration file) in the HDFS. Once placed, JobTracker tries to assign tasks to TaskTrackers on the DataNodes based on data locality.

3. The TaskTracker takes care of starting the Map tasks on the DataNodes by picking up the client program from the shared location on the HDFS.

4. The progress of the operation is relayed back to the JobTracker by the TaskTracker.

5. On completion of the Map task an intermediate file is created on the local filesystem of the TaskTracker.

6. Results from Map tasks are then passed on to the Reduce task.

7. The Reduce tasks works on all data received from map tasks and writes the final output to HDFS.

8. After the task completes the intermediate data generated by the TaskTracker is deleted.

4.4 Who All Uses Hadoop

Application and organizations using Hadoop include:

1. A9.com - Amazon

   - Build Amazon's product search indices using the streaming API and pre-existing C++, Perl, and Python tools.
   - Process millions of sessions daily for analytics, using both the Java and streaming APIs.
   - Clusters vary from 1 to 100 nodes.

2. Adobe

   - Use Apache Hadoop and Apache HBase in several areas from social services to structured data storage and processing for internal use.
   - Currently have about 30 nodes running HDFS, Hadoop and HBase in clusters ranging from 5 to 14 nodes on both production and development. We plan a deployment on an 80 nodes cluster.

- Constantly write data to Apache HBase and run MapReduce jobs to process then store it back to Apache HBase or external systems.

3. EBay
   - 532 nodes cluster (8 * 532 cores, 5.3PB).
   - Heavy usage of Java MapReduce, Apache Pig, Apache Hive, Apache HBase.
   - Using it for Search optimization and Research.
4. Facebook
   - Use Apache Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning.
   - Currently have 2 major clusters:
     - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
     - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
     - Each (commodity) node has 8 cores and 12 TB of storage.
     - Heavy users of both streaming as well as the Java APIs. We have built a higher level data warehousing framework using these features called Hive.
5. Google
   - Used for development projects
   - MapReduce jobs written in Groovy use Apache Hadoop Java APIs

6. IBM

    - Blue Cloud Computing Clusters
    - Enhances this open source technology to withstand the demands of their enterprise, adding administrative, discovery, development, provisioning, and security features, along with best-in-class analytical capabilities from IBM Research.

7. Twitter

    - Use Apache Hadoop to store and process tweets, log files, and many other types of data generated across Twitter. We store all data as compressed LZO files.
    - Use both Scala and Java to access Hadoop's MapReduce APIs.
    - Use Apache Pig heavily for both scheduled and ad-hoc jobs, due to its ability to accomplish a lot with few statements.

8. Yahoo!

    - More than 100,000 CPUs in ¿40,000 computers running Hadoop
    - biggest cluster 4500 nodes (2*4cpu boxes 4*1TB disk, 16GB RAM)
    - Used to support research for Ad Systems and Web Search
    - Also used to do scaling tests to support development of Apache Hadoop on larger clusters
    - greater than 60% of Hadoop Jobs within Yahoo are Apache Pig jobs

Chapter 5

Streaming Twitter Data

Social media has gained immense popularity with marketing teams, and Twitter is an effective tool for a company to get people excited about its products. Twitter makes it easy to engage users and communicate directly with them, and in turn, users can provide word-of-mouth marketing for companies by discussing the products. Given limited resources, and knowing we may not be able to talk to everyone we want to target directly, marketing departments can be more efficient by being selective about whom we reach out to.

Twitter Data or tweets can be streamed directly to HDFS and can be processed using Hadoop Map-Reduce. The streaming of tweets is done using Apache Flume.

5.1 Apache Flume

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.
Flume lets Hadoop users ingest high-volume streaming data into HDFS for storage. Specifically, Flume allows users to:

- Stream data : Ingest streaming data from multiple sources into Hadoop for storage and analysis
- Insulate systems : Buffer storage platform from transient spikes, when the rate of incoming data exceeds the rate at which data can be written to the destination
- Guarantee data delivery : Flume NG uses channel-based transactions to guarantee reliable message delivery. When a message moves from one agent to another, two transactions are started, one on the agent that delivers the event and the other on the agent that receives the event. This ensures guaranteed delivery semantics
- Scale horizontally : To ingest new data streams and additional volume as needed

Enterprises use Flume's powerful streaming capabilities to land data from high-throughput streams in the Hadoop Distributed File System (HDFS). Typical sources of these streams are application logs, sensor and machine data, geo-location data and social media. These different types of data can be landed in Hadoop for future analysis using interactive queries in Apache Hive. Or they can feed business dashboards served ongoing data by Apache HBase.

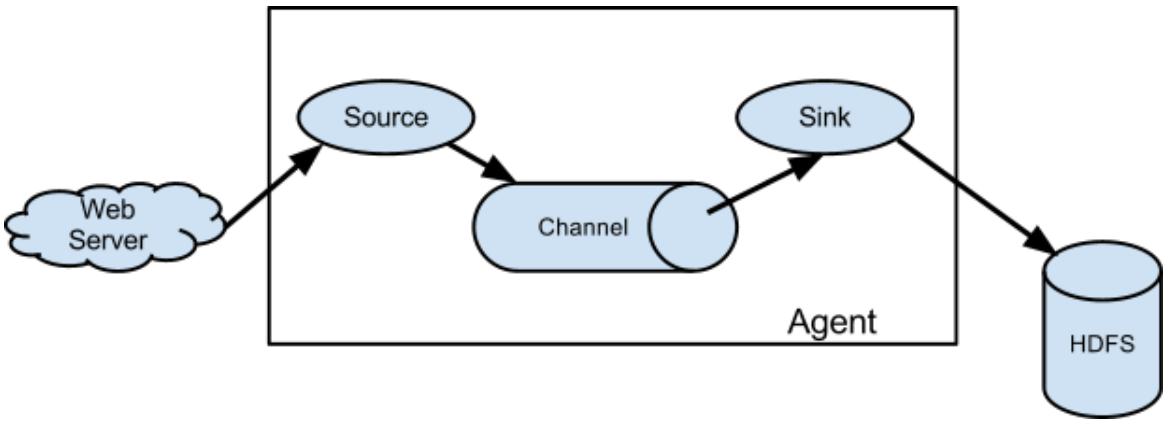


Figure 5.1: Data Flow model in flume

5.1.1 System Requirements

- Java Runtime Environment - Java 1.6 or later (Java 1.7 Recommended)
- Memory - Sufficient memory for configurations used by sources, channels or sinks
- Disk Space - Sufficient disk space for configurations used by channels or sinks
- Directory Permissions - Read/Write permissions for directories used by agent

5.1.2 Architecture

Flume's high-level architecture is built on a streamlined codebase that is easy to use and extend. The project is highly reliable, without the risk of data loss. Flume also supports dynamic reconfiguration without the need for a restart, which reduces downtime for its agents.The following components make up Apache Flume:

- Event : A singular unit of data that is transported by Flume (typically a single log entry)
- Source : The entity through which data enters into Flume. Sources either actively poll for data or passively wait for data to be delivered to them. A variety of sources allow data to be collected, such as log4j logs and syslogs
- Sink : The entity that delivers the data to the destination. A variety of sinks allow data to be streamed to a range of destinations. One example is the HDFS sink that writes events to HDFS
- Channel : The conduit between the Source and the Sink. Sources ingest events into the channel and the sinks drain the channel
- Agent : Any physical Java virtual machine running Flume. It is a collection of sources, sinks and channels
- Client : The entity that produces and transmits the Event to the Source operating within the Agent

Flume components interact in the following way:

1. A flow in Flume starts from the Client.
2. The Client transmits the Event to a Source operating within the Agent.
3. The Source receiving this Event then delivers it to one or more Channels.
4. One or more Sinks operating within the same Agent drains these Channels.

5. Channels decouple the ingestion rate from drain rate using the familiar producer-consumer model of data exchange.

6. When spikes in client side activity cause data to be generated faster than can be handled by the provisioned destination capacity can handle, the Channel size increases. This allows sources to continue normal operation for the duration of the spike.

7. The Sink of one Agent can be chained to the Source of another Agent. This chaining enables the creation of complex data flow topologies.

Because Flume's distributed architecture requires no central coordination point. Each agent runs independently of others with no inherent single point of failure, and Flume can easily scale horizontally.

5.1.3 Reliability

The events are staged in a channel on each agent. The events are then delivered to the next agent or terminal repository (like HDFS) in the flow. The events are removed from a channel only after they are stored in the channel of next agent or in the terminal repository. This is a how the single-hop message delivery semantics in Flume provide end-to-end reliability of the flow.

Flume uses a transactional approach to guarantee the reliable delivery of the events. The sources and sinks encapsulate in a transaction the storage/retrieval, respectively, of the events placed in or provided by a transaction provided by the channel. This ensures that the set of events are reliably passed from point to point in the flow. In the case of a multi-hop flow, the sink from the previous hop and the source from the next hop both

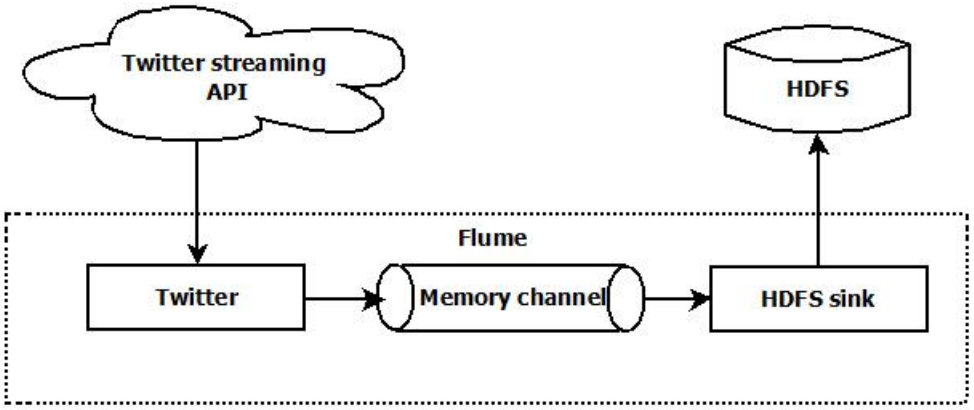


Figure 5.2: Control Flow of the Twitter Data

have their transactions running to ensure that the data is safely stored in the channel of the next hop.

Chapter 6

System Modeling

System model illustrates various diagrams such as use case diagram, activity diagram. It helps to analyse the details of the system in a simple manner. These diagrams provides a clear picture about working of the whole system.

6.1 UML Diagram

The UML is a language for visualizing, specifying, constructing and documenting the artifacts of software intensive systems. UML diagrams are one of the voccabulary of the diagram.

6.1.1 Use case Diagram

A use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system. Fig 4.1 shows use case diagram of this system.

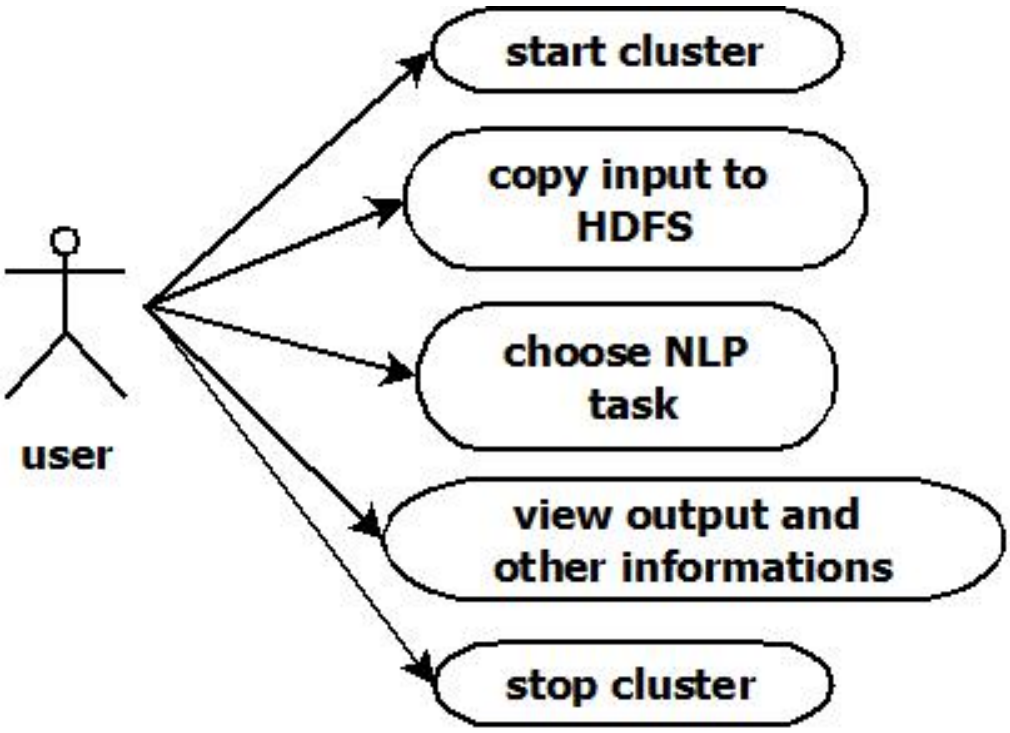


Figure 6.1: Use case diagram

Chapter 7

Implementation Model

The system basically does the NLP tasks in a fast and efficient way.

7.1 Algorithm

The basic algorithm is as follows :

1. Start cluster.
2. Get the input from the user.
3. Copy the input to HDFS.
4. Send the input to map function.
5. Split the input into tokens.
6. Perform the desired NLP tasks.
7. Output corresponding (key,value) pairs from mapper and input it to the reducer.
8. Perform the reduce steps and determine the count of each word.

9. Output the corresponding (key,value) pairs from reducer.

10. Retreive the output from hdfs and display it to the user.

In the map function, the desired NLP task is done for each word. The reducer collects these words, combines them and then finds the word count of each word. So the word along with its count is given as output to the user.

Chapter 8

System Requirements

This chapter explains about the Hardware platform, Tools and Language.

8.1 Hardware Requirements

- Nodes : 3numbers
- Switch : 1number
- LAN cables : 3numbers
- Processor : Intel(R)Core(TM)i3
- Speed : 2.50 GHZ
- RAM : 2.00GB
- Hard disk : 320 GB

8.2 Software Requirements

8.2.1 Map-reduce

- Platform : Linux(Ubuntu 14.10)
- Framework : Hadoop-1.1.2
- Language : Java

8.2.2 Streaming Tweets

- Platform : Linux(Ubuntu 14.10)
- Framework : Hadoop-1.1.2
- Tool Used : Apache Flume-1.4.0
- Language : Python

Chapter 9

Input

Unstructured natural language text data is given as input. The user is given the option of choosing two types of input data:

1. text consisting of novels, movie reviews etc.
2. tweets

Hadoop map-reduce works well with big data ranging from gigabytes to terabytes.

Chapter 10

Output

Text after performing the chosen NLP task in a fast and efficient way along with the count of each word in the text.

Chapter 11

User Interface

The user interface of the system allows user to choose the required NLP tasks and to view the output and other information related to the map reduce job execution. The snapshots of the User Interface is given below:

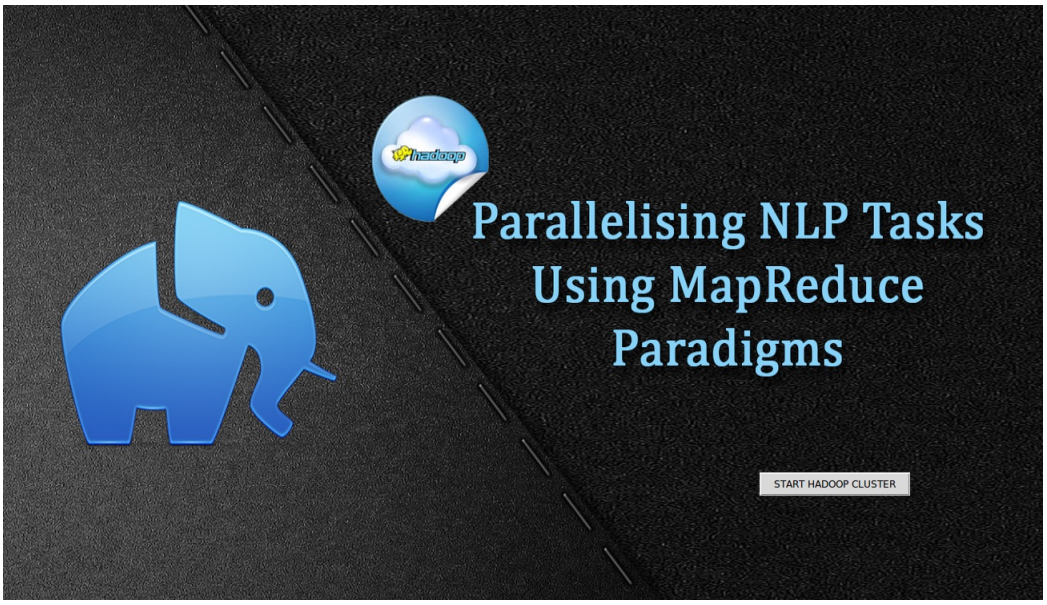


Figure 11.1: Screenshot 1

Figure 11.2: Screenshot 2

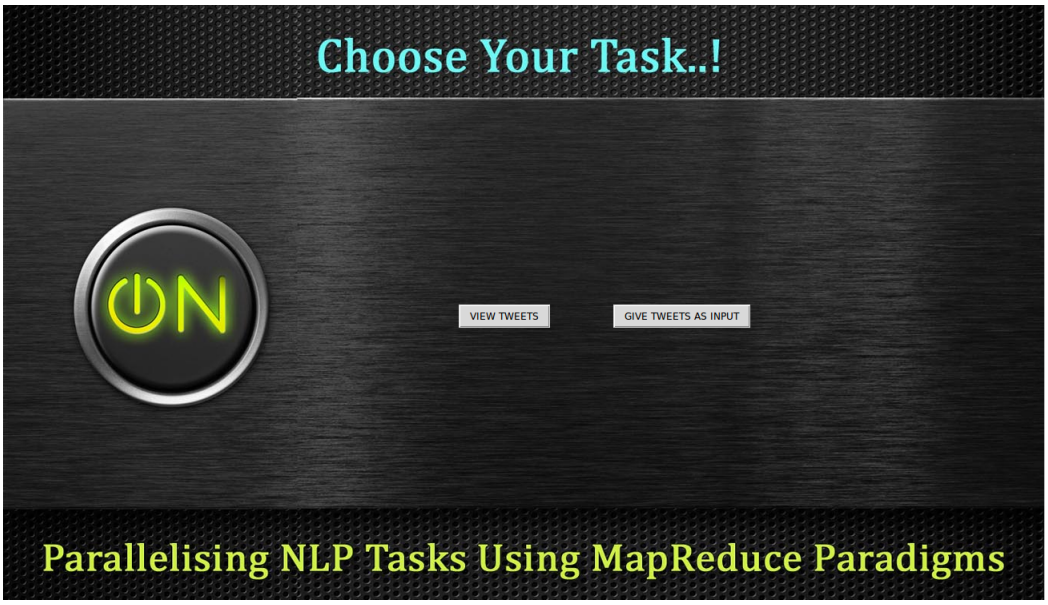


Figure 11.3: Screenshot 3

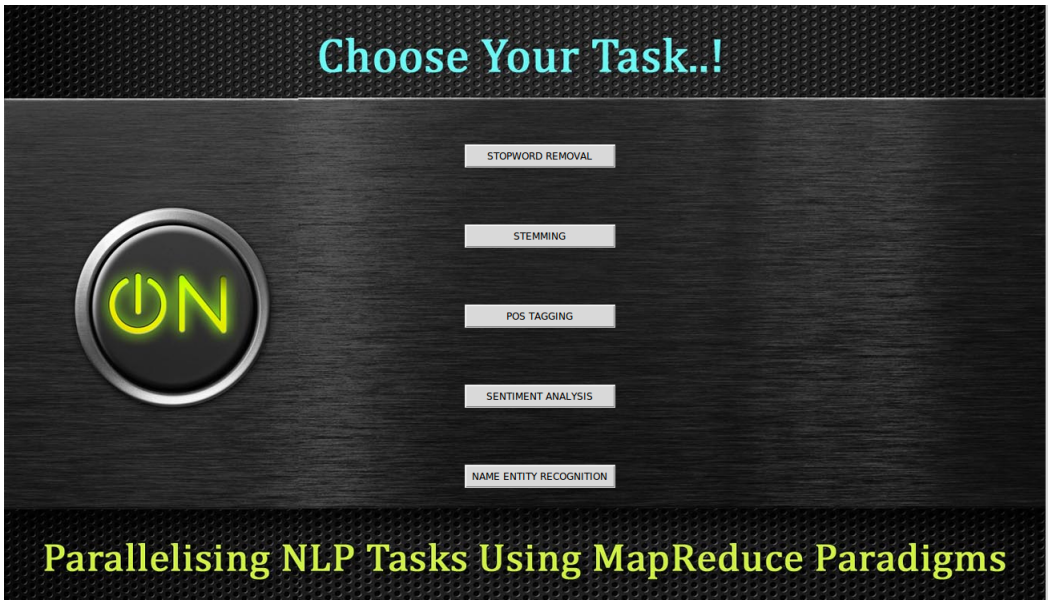


Figure 11.4: Screenshot 4

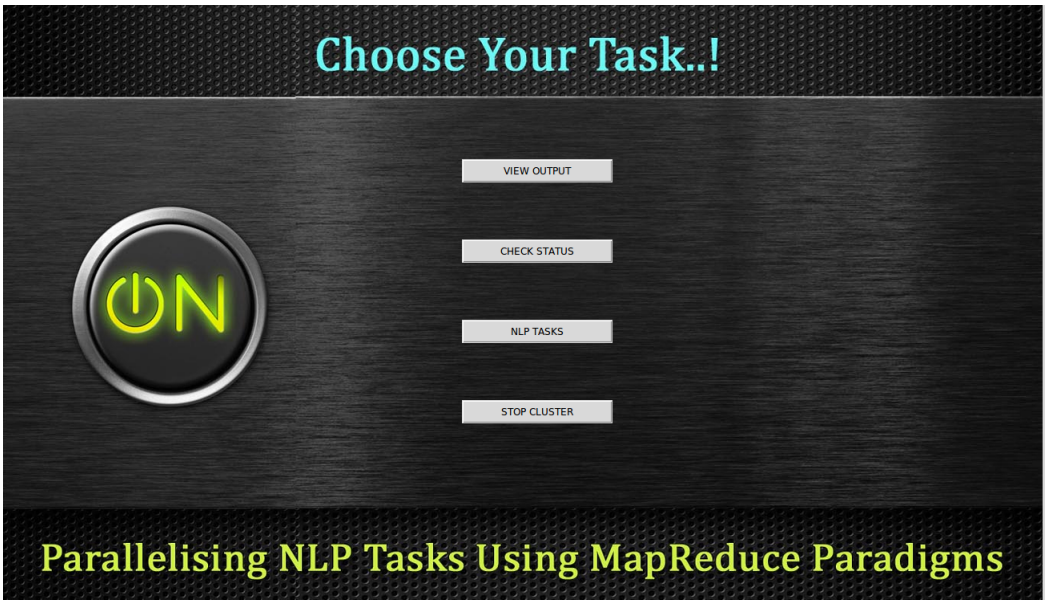


Figure 11.5: Screenshot 5

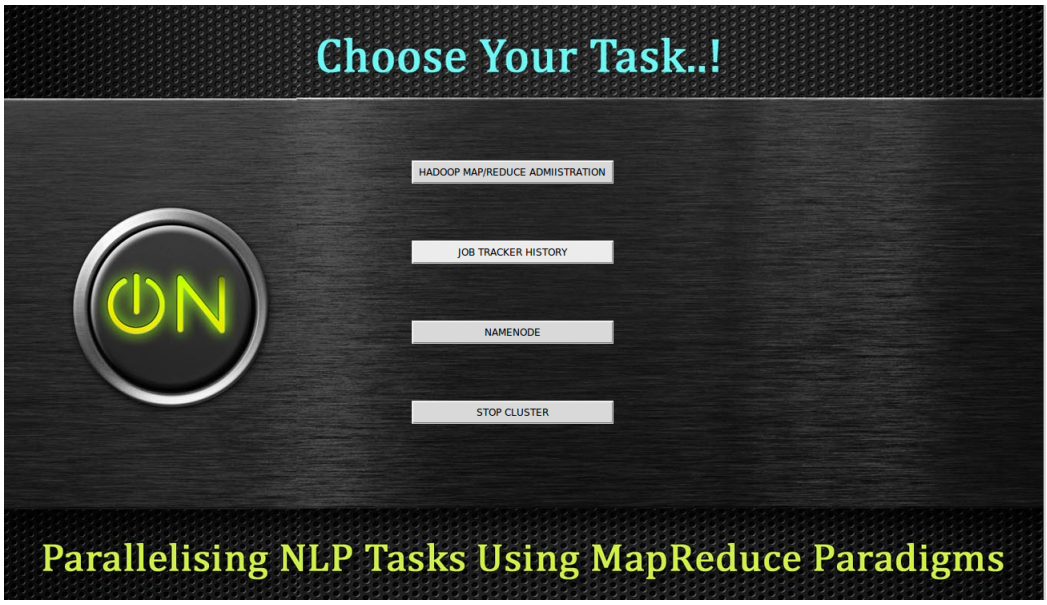


Figure 11.6: Screenshot 6

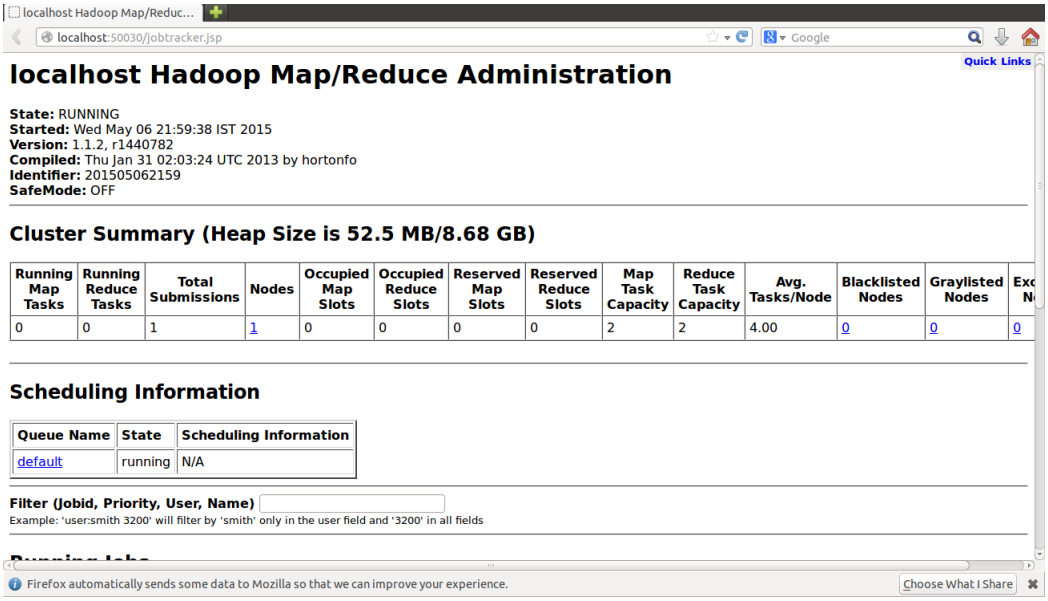


Figure 11.7: Screenshot 7

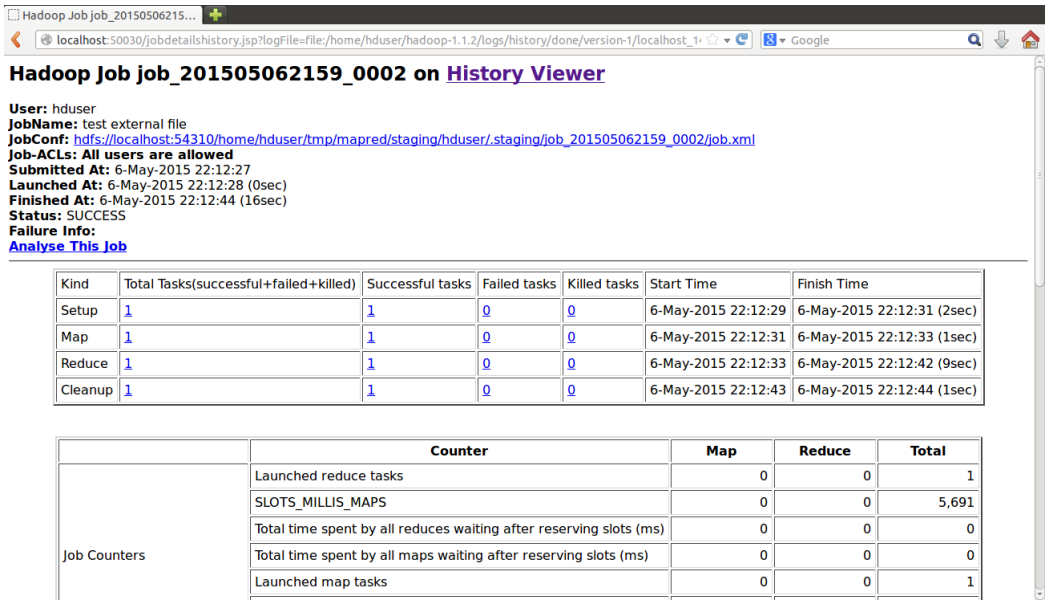


Figure 11.8: Screenshot 8

Figure 11.9: Screenshot 8

Chapter 12

Performance Evaluation

Evaluation is done over 10 input data ranging from kilobytes to gigabytes. The execution times of each cluster modes for Stemming is shown below:

Input Size	Eclipse	Single Node	Multinode
30KB	15s	18s	33s
90KB	16s	117s	19s
4MB	22s	22s	1min 17s
150MB	26m	3m 11s	2m 4s
544MB	30m	9m	8m 6s
1.8GB	1h 40m	24m 33s	16m 4s
2.4GB	2h	36m 38s	27m 9s
5.7GB	2h 40m	1h 20m	28m 4s

Table 12.1: Execution time for Stemming

The execution times of each cluster modes for Stopword Removal is shown below:

Input Size	Single Node	Multinode
30KB	13s	15s
90KB	13s	14s
355KB	15s	16s
4MB	15s	16s
38MB	49s	27s
150MB	42s	30s
544MB	2min 12s	1min 13s
804MB	4min 30s	1min 10s
1.8GB	6min 8s	2min 26s
2.4GB	9min 3s	3min 31sec
5.0GB	20min 10s	7min 12s
11.3GB	42min 24s	14min 59s

Table 12.2: Execution time for Stopword Removal

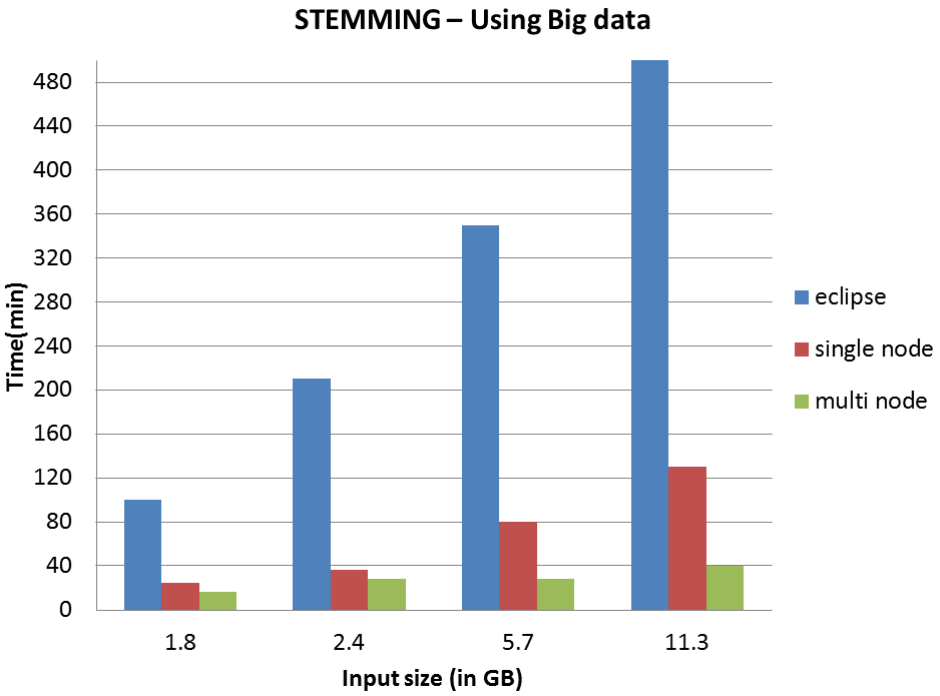


Figure 12.1: Performance of stemming map-reduce program in different cluster modes for large input size

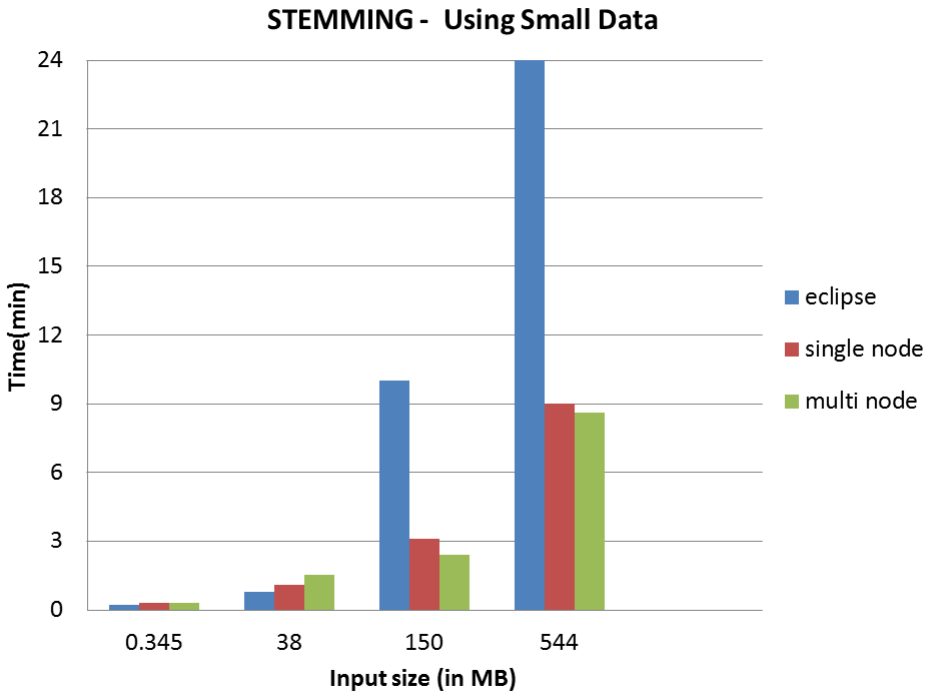


Figure 12.2: Performance of stemming map-reduce program in different cluster modes for small input size

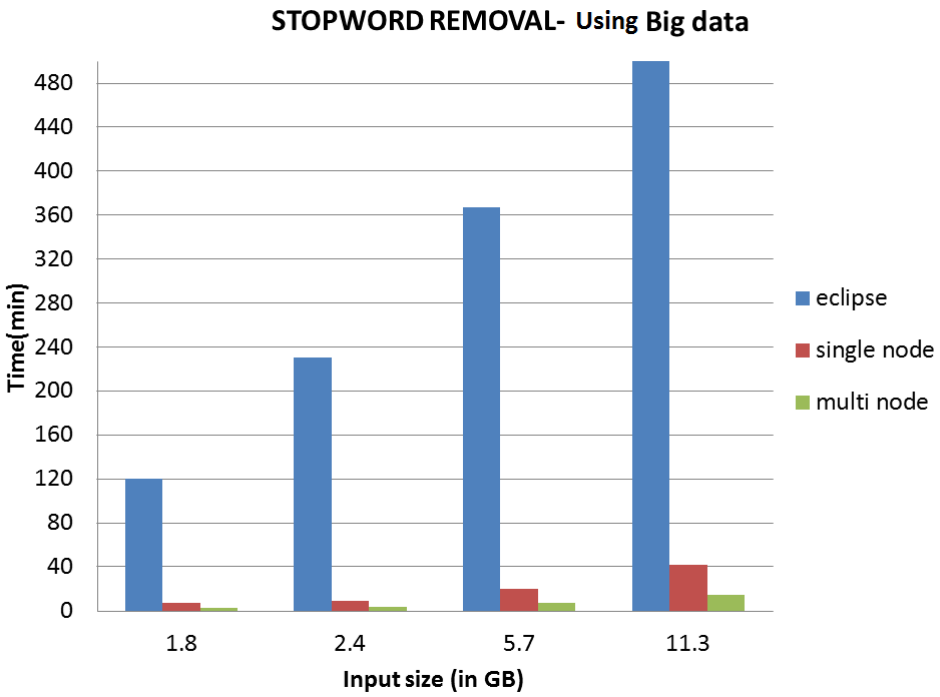


Figure 12.3: Performance of stopword removal map-reduce program in different cluster modes for large input size

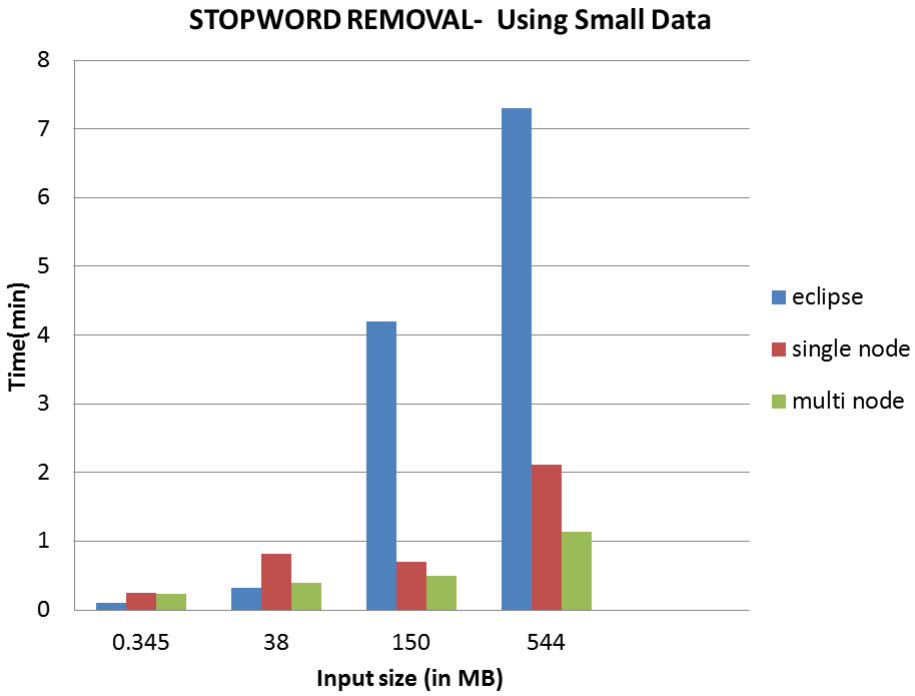


Figure 12.4: Performance of stopword removal map-reduce program in different cluster modes for small input size

We can deduce from the above results that multinode cluster works better than single node cluster as well as eclipse IDE for the same code. The time taken to execute a map reduce program in multinode is way less than single node. This implies that parallelising of tasks takes place among the nodes and that too in a fast and efficient way.

Chapter 13

Drawbacks and Future Work

13.1 Drawbacks

The drawbacks of the system are:

- In a multi-node cluster, should some nodes be faster than others, such nodes will perform and finish quicker, while the cluster waits for slow nodes to complete their tasks before presenting the output to the user.
- Hadoop tries to tackle this issue with its straggler mitigating mechanism.
- But straggler mitigating scheme is inefficient.

13.2 Future Works

Future Work include integrating Spark with Hadoop.

- Apache Spark is an execution engine that broadens the type of computing workloads Hadoop can handle, while also tuning the performance of the big data framework.Rather than just processing a batch of stored

data after the fact, as is the case with MapReduce, Spark can also manipulate data in real time using Spark Streaming.

- Rather than just processing a batch of stored data after the fact, as is the case with MapReduce, Spark can also manipulate data in real time using Spark Streaming.

- Spark is able to execute batch-processing jobs between 10 to 100 times faster than the MapReduce engine .

Chapter 14

Conclusion

The need to process enormous quantities of data has never been greater. Not only are terabyte- and petabyte-scale datasets rapidly becoming commonplace, but there is consensus that great value lies buried in them, waiting to be unlocked by the right computational tools. Here, map-reduce implementations of some important NLP tasks were proposed. The processing of big data in multinode was found to be much faster than both single node and eclipse ide for the same input size and code. As the number of nodes in the cluster increases, the processing speed minimizes. MapReduce can be exploited to solve a variety of problems related to text processing at scales that would have been unthinkable a few years ago. By scaling "out" with commodity servers, we were able to economically bring large clusters of machines to bear on problems of interest.

References

[1] Ralf Lammel, Data Programmability Team, Microsoft Corp, Redmond, WA, USA "Google's MapReduce Programming Model".

[2] Chuck Lam, "Hadoop in Action". *Manning publications*

[3] Daniel Jurafsky and James H.Martin, "Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". *Prentice Hall, Englewood Cliffs publications*

[4] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters." *Google Research Publication.*

[5] "Running Hadoop on Ubuntu Linux (Single-Node Cluster)" *http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/ , visited on october 2014.*

[6] "Running Hadoop on Ubuntu Linux (Multi-Node Cluster)" *http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/ , visited on march 2015.*

[7] Jon Natkins, "How-to: Analyze Twitter Data with Apache Hadoop." *http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/ , visited on march 2015.*

[8] De Campos, T.E., Babu, B.R., Varma. M, "Hadoop" *http://hadoop.apache.org , visited on September, 2014.*

[9] "Apache Spark", *https://spark.apache.org/ , visited on february, 2015.*

[10] Writing an Hadoop MapReduce Program in Python" *http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/ , visited on december 2014* .

Appendix

A.1 Hadoop Single Node Setup

Prerequisites

Hadoop requires a working Java 1.6+ installation. Commands to run for the installation are:

- *sudo apt-get install python-software-properties*
- *sudo add-apt-repository ppa:ferramroberto/java*
- *sudo apt-get update*
- *sudo apt-get install sun-java7-jdk*

Adding a dedicated Hadoop system user

Add a dedicated user so that it helps to separate the Hadoop installation from other software applications and user accounts running on the same machine

- *sudo addgroup hadoop*
- *sudo adduser –ingroup hadoop hduser*

This will add the user hduser and the group hadoop to our local machine.

Configuring SSH

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus your local machine if you want to use Hadoop on it. First, we have to generate an SSH key for the hduser user.

- *su - hduser*
- *ssh-keygen -t rsa -P ""*

The second line will create an RSA key pair with an empty password. You have to enable SSH access to your local machine with this newly created key.

- $cat /home/hduser/.ssh/id_rsa.pub >> /home/hduser/.ssh/authorized_keys$

The final step is to test the SSH setup by connecting to your local machine with the hduser user. The step is also needed to save your local machine's host key fingerprint to the hduser user's known hosts file.

- *ssh localhost*

The teriminal will display the below message:
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44:f9:36:26.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Linux ubuntu 2.6.32-22-generic 33-Ubuntu SMP Wed Apr 28 13:27:30 UTC 2010 i686 GNU/Linux Ubuntu 10.04 LTS

Disabling IPv6

To disable IPv6 on Ubuntu 10.04 LTS, open /etc/sysctl.conf in the editor of your choice and add the following lines to the end of the file:

- *net.ipv6.conf.all.disable_ipv6 = 1*
  net.ipv6.conf.default.disable_ipv6 = 1
  net.ipv6.conf.lo.disable_ipv6 = 1

Hadoop

Download Hadoop from the Apache Download Mirrors and extract the contents of the Hadoop package to a ocation of your choice. I picked /usr/local/hadoop. Make sure to change the owner of all the files to the hduser user and hadoop.

- *cd /usr/local*
- *sudo tar xzf hadoop-1.0.3.tar.gz*
- *sudo mv hadoop-1.0.3 hadoop*
- *sudo chown -R hduser:hadoop hadoop*

Update $HOME/.bashrc

Add the following lines to the end of the $HOME/.bashrc file of user hduser. If you use a shell other than bash, you should of course update its appropriate configuration files instead of .bashrc

- *export HADOOP_HOME=/usr/local/hadoop*
- *export JAVA_HOME=/usr/lib/jvm/java-7-sun*
- *export PATH=$PATH:$HADOOP_HOME/bin*

Configuration

hadoop-env.sh

The only required environment variable we have to configure for Hadoop in this tutorial is JAVA_HOME. Open conf/hadoop-env.sh in the editor of your choice and set the JAVA_HOME environment variable to the Sun JDK/JRE 7 directory.

- *export JAVA_HOME=/usr/lib/jvm/java-7-sun*

conf/*-site.xml

We will configure the directory where Hadoop will store its data files, the network ports it listens to, etc. Our setup will use Hadoop's Distributed File System, HDFS, even though our little "cluster" only contains our single local machine.
Now we create the directory and set the required ownerships and permissions:

- *sudo mkdir -p /app/hadoop/tmp*
- *sudo chown hduser:hadoop /app/hadoop/tmp*
- *sudo chmod 750 /app/hadoop/tmp*

Add the following snippets between the ¡configuration¿ ... ¡/configuration¿ tags in the respective configuration XML file.

In file conf/core-site.xml:

< property >
< name > hadoop.tmp.dir < /name >

$< value > /app/hadoop/tmp < /value >$

$< description > Abaseforothertemporarydirectories. < /description >$

$< /property >$

$< property >$

$< name > fs.default.name < /name >$

$< value > hdfs : //localhost : 54310 < /value >$

$< property >$

In file conf/mapred-site.xml:

$< property >$

$< name > mapred.job.tracker < /name >$

$< value > localhost : 54311 < /value >$

$< description >$*The host and port that the MapReduce job tracker runs*

at. If "local", then jobs are run in-process as a single map

and reduce task.

$< /description >$

$< /property >$

In file conf/hdfs-site.xml:

$< property >$

$< name > dfs.replication < /name >$

$< value > 1 < /value >$

$< description > Defaultblockreplication.$

The actual number of replications can be specified when the file is created.

The default is used if replication is not specified in create time.

< /description >

¡/property¿

Formatting the HDFS filesystem via the NameNode

To format the filesystem (which simply initializes the directory specified by the dfs.name.dir variable), run the command:

- */usr/local/hadoop/bin/hadoop namenode -format*

Starting your single-node cluster

Run the command:

- */usr/local/hadoop/bin/start-all.sh*

This will startup a Namenode, Datanode, Jobtracker and a Tasktracker on your machine. A nifty tool for checking whether the expected Hadoop processes are running is jps.

- *jps*

  2287 TaskTracker

  2149 JobTracker

  1938 DataNode

  2085 SecondaryNameNode

  2349 Jps

  1788 NameNode

You can also check with netstat if Hadoop is listening on the configured ports.

- *sudo netstat -plten — grep java*

  tcp 0 0 0.0.0.0:50070 0.0.0.0: LISTEN 1001 9236 2471/java*

  tcp 0 0 0.0.0.0:50010 0.0.0.0: LISTEN 1001 9998 2628/java*

  tcp 0 0 0.0.0.0:48159 0.0.0.0: LISTEN 1001 8496 2628/java*

  tcp 0 0 0.0.0.0:53121 0.0.0.0: LISTEN 1001 9228 2857/java*

  tcp 0 0 127.0.0.1:54310 0.0.0.0: LISTEN 1001 8143 2471/java*

  tcp 0 0 127.0.0.1:54311 0.0.0.0: LISTEN 1001 9230 2857/java*

  tcp 0 0 0.0.0.0:59305 0.0.0.0: LISTEN 1001 8141 2471/java*

  tcp 0 0 0.0.0.0:50060 0.0.0.0: LISTEN 1001 9857 3005/java*

  tcp 0 0 0.0.0.0:49900 0.0.0.0: LISTEN 1001 9037 2785/java*

  tcp 0 0 0.0.0.0:50030 0.0.0.0: LISTEN 1001 9773 2857/java*

Stopping your single-node cluster

Run the command:

- */usr/local/hadoop/bin/stop-all.sh*

  stopping jobtracker

  localhost: stopping tasktracker

  stopping namenode

  localhost: stopping datanode

  localhost: stopping secondarynamenode

to stop all the daemons running on your machine.

Running a MapReduce job

Restart the Hadoop cluster

Restart your Hadoop cluster if it's not running already.

Copy local example data to HDFS

Before we run the actual MapReduce job, we first have to copy the input files from our local file system to Hadoop's HDFS.

- */usr/local/hadoop$ bin/hadoop dfs -copyFromLocal /tmp/gutenberg /user/hduser/guten*
- */usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser*
  Found 1 items
  drwxr-xr-x - hduser supergroup 0 2010-05-08 17:40 /user/hduser/gutenberg
  hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser/gutenberg
  Found 3 items
  -rw-r–r– 3 hduser supergroup 674566 2011-03-10 11:38
  /user/hduser/gutenberg/pg20417.txt
  -rw-r–r– 3 hduser supergroup 1573112 2011-03-10 11:38
  /user/hduser/gutenberg/pg4300.txt
  -rw-r–r– 3 hduser supergroup 1423801 2011-03-10 11:38
  /user/hduser/gutenberg/pg5000.txt

considering the input is in /tmp/gutenberg/

Run the MapReduce job

Now, we actually run the WordCount example job.

- */usr/local/hadoop$ bin/hadoop jar hadoop*examples*.jar wordcount /user/hduser/gutenberg /user/hduser/gutenberg-output*

This command will read all the files in the HDFS directory /user/hduser/gutenberg, process it, and store the result in the HDFS directory /user/hduser/gutenberg-output.
Example output of the previous command in the console:

*hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar hadoop*examples*.jar wordcount*
/user/hduser/gutenberg /user/hduser/gutenberg-output
10/05/08 17:43:00 INFO input.FileInputFormat: Total input paths to process : 3
10/05/08 17:43:01 INFO mapred.JobClient: Running job: job_201005081732_0001
10/05/08 17:43:02 INFO mapred.JobClient: map 0% reduce 0%
10/05/08 17:43:14 INFO mapred.JobClient: map 66% reduce 0%
10/05/08 17:43:17 INFO mapred.JobClient: map 100% reduce 0%
10/05/08 17:43:26 INFO mapred.JobClient: map 100% reduce 100%
10/05/08 17:43:28 INFO mapred.JobClient: Job complete: job_201005081732_0001
10/05/08 17:43:28 INFO mapred.JobClient: Counters: 17
10/05/08 17:43:28 INFO mapred.JobClient: Job Counters
10/05/08 17:43:28 INFO mapred.JobClient: Launched reduce tasks=1
10/05/08 17:43:28 INFO mapred.JobClient: Launched map tasks=3
10/05/08 17:43:28 INFO mapred.JobClient: Data-local map tasks=3
10/05/08 17:43:28 INFO mapred.JobClient: FileSystemCounters
10/05/08 17:43:28 INFO mapred.JobClient: FILE_BYTES_READ=2214026
10/05/08 17:43:28 INFO mapred.JobClient: HDFS_BYTES_READ=3639512
10/05/08 17:43:28 INFO mapred.JobClient: FILE_BYTES_WRITTEN=3687918

10/05/08 17:43:28 INFO mapred.JobClient: HDFS_BYTES_WRITTEN=880330
10/05/08 17:43:28 INFO mapred.JobClient: Map-Reduce Framework
10/05/08 17:43:28 INFO mapred.JobClient: Reduce input groups=82290

10/05/08 17:43:28 INFO mapred.JobClient: Combine output records=102286
10/05/08 17:43:28 INFO mapred.JobClient: Map input records=77934
10/05/08 17:43:28 INFO mapred.JobClient: Reduce shuffle bytes=1473796
10/05/08 17:43:28 INFO mapred.JobClient: Reduce output records=82290
10/05/08 17:43:28 INFO mapred.JobClient: Spilled Records=255874
10/05/08 17:43:28 INFO mapred.JobClient: Map output bytes=6076267
10/05/08 17:43:28 INFO mapred.JobClient: Combine input records=629187
10/05/08 17:43:28 INFO mapred.JobClient: Map output records=629187
10/05/08 17:43:28 INFO mapred.JobClient: Reduce input records=102286

Check if the result is successfully stored in HDFS directory /user/hduser/gutenberg-output:

- */usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser*
  Found 2 items
  drwxr-xr-x - hduser supergroup 0 2010-05-08 17:40 /user/hduser/gutenberg
  drwxr-xr-x - hduser supergroup 0 2010-05-08 17:43 /user/hduser/gutenberg-output
  hduser@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls /user/hduser/gutenberg-output
  Found 2 items
  drwxr-xr-x - hduser supergroup 0 2010-05-08 17:43 /user/hduser/gutenberg-output/_logs
  -rw-r–r– 1 hduser supergroup 880802 2010-05-08 17:43 /user/hduser/gutenberg-

output/part-r-00000

Retrieve the job result from HDFS

To inspect the file, you can copy it from HDFS to the local file system. Alternatively, you can use the command.

- */usr/local/hadoop$ bin/hadoop dfs -cat /user/hduser/gutenberg-output/part-r-00000*

to read the file directly from HDFS without copying it to the local file system.
In this tutorial, we will copy the results to the local file system though.

Hadoop Web Interfaces

Hadoop comes with several web interfaces which are by default (see conf/hadoop-default.xml) available at these locations:

- *http://localhost:50070/ - web UI of the NameNode daemon*
- *http://localhost:50030/ - web UI of the JobTracker daemon*
- *http://localhost:50060/ - web UI of the TaskTracker daemon*

Running Hadoop on Ubuntu Linux (Multi-Node Cluster)

Configuring single-node clusters first

First we need to configure the single node cluster to function the multinode cluster. Follow the steps as mentioned earlier in appendix A.1.

Networking

The easiest is to put both machines in the same network with regard to hardware and software configuration, for example connect both machines via a single hub or switch and configure the network interfaces to use a common network such as 192.168.0.x/24. To make it simple, IP address 192.168.0.1 is assigned to the master machine and 192.168.0.2 to the slave machine. Update /etc/hosts on both machines with the following lines:

- *192.168.0.1 master*
- *192.168.0.2 slave*

SSH access

The hduser user on the master (aka hdusermaster) must be able to connect a) to its own user account on the master - i.e. ssh master in this context and not necessarily ssh localhost - and b) to the hduser user account on the slave (aka hduser@slave) via a password-less SSH login. As followed in the configuration of single-node cluster, you just have to add the hduser-master's public SSH key to the authorized_keys file of hduserslave (in this

user's $HOME/.ssh/authorized_keys). You can do this manually or use the following SSH command:

- *hduser@master: $ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduserslave*

This command will prompt you for the login password for user hduser on slave, then copy the public SSH key for you, creating the correct directory and fixing the permissions as necessary.

The final step is to test the SSH setup by connecting with user hduser from the master to the user account hduser on the slave. The step is also needed to save slave's host key fingerprint to the hdusermaster's known_hosts file.

So, connecting from master to master

- *ssh master*

  The authenticity of host 'master (192.168.0.1)' can't be established. RSA key fingerprint is 3b:21:b3:c0:21:5c:7c:54:2f:1e:2d:96:79:eb:7f:95. Are you sure you want to continue connecting (yes/no)? yes Warning: Permanently added 'master' (RSA) to the list of known hosts. Linux master 2.6.20-16-386 #2 Thu Jun 7 20:16:13 UTC 2007 i686

and from master to slave.

- *ssh slave*

  The authenticity of host 'slave (192.168.0.2)' can't be established. RSA key fingerprint is 74:d7:61:86:db:86:8f:31:90:9c:68:b0:13:88:52:72. Are you sure you want to continue connecting (yes/no)? yes Warning: Permanently added 'slave' (RSA) to the list of known hosts. Ubuntu 10.14

Configuration

On master, update conf/masters that it looks like this:

- *master*

The conf/slaves file lists the hosts, one per line, where the Hadoop slave daemons (DataNodes and TaskTrackers) will be run. We want both the master box and the slave box to act as Hadoop slaves because we want both of them to store and process data.

On master, update conf/slaves that it looks like this:

- *master*
- *slave*

If you have additional slave nodes, just add them to the conf/slaves file, one hostname per line.

- *master*
- *slave*
- *anotherslave01*
- *anotherslave02*
- *anotherslave03*

conf/*-site.xml (all machines)

You must change the configuration files conf/core-site.xml, conf/mapred-site.xml and conf/hdfs-site.xml on ALL machines as follows.

$< property >$

$< name > fs.default.name < /name >$

$< value > hdfs : //master : 54310 < /value >$
$< /property >$

Second, we have to change the mapred.job.tracker parameter (in conf/mapred-site.xml), which specifies the JobTracker (MapReduce master) host and port. Again, this is the master in our case.

$< property >$
$< name > mapred.job.tracker < /name >$
$< value > master : 54311 < /value >$
$< /property >$

Third, we change the dfs.replication parameter (in conf/hdfs-site.xml) which specifies the default block replication. It defines how many machines a single file should be replicated to before it becomes available. The default value of dfs.replication is 3. However, we have only two nodes available, so we set dfs.replication to 2.

$< property >$
$< name > dfs.replication < /name >$
$< value > 2 < /value >$
$< /property >$

Formatting the HDFS filesystem via the NameNode

Before we start our new multi-node cluster, we must format Hadoop's distributed filesystem (HDFS) via the NameNode. You need to do this the first time you set up an Hadoop cluster.

- *bin/hadoop namenode -format*

  ... INFO dfs.Storage: Storage directory /app/hadoop/tmp/dfs/name has been successfully formatted.

Starting the multi-node cluster

Starting the cluster is performed as follows in master.

- *hduser@master:/usr/local/hadoop$ bin/start-dfs.sh*

- *hduser@master:/usr/local/hadoop$ jps*

  14799 NameNode

  15314 Jps

  14880 DataNode

  14977 SecondaryNameNode