School of Electrical and Information Engineering

University of the Witwatersrand, Johannesburg

ELEN7046 - Software Technologies and Techniques

Individual Project Report

Big Data and Data Visualisation using Commodity Hardware and Open Source Software

Solution: Twit-Con-Pro

July 4th, 2016

Author: Gareth Stephenson (778919)

Group Members:

Matsobane Khwinana	779053
Sidwell Mokhemisa	1229756
Dave Cloete	1573016
Kyle Trehaeven	0602877N

https://github.com/garethstephenson/ELEN7046 Group2 2016

Abstract

The hardware and software used to construct a cost-effective cluster computing framework, which is capable of processing Big Data workloads, is presented. The system is designed to processes social media data related to American presidential elections and South African political party municipal elections for use by a data visualisation framework.

An overview of the hardware infrastructure based on Raspberry Pi 3 Model B’s, as well as the software solution created using Apache Spark and Scala, is discussed. In conclusion, the system, which is the combination of economically priced hardware and open source software, along with targeted custom software applications, demonstrates itself to be a viable Big Data solution for small businesses, start-ups and academia.

Table of Contents

Abstract i

List of Figures iii

List of Tables iii

1. Introduction 1

2. Background 1

3. Requirements 1

 3.1 Functional 1

 3.2 Non-functional 1

4. Approach 2

 4.1 Infrastructure Hardware 2

  4.1.1 Raspberry Pi 3 Model B 2

  4.1.2 TP-LINK 10/100Mbps Desktop Switches 2

  4.1.3 EZCOOL 450W Power Supply Unit 2

 4.2 Infrastructure Software 4

  4.2.1 Raspbian Linux 4

  4.2.2 Apache Spark 4

5. Design Overview of the Data Transformation Applications 5

 5.1 Apache Spark Cluster Computing Components 5

 5.2 Category Count per Hour Application 6

 5.3 Category Count per Day Application 9

6. Challenges 10

7. Recommendations 10

8. Conclusion 10

References 11

A Timesheet 12

B Apache Spark Installation & Configuration 13

C Cost of Hardware 15

D MicroSDHC Card Benchmarks 16

List of Figures

Figure 1: Raspberry Pi 3 infrastructure 2
Figure 2: Modified ATX PSU with switches 3
Figure 3: Raspberry Pi PSU 3
Figure 4: TP-LINK PSU 3
Figure 5: ATX PSU ratings 3
Figure 6: Apache Spark's cluster computing components 5
Figure 7: CategoryCountPerHour class 6
Figure 8: Tweet data sample in JSON format 7
Figure 9: Tweet class in Scala with accompanying HashTag class 7
Figure 10: DAG of CategoryCountPerHour program flow 8
Figure 11: CategoryCountPerHour JSON output sample 8
Figure 12: Input and output classes for processing CategoryCount JSON data 8
Figure 13: CategoryCountPerDay class 9
Figure 14: CategoryCountPerDay JSON output sample 9
Figure 15: MicroSDHC card benchmarks 16

List of Tables

Table 1: RAM use per system 4
Table 2: Combined RAM footprints 5
Table 3: Time to upload data sources 5
Table 4: Cost of hardware 15

1. Introduction

Data, in the age of modern computing, is being produced at an alarming, almost incomprehensible rate, and making sense of it all is the primary objective of the Big Data movement [1]. Seeking meaningful information from Big Data, whether by human analysis or machine learning algorithms, will reveal never-before seen insights into not only what is happening all around us, but why.

This report focuses on the data transformation process, from building the infrastructure and installing and configuring the cluster-computing framework, to writing the software that creates the output to be used by the data visualisation component of the solution.

2. Background

The objective of the project is to find a data source that matches the required field of interest selected by the group, to process that data into a form of information which will provide meaningful insight into that data, and to finally present the processed information via a visualisation framework so further examination can occur.

The field of interest selected by the group is related to elections happening in both the United States of America and South Africa. Attention is given primarily to the individuals campaigning within the USA, namely Donald Trump and Hillary Clinton. For the South African municipal elections, the areas of focus are the parties, namely the ANC, DA and the EFF. These identified subjects are used as the categories within the data transformation process.

Twitter was chosen as the data source for the group project, as it was felt this better portrayed the sentiment of the people, not just online and traditional media outlets. Also, the Twitter REST API proved easy to integrate with [2].

3. Requirements

The requirements of the data transformation process, both functional and non-functional, relating to hardware and software are listed below.

3.1 Functional

- Transform tweet data into the sum of categories per hour for any given day
- Transform the output from the above step into the sum of categories per day
- Transform the output from the above step into the total sum for each category
- Transform tweet data into the sum of positive and negative sentiment for each category for any given day
- Transform tweet data into a sum of commonly found words

3.2 Non-functional

- Build a viable Big Data processing solution using low-cost commodity hardware and open source software
- Purchase, configure and network a number of Raspberry Pi 3 Model B [3] devices
- Create a single power supply to run the Raspberry Pi's and network switches
- Install and configure cluster-computing framework software capable of on running on Raspberry Pi 3's
- Learn Scala in order to create the software to fulfil the functional requirements of the project

4. Approach

The data transformation process of the project is unique due to the dual outcomes expected it. Not only must the data transformation process transform tweet data into meaningful information for the visualisation framework to consume, it also needs to perform the data transformations on a platform that is affordable and powerful enough for it to be considered as a viable choice for small businesses, start-ups and academia. (See Appendix C)

The approach taken to build the hardware infrastructure was very exploratory in nature, as none of the members of the group had ever undertaken such a task before. A pragmatic approach to creating the data transformation software is employed, where each successive step leveraged the knowledge gained from the previous step.

4.1 Infrastructure Hardware

4.1.1 Raspberry Pi 3 Model B

Five Raspberry Pi 3 Model B units were selected to run the cluster computing framework infrastructure, cumulatively bringing 20 CPU cores and 5GB of RAM into the pool of available resources. Each Raspberry Pi 3 is equipped with a 32GB MicroSDHC, Class 10 specification card with a minimum speed rating of UHS-I U1 [4]. The master node was equipped with a UHS-I U3 MicroSD card for better performance on read operations, as it is also acting as the file server. (See Appendix D)

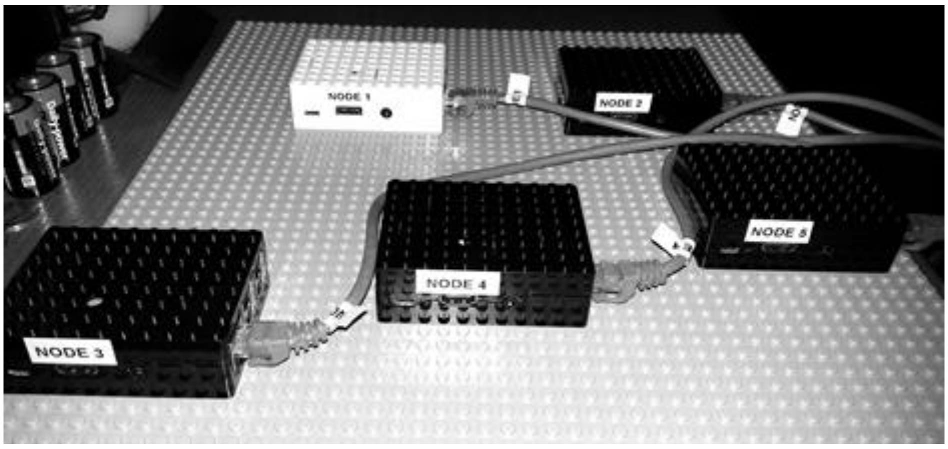


Figure 1: Raspberry Pi 3 infrastructure

4.1.2 TP-LINK 10/100Mbps Desktop Switches

Two 5-port, 10/100Mbps TP-LINK desktop switches (TL-SF1005D) were selected for connecting the Raspberry Pi's on a local area network (LAN). The speed of the desktop switches match the speed of the Raspberry Pi's on-board LAN module, so there is no need to invest in more expensive gigabit capable units.

4.1.3 EZCOOL 450W Power Supply Unit

A PC ATX 450W power supply unit (PSU) (JSP-450P08N) is used as a power source to drive the Raspberry Pi's as well as the desktop switches. All the devices are rated to run between 5V and 5.1V, with the Raspberry Pi's requiring a maximum of 2.5A and the desktop switches requiring a maximum of 0.6A each. The ATX PSU is rated as being able to deliver up to 40A at 5V, so it has more than enough capacity to power the entire infrastructure. The PSU was opened up and had all the 3.3V and 12V leads removed, leaving only the 5V leads along with accompanying ground leads. The connectors for PC devices and the PC motherboard were removed and discarded, and 5V micro-USB male B type cables were soldered on to the 5V leads. These would be used to power the Raspberry Pi's. The 1.4mm DC male power plugs from the original PSUs of the desktop switches were removed and soldered onto available 5V leads on the ATX PSU.

This configuration was agreed upon by the group as sufficient to serve as the base for the data transformation infrastructure hardware.

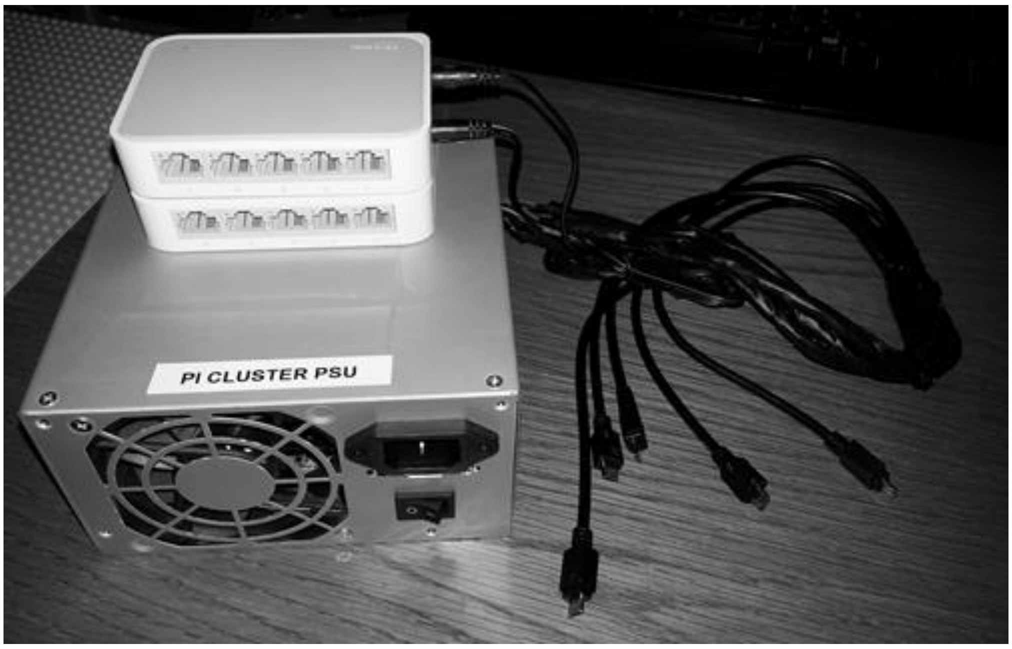
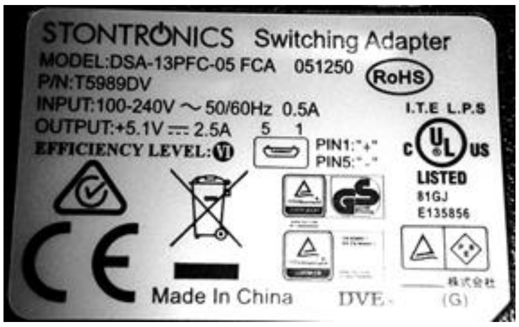
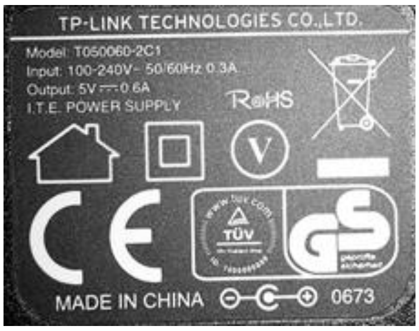


Figure 2: Modified ATX PSU with switches



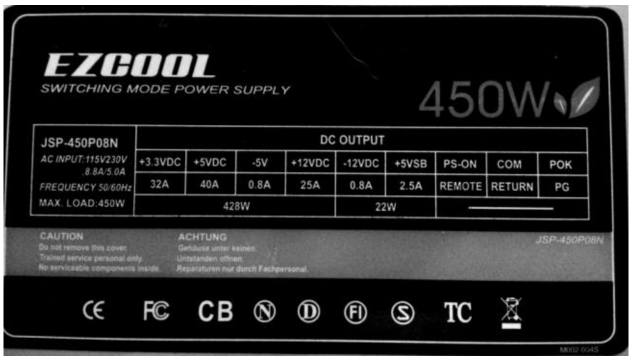
STONTRONICS Switching Adapter
MODEL:DSA-13PFC-05 FCA 051250 RoHS
P/N:T5989DV
INPUT:100-240V ~ 50/60Hz 0.5A I.T.E L.P.S
OUTPUT:+5.1V ⎓ 2.5A 5 1
EFFICIENCY LEVEL:VI PIN1:"+" PIN5:"-"
c UL us LISTED 81GJ E135856
Made In China DVE (G)

Figure 3: Raspberry Pi PSU



TP-LINK TECHNOLOGIES CO.,LTD.
Model: T050060-2C1
Input: 100-240V~ 50/60Hz 0.3A
Output: 5V ⎓ 0.6A
I.T.E. POWER SUPPLY
RoHS
TÜV www.tuv.com
GS geprüfte Sicherheit
MADE IN CHINA 0673

Figure 4: TP-LINK PSU



EZCOOL
SWITCHING MODE POWER SUPPLY
450W

JSP-450P08N	DC OUTPUT								
AC INPUT:115V230V 8.8A/5.0A	+3.3VDC	+5VDC	-5V	+12VDC	-12VDC	+5VSB	PS-ON	COM	POK
FREQUENCY 50/60Hz	32A	40A	0.8A	25A	0.8A	2.5A	REMOTE	RETURN	PG
MAX. LOAD:450W	428W				22W		——		

CAUTION
Do not remove this cover.
Trained service personal only.
No serviceable components inside.

ACHTUNG
Gehäuse unter keinen.
Umständen öffnen.
Reparaturen nur durch Fachpersonal.

JSP-450P08N

CE FC CB N D FI S TC

M002-004S

Figure 5: ATX PSU ratings

4.2 Infrastructure Software

The Raspberry Pi environment is limited in system resources, specifically RAM. It only has 1GB of RAM which is partitioned for use between the system (accessible by the user) and the GPU.

The Raspberry Pi's are configured with the least amount of RAM for the GPU (16MB), leaving the remaining space available for user applications. No windowing system (like the X Window System) will be used, so any larger allocation of RAM for the GPU is unnecessary.

4.2.1 Raspbian Linux

The Raspbian Linux operating system (OS) [5], a derivate of Debian Linux, is installed on the Raspberry Pi's as it has been tailored to best utilise the hardware's resources and features [6]. Further configuration is done on the OS to disable the daemons responsible for controlling the on-board Bluetooth and Wi-Fi modules in order to free up even more available RAM.

4.2.2 Apache Spark

When the project was initiated, Hadoop [7] was suggested as the cluster computing framework to use for the data transformation process. It has a long, successful track record and there are numerous tutorials online on how to install it on Raspberry Pi's. However, Hadoop's original design is based on a distributed file system that is very disk intensive, a feature that does not work well with the Raspberry Pi's due to their slow disk access (see Appendix D). Upon investigation, an alternative framework was considered, namely Apache Spark [8]. It is advertised as being faster than Hadoop by 10 to 100 times, between disk and memory based operations respectively. It also mentions that is can work with a variety of file systems, from local disk to networked and distributed (partitioned) file schemes.

Both Hadoop and Apache Spark were installed on the Raspberry Pi infrastructure to see which solution would suit the needs of the project best.

Linux's Network File System (NFS) service is installed and enabled on the Raspberry Pi master node so the slaves in the Apache Spark configuration can operate on a single source of data, with the least amount of memory overhead.

When using Hadoop's Distributed File System (HDFS) [9], the RAM used by the slave nodes increased from 99MB to 125MB, and from 181MB to 206MB on the master node. In addition to extra RAM use, the time to upload the source data (1GB of plain text files) to HDFS takes around 8 minutes, as the data was partitioned across all the data nodes within the cluster. The same source data takes only 2 minutes to copy to the Apache Spark master node when using Linux's `scp` utility.

Table 1: RAM use per system

System	RAM used (MB)	RAM used (%)
Raspbian OS	35	3.6
Apache Spark (master)	170	17.47
Apache Spark (slave)	115	11.82
Hadoop (master) – Empty HDFS	181	18.6
Hadoop (slave) – Empty HDFS	99	10.17
Hadoop (master) – Populated HDFS	206	21.17
Hadoop (slave) – Populated HDFS	125	12.85

When comparing the amount of remaining RAM available for data transformation operations (Table 2) and the time taken to load data into the network (Table 3), Apache Spark was the primary candidate due to its RAM utilisation and speed of file access. See Appendix B for installation & configuration instructions for Apache Spark in a Raspberry Pi clustered environment.

Table 2: Combined RAM footprints

System with OS	RAM footprint (%)
Apache Spark (master)	21.06
Hadoop (master) – Populated HDFS	24.77
Apache Spark (slave)	15.42
Hadoop (slave) – Populated HDFS	16.44

Table 3: Time to upload data sources

Data destination	Time to upload (MM:ss)
Linux NFS	01:53
Hadoop HDFS	08:02

5. Design Overview of the Data Transformation Applications

The data transformation applications are written in Scala [10], the language in which Apache Spark was created. Apache Spark comes with a rich Scala API, allowing for easy use of the Resilient Distributed Dataset (RDD) [11], the fundamental building block used in parallel programming operations in Apache Spark.

A brief overview of Apache Spark's cluster computing components will be discussed so a better understanding of how parallel programming within the context of the framework is managed and executed.

5.1 Apache Spark Cluster Computing Components

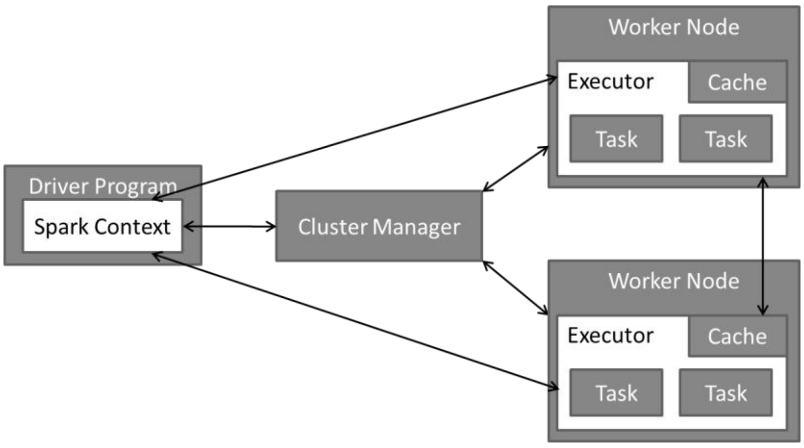


Figure 6: Apache Spark's cluster computing components

The applications created to perform map-reduce operations within Apache Spark need to run using an instance of a *SparkContext*. These are called *Driver* Programs in Apache Spark.

The *SparkContext* connects to a *Cluster Manager* which allocates resources across nodes for the application. In the context of this project, the *Cluster Manager* is Apache Spark's standalone cluster manager.

The *SparkContext* then acquires *Executor*s on *Worker Node*s in the cluster, which will run computations and store data for the applications hosting the *SparkContext*.

The applications are then sent through to the *Executors*, followed by the *Task*s that the *Executor*s need to execute.

*Executor*s can communicate with each other using peer-to-peer networking (like BitTorrent [12]), in order to transmit shared data (broadcast variables).

Once the *Executor*s have completed their *Task*s, they communicate back to the *Driver* Program. [13]

The data transformation operations consist of two main applications: `CategoryCountPerHour` and `CategoryCountPerDay`. The logical progression of data flow and data transformation for each application is explained in detail below.

5.2 Category Count per Hour Application

The data transformation operation steps consist of:

- Loading tweet data from plain text files into `Tweet` objects
- Performing various map, filter and reduce operations on those `Tweet` objects related to the categories and timings mentioned in sections 2 and 3.1
- Outputting a JavaScript Object Notation (JSON) text file for use within the visualisation framework and the `CategoryCountPerDay` application

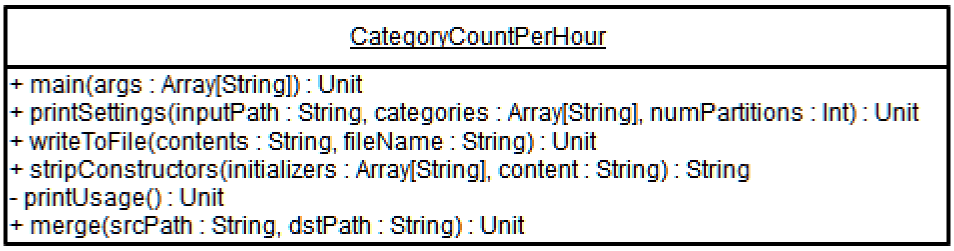

CategoryCountPerHour
+ main(args : Array[String]) : Unit
+ printSettings(inputPath : String, categories : Array[String], numPartitions : Int) : Unit
+ writeToFile(contents : String, fileName : String) : Unit
+ stripConstructors(initializers : Array[String], content : String) : String
- printUsage() : Unit
+ merge(srcPath : String, dstPath : String) : Unit

Figure 7: CategoryCountPerHour class

Apache Spark executes applications by calling the `main` method on a specified class. When submitting an application to Apache Spark, arguments can be supplied to the `main` method as a space separated list of parameters.

To execute the `CategoryCountPerHour` application, the following command needs to be issued:

```
spark-submit --class org.TwitConPro.CategoryCountPerHour TwitConPro-assembly-
1.0.jar /data/20160610 Trump,Clinton 16
```

The command is broken down as follows:

- `spark-submit` is the execution engine, which requires a class with a `main` method to run. In this case, `org.TwitConPro.CategoryCountPerHour` is supplied
- The second parameter, `TwitConPro-assembly-1.0.jar`, is the Scala jar file that contains the `org.TwitConPro.CategoryCountPerHour` class
- The third parameter is the path to the tweet data files to be transformed. This can be a single file or a directory containing multiple files
- The fourth parameter is a comma separated list of keywords to use as categories during the transformation operation
- The fifth parameter is an optional parameter used to instruct Apache Spark on how many cores/partitions to utilise during data processing

A sample of the tweet data is presented in Figure 8:

```
{
 "createdBy": "Particle News",
 "createdAt": ISODate("2016-06-10T03:02:05Z"),
 "coords": ["latitude", 37.3541079, "longitude", 37.3541079],
 "favouriteCount": 0,
 "hashtags": [],
 "twitterID": NumberLong("741103109980618753"),
 "inReplyToName": "",
 "inReplyToStatusID": NumberLong(-1),
 "inReplyToUserID": NumberLong(-1),
 "isRetweet": false,
 "language": "English",
 "place": "",
 "sensitive": false,
 "quotedStatusID": NumberLong(-1),
 "retweeted": false,
 "retweetedCount": 0,
 "tweetText": "A former US ambassador to the Middle East pointed out the inherent flaw in
the Trump... https://t.co/75IUDa9rIn https://t.co/ixChZTEPy8",
 "tweetURL": "https://twitter.com/jess247news/status/741103109980618753"
}
```

Figure 8: Tweet data sample in JSON format

This data is mapped into *Tweet* object instances, as presented in Figure 9:

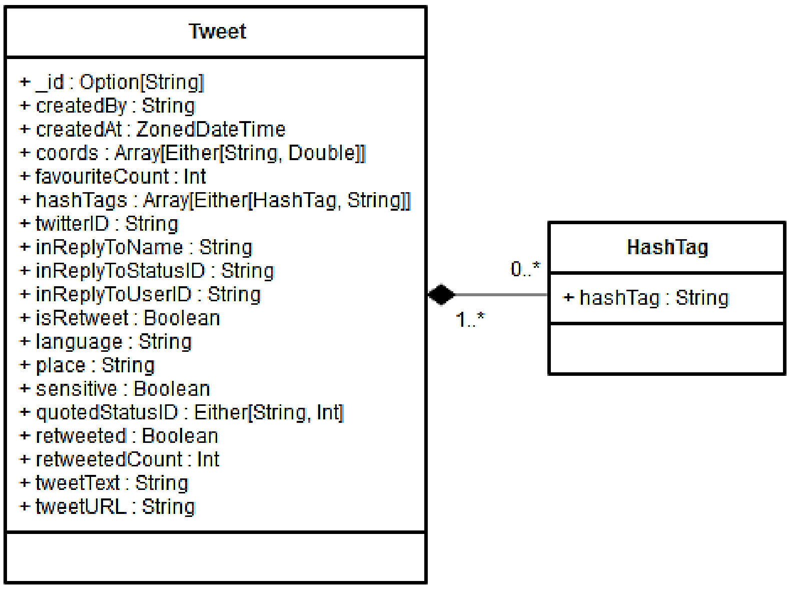


Figure 9: Tweet class in Scala with accompanying HashTag class

The mapped *Tweet* objects are then sorted by date, filtered by hour per day and category, then reduced by category into the sum of unique categories per hour.

The application flow is visually represented via the directed acyclic graph (DAG) in Figure 10:

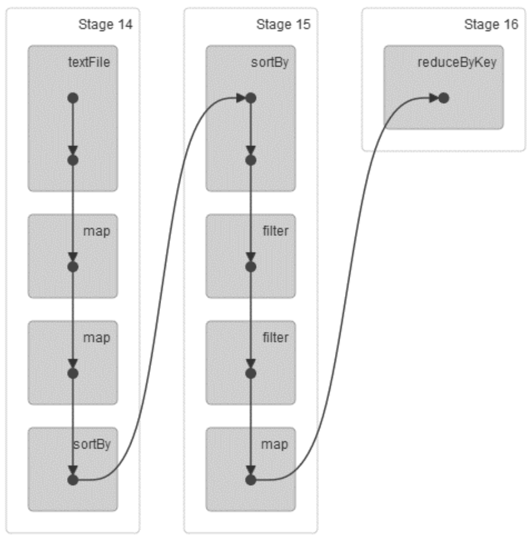


Figure 10: DAG of CategoryCountPerHour program flow

A sample of the output from the `CategoryCountPerHour` application is presented in Figure 11:

```
[{
  "Date": "2016-06-10T03:00:00Z",
  "Data": [
    { "Category": "Trump", "Count": 9947 },
    { "Category": "Clinton", "Count": 6125 }
  ]
}, {
  "Date": "2016-06-10T04:00:00Z",
  "Data": [
    { "Category": "Trump", "Count": 11512 },
    { "Category": "Clinton", "Count": 7130 }
  ]
}]
```

Figure 11: CategoryCountPerHour JSON output sample

Just as the text data of tweets are mapped to a Scala `Tweet` object, the converse also applies when producing the JSON output from Scala objects. The following Scala classes were used to produce the JSON output:

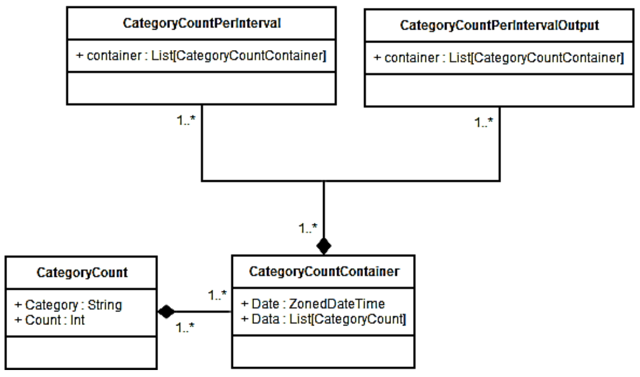

CategoryCountPerInterval
+ container : List[CategoryCountContainer]
CategoryCountPerIntervalOutput
+ container : List[CategoryCountContainer]
1..*
1..*
1..*
CategoryCount
+ Category : String
+ Count : Int
1..*
1..*
CategoryCountContainer
+ Date : ZonedDateTime
+ Data : List[CategoryCount]

Figure 12: Input and output classes for processing CategoryCount JSON data

The top-level JSON array in Figure 11 is represented by the `CategoryCountPerIntervalOutput` class in Figure 12. The JSON objects inside that array are represented by the `List[CategoryCountContainer]` class, which is an array of objects based on the `List[CategoryCount]` class in the `Data` attribute.

5.3 Category Count per Day Application

The data transformation operation steps consist of:

- Loading the JSON output of the `CategoryCounterPerHour` application into a `CategoryCountPerInterval` object (see Figure 12)
- Flattening the hourly based data into a given day by summing up the totals per category mentioned in sections 2 and 3.1
- Outputting a JavaScript Object Notation (JSON) text file for use within the visualisation framework

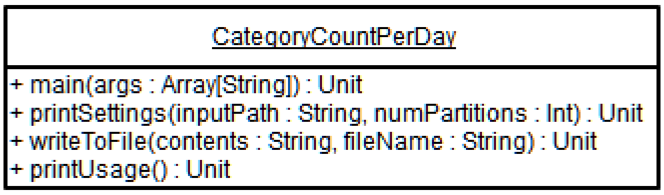

CategoryCountPerDay
+ main(args : Array[String]) : Unit
+ printSettings(inputPath : String, numPartitions : Int) : Unit
+ writeToFile(contents : String, fileName : String) : Unit
+ printUsage() : Unit

Figure 13: CategoryCountPerDay class

The output presented in Figure 11 is used as the input for the `CategoryCountPerDay` application, and is supplied as a parameter to the `main` method.

To execute the `CategoryCountPerDay` application, the following command needs to be issued:

```
spark-submit --class org.TwitConPro.CategoryCountPerDay TwitConPro-assembly-1.0.jar
/data/hourlydata.json 16
```

The command is broken down as follows:

- `spark-submit` is the execution engine, which requires a class to run with a `main` method to run. In this case, `org.TwitConPro.CategoryCountPerDay` is supplied
- The second parameter, `TwitConPro-assembly-1.0.jar`, is the Scala jar file that contains the `org.TwitConPro.CategoryCountPerDay` class
- The third parameter is the path to the output file to be transformed
- The fourth parameter is an optional parameter used to instruct Apache Spark on how many cores/partitions to utilise during data processing

A sample of the output from the `CategoryCountPerDay` application is presented in Figure 14:

```
[{
  "Date": "2016-06-10T00:00:00Z",
  "Data": [
    { "Category": "Trump", "Count": 165018 },
    { "Category": "Clinton", "Count": 108596 }
  ]
}, {
  "Date": "2016-06-11T00:00:00Z",
  "Data": [
    { "Category": "Trump", "Count": 113174 },
    { "Category": "Clinton", "Count": 55724 }
  ]
}]
```

Figure 14: CategoryCountPerDay JSON output sample

Notice that the `Date` attribute shows the date with a zeroed time offset, as it is a representation of an entire day's count of categories.

6. Challenges

The challenges experienced during the project were both technical and nontechnical.

On the technical side, learning how to create the physical cluster computing framework was a new challenge, having had no prior experience in that field. Many mistakes were made with regards to how the environment should be setup and tuned, especially when using hardware like the Raspberry Pi’s, which are resource constrained. Many times the OS on the MicroSDHC cards were corrupted due to the Raspberry Pi’s not dealing well with unexpected power loss.

Learning Scala was a unique challenge, as the programming experience in the group was mostly in Object-Oriented (OO) languages. Scala provides a hybrid OO/functional programming paradigm which took some time to adjust to, particularly around working with immutable state.

On the nontechnical side, the work allocation seemed to be unevenly balanced. The majority of the work was in the hardware setup, cluster computing software configuration and data transformation programming, as well as the data visualisation component setup and programming. This had a major impact on the quantity of work produced for the final product.

Due to time constraints, only two of the five functional requirements listed in section 3.1 were completed by the time this group project was due.

7. Recommendations

A data stream instead of flat JSON files is recommended for a solution such as this. It would fit the intent of the application much better as data could be processed near real-time instead of being batched. Processing historic data in large batches is very expensive to compute, as large datasets require extra management overhead to partition the data into smaller pieces in order to be processed in the resource constrained environment of the Raspberry Pi’s.

8. Conclusion

It is possible to create a cost-effective Big Data processing system using commodity hardware and open source software. The adage “many hands make light work” is certainly true when applied to the world of cluster computing, leveraging the power of parallel computing. The Raspberry Pi cluster computing system presented in this group project is capable of handling the workload issued to it, and would handle various other Big Data processing scenarios quite well.

With future releases of Apache Spark hinting at being 10 times faster than it already is, it might be possible to get even greater performance out of the Raspberry Pi hardware [14].

References

[1] S. Sardana and S. Sardana, “Big Data: It's Not A Buzzword, It's a Movement,” Forbes, 20 November 2013. [Online]. Available: http://www.forbes.com/sites/sanjeevsardana/2013/11/20/bigdata/#3b507d136be7. [Accessed 1 July 2016].

[2] Twitter, Inc., “REST APIs | Twitter Developers,” Twitter, Inc., 2016. [Online]. Available: https://dev.twitter.com/rest/public. [Accessed 3 July 2016].

[3] Raspberry Pi Foundation, “Raspberry Pi 3 Model B,” Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 3 July 2016].

[4] SD Association, “Speed Class,” SD Association, [Online]. Available: https://www.sdcard.org/developers/overview/speed_class/. [Accessed 3 July 2016].

[5] Raspbian, “Welcome to Raspbian,” [Online]. Available: https://www.raspbian.org. [Accessed 3 July 2016].

[6] Raspbian, “Raspbian FAQ,” Raspbian, [Online]. Available: https://www.raspbian.org/RaspbianFAQ#What_is_Raspbian.3F. [Accessed 3 July 2016].

[7] The Apache Software Foundation, “Welcome to Apache Hadoop!,” The Apache Software Foundation, 2014. [Online]. Available: http://hadoop.apache.org. [Accessed 3 July 2016].

[8] The Apache Software Foundation, “Apache Spark Lightning-fast cluster computing,” The Apache Software Foundation, [Online]. Available: http://spark.apache.org. [Accessed 3 July 2016].

[9] The Apache Software Foundation, “HDFS User Guide,” The Apache Software Foundation, 26 January 2016. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Overview. [Accessed 3 July 2016].

[10] École Polytechnique Fédérale de Lausanne (EPFL), “The Scala Programming Language,” École Polytechnique Fédérale de Lausanne (EPFL), [Online]. Available: http://www.scala-lang.org. [Accessed 3 July 2016].

[11] The Apache Software Foundation, “Spark Programming Guide,” The Apache Software Foundation, [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds. [Accessed 3 July 2016].

[12] BitTorrent Inc., “About BitTorrent,” BitTorrent Inc., 2016. [Online]. Available: http://www.bittorrent.com/company/about. [Accessed 3 July 2016].

[13] The Apache Software Foundation, “Cluster Mode Overview,” The Apache Software Foundation, [Online]. Available: http://spark.apache.org/docs/latest/cluster-overview.html. [Accessed 3 July 2016].

[14] S. Agarwal, D. Liu and R. Xin, “Apache Spark as a Compiler: Joining a Billions Rows per Second on a Laptop,” Databricks, 23 May 2016. [Online]. Available: https://databricks.com/blog/2016/05/23/apache-spark-as-a-compiler-joining-a-billion-rows-per-second-on-a-laptop.html. [Accessed 3 July 2016].

Appendix

A Timesheet

ELEN-7046 Group Project – Individual Time sheet - Gareth Stephenson					
Task	**Date**	**Start Time**	**End Time**	**Estimated Hours**	**Actual Hours**
Create PI network with 2 nodes	22 April 2016	18:00:00	22:00:00	4	04:00:00
Modify PC ATX PSU to work with PI's	23 April 2016	12:00:00	18:00:00	6	06:00:00
Project Group Meeting 1	24 April 2016	14:30:00	16:30:00	2	02:00:00
Expand PI network with 2 nodes	25 April 2016	19:00:00	23:00:00	2	04:00:00
Trying to install Hadoop on PI cluster	27 April 2016	09:00:00	17:00:00	4	08:00:00
Trying to install Apache Spark on PI cluster on 4 nodes	28 April 2016	18:00:00	23:00:00	4	05:00:00
Added extra node to the PI cluster, installed Hadoop and Spark on node	29 April 2016	18:00:00	23:00:00	4	05:00:00
Project Group Meeting 2	1 May 2016	14:30:00	16:30:00	2	02:00:00
Learning scala and by working writing word count app, fine tuning Apache Spark	2 May 2016	18:00:00	21:00:00	3	03:00:00
Learning scala and by working writing word count app, fine tuning Apache Spark	3 May 2016	18:00:00	21:00:00	3	03:00:00
Learning scala and by working writing word count app, fine tuning Apache Spark	4 May 2016	21:00:00	22:00:00	2	01:00:00
Learning scala and by working writing word count app, fine tuning Apache Spark	5 May 2016	18:00:00	21:00:00	3	03:00:00
Started CategoryCountPerHour program	6 May 2016	18:00:00	21:00:00	3	03:00:00
Working on CategoryCountPerHour program	7 May 2016	10:00:00	22:00:00	8	12:00:00
Project Group Meeting 3	8 May 2016	14:30:00	16:30:00	2	02:00:00
Working on CategoryCountPerHour program	9 May 2016	18:00:00	22:00:00	4	04:00:00
Running twitter data through the Scala CategoryCountPerHour program	10 May 2016	18:00:00	23:00:00	4	05:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	11 May 2016	21:00:00	23:00:00	2	02:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	12 May 2016	18:00:00	22:00:00	4	04:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	13 May 2016	18:00:00	23:00:00	5	05:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	14 May 2016	10:00:00	22:00:00	8	12:00:00
Project Group Meeting 4	15 May 2016	14:30:00	16:30:00	2	02:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	16 May 2016	18:00:00	23:00:00	3	05:00:00
Calibrating CategoryCountPerHour program based on twitter data using SparkSQL	17 May 2016	18:00:00	23:00:00	3	05:00:00
Master node failure, rebuilding Apache Spark master	21 May 2016	09:00:00	21:00:00	9	12:00:00
Project Group Meeting 5	22 May 2016	14:30:00	16:30:00	2	02:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	23 May 2016	18:00:00	20:00:00	2	02:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	28 May 2016	10:00:00	22:00:00	4	12:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	29 May 2016	12:00:00	20:00:00	4	08:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	30 May 2016	18:00:00	20:00:00	2	02:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	31 May 2016	18:00:00	20:00:00	2	02:00:00
Moving away from Apache SparkSQL, trying to use object based map reduce	2 June 2016	18:00:00	20:00:00	2	02:00:00
Project Group Meeting 6	5 June 2016	12:00:00	18:00:00	2	06:00:00
Got CategoryCountPerHour program to use object based map reduce	7 June 2016	18:00:00	20:00:00	2	02:00:00
Got CategoryCountPerHour program to use object based map reduce	9 June 2016	18:00:00	20:00:00	2	02:00:00
Got CategoryCountPerHour program to use object based map reduce	10 June 2016	18:00:00	20:00:00	2	02:00:00
Project Group Meeting 7	11 June 2016	10:00:00	19:00:00	2	09:00:00
Working on CategoryCountPerDay program	12 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	13 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	14 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	16 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	17 June 2016	18:00:00	20:00:00	2	02:00:00
Project Group Meeting 8	18 June 2016	09:00:00	21:00:00	8	12:00:00
Project Group Meeting 9	19 June 2016	09:00:00	13:00:00	2	04:00:00
Working on CategoryCountPerDay program	20 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	21 June 2016	18:00:00	20:00:00	2	02:00:00
Group Presentation	22 June 2016				00:00:00
Working on CategoryCountPerDay program	23 June 2016	18:00:00	20:00:00	2	02:00:00
Working on CategoryCountPerDay program	24 June 2016	18:00:00	20:00:00	2	02:00:00
Project Group Meeting 10	26 June 2016	14:30:00	17:30:00	2	03:00:00
Working on Group Report	27 June 2016				00:00:00
Working on Group Report	28 June 2016				00:00:00
Working on Group Report	29 June 2016				00:00:00
Group Meeting 11(Virtual)	30 June 2016	18:30:00	20:00:00	1	01:30:00
Project Group Meeting 12	1 July 2016	10:00:00	17:00:00	8	07:00:00
				163	205:30:00

B Apache Spark Installation & Configuration

This is a very high-level guide on how to install and configure Apache Spark 1.6.1 on the Raspberry Pi master and slave nodes.

Assumptions

- Raspbian Linux (kernel version 4.4) is installed on all nodes
- Each of the nodes are configured with unique static IP addresses
- The /etc/hosts file on each node is configured to find the other nodes
- ssh access is enabled and configured on all nodes

Prerequisites for Apache Spark 1.6.1

Java 8 SE SDK (preferably Oracle's distribution) and added to the PATH environment variable, as well as having the JAVA_HOME environment variable set.

Apache Spark Installation

These steps must be performed on all nodes except when specified otherwise.

Create a spark group and user

```
sudo groupadd spark
sudo useradd -g spark spark
```

Download the binary

```
wget http://apache.is.co.za/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz
```

Extract the file to /srv/spark/

```
sudo mkdir -p /srv/spark/
sudo tar -xzf spark-1.6.1-bin-hadoop2.6.tgz -C /srv/spark/
```

Give the spark user ownership of the spark installation

```
sudo chown -R spark:spark /srv/spark/
```

Make a copy of spark-env.sh.template

```
cp /srv/spark/spark-1.6.1-bin-hadoop2.6/conf/spark-env.sh.template /srv/spark/spark-1.6.1-
bin-hadoop2.6/conf/spark-env.sh
```

Modify spark-env.sh with the following settings

```
SPARK_LOCAL_IP=<IP of node>
SPARK_MASTER_IP=<IP of master node>
SPARK_WORKER_CORES=4
SPARK_WORKER_MEMORY=768m
SPARK_EXECUTOR_CORES=4
SPARK_EXECUTOR_MEMORY=768m
```

Make spark-env.sh executable

```
sudo chmod +x /srv/spark/spark-1.6.1-bin-hadoop2.6/conf/spark-env.sh
```

Make a copy of spark-defaults.conf.template

```
cp /srv/spark/spark-1.6.1-bin-hadoop2.6/conf/spark-defaults.conf.template /srv/spark/spark-
1.6.1-bin-hadoop2.6/conf/spark-defaults.conf
```

Modify spark-defaults.conf with the following settings

```
spark.master                      spark://<master node>:7077
spark.eventLog.enabled            true
spark.eventLog.dir                /tmp
spark.driver.memory               256m
```

On the master node, make a copy of slaves.template

```
cp /srv/spark/spark-1.6.1-bin-hadoop2.6/conf/slaves.template cp /srv/spark/spark-1.6.1-bin-hadoop2.6/conf/slaves
```

On the master node, modify slaves with the host names of the slave nodes, separated by a new line

```
<slave node 1 hostname>
<slave node 2 hostname>
<slave node 3 hostname>
<slave node 4 hostname>
```

On the master node, run ssh-keygen while logged in as the spark user

```
ssh-keygen
```

On the master node, run ssh-copy-id so automatic authentication and login from the master node is possible

```
ssh-copy-id <slave node name>
```

Start the Apache Spark master and slave nodes

```
/srv/spark/spark-1.6.1-bin-hadoop2.6/sbin/start-all.sh
```

Execute your Driver Program in the Apache Spark cluster

```
/srv/spark/spark-1.6.1-bin-hadoop2.6/bin/spark-submit --class
<your.apache.spark.Driver.Program> <your-driver-program-assembly.jar> [arguments to your Driver Program]
```

Stop the Apache Spark master and slave nodes

```
/srv/spark/spark-1.6.1-bin-hadoop2.6/sbin/stop-all.sh
```

C Cost of Hardware

The text refers to low-cost, commodity hardware. In Table 4, a pricing breakdown of all the hardware used in the project is presented.

Table 4: Cost of hardware

Description	Quantity	Cost (Rands)
Raspberry Pi 3 Model B	5	3650
Generic 450W PC ATX PSU	1	299
Kingston MicroSDHC, 32GB, UHS-I U3 (SD card)	1	256
ADATA MicroSDHC, 32GB, UHS-I U1 (SD card)	4	500
TP-Link 5 port 10/100 Desktop Switches	2	318
Generic CAT5 UTP LAN cables	5	95
Generic USB A Male to Micro USB B Male cables (power)	5	290

Total cost of hardware: R 5408

(Note: All hardware priced as of 3rd July 2016)

D MicroSDHC Card Benchmarks

Below are the benchmarks of each of the MicroSDHC cards in each of the nodes in the cluster. These metrics were obtained using the Linux *hdparm* program.

```
$ hdparm -Tt /dev/mmcblk0p7

node01:
/dev/mmcblk0p7:
 Timing cached reads:   1320 MB in  2.00 seconds = 659.75 MB/sec
 Timing buffered disk reads:  62 MB in  3.09 seconds =  20.07 MB/sec

node02:
/dev/mmcblk0p7:
 Timing cached reads:   960 MB in  2.00 seconds = 480.34 MB/sec
 Timing buffered disk reads:  42 MB in  3.04 seconds =  13.83 MB/sec

node03:
/dev/mmcblk0p7:
 Timing cached reads:   966 MB in  2.00 seconds = 482.73 MB/sec
 Timing buffered disk reads:  42 MB in  3.04 seconds =  13.83 MB/sec

node04:
/dev/mmcblk0p7:
 Timing cached reads:   960 MB in  2.00 seconds = 480.33 MB/sec
 Timing buffered disk reads:  42 MB in  3.04 seconds =  13.83 MB/sec

node05:
/dev/mmcblk0p7:
 Timing cached reads:   970 MB in  2.00 seconds = 485.09 MB/sec
 Timing buffered disk reads:  42 MB in  3.04 seconds =  13.83 MB/sec
```

Figure 15: MicroSDHC card benchmarks