

**PERFORMANCE EVALUATION OF NoSQL DATABASES
ON STREAMING DATA**

DANI MFUNGO

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF DODOMA

OCTOBER, 2017

**PERFORMANCE EVALUATION OF NoSQL DATABASES
ON STREAMING DATA**

By

Dani Mfungo

A Dissertation submitted in partial fulfillment of the requirements for the
degree of Master of Science in Computer Science of The University of Dodoma.

The University of Dodoma

October, 2017

CERTIFICATION

The undersigned certify that they have read and hereby recommend for the acceptance by The University of Dodoma dissertation entitled, “**Performance evaluation of NoSQL Databases on Streaming Data**” in partial fulfillment of the requirements for award of Master’s degree of Science in Computer Science at The University of Dodoma.

.....

PROF. JUSTINIAN ANATORY

(SUPERVISOR)

.....

Dr. MASOUD MASOUD

(SUPERVISOR)

Date.....

DECLARATION

AND

COPYRIGHT

I, Dani Mfungo, declare that this dissertation is my own original work and that it has not been presented and will not be presented to any University or institution, for a similar or any other degree award.

Signature.....

No part of this thesis may be reproduced, stored in any retrieval system, or transmitted in any form or by any means without prior written permission of the author or The University of Dodoma.

ACKNOWLEDGEMENTS

Many people have contributed to this research in one way or another. Without the support, encouragement and help, that researcher received from various people, it would have not been possible to conduct this research. Researcher would like to take this opportunity to thank those people.

First and foremost, Researcher would like to thank his Supervisors', Prof. Justinian Anatory and Masoud Masoud (PhD) for their support and guidance since the beginning of this thesis to the end. Dr. Masoud, who is very knowledgeable and deep understanding in the field of database, gave me valuable advices in advance thus I am very grateful to have him as my supervisor.

Researcher, do also appreciate prof. Mohammed Guller from University of California for his kindness and time to help whenever possible on Spark Streaming technology.

Also, researcher would like to say thanks to the whole group of Cassandra-HBase-Spark from Gitter for their support and contribution, without them this study could have been tough.

Next but not last, researcher would like to thank his classmate; Mr. Barongo Justus, Mr. Sila Lwendo, Mr. Erick Bunyara, Mr. Sutte Masuha and Mr. Kinto Arusha for their collaboration during the whole period of studies at The University of Dodoma.

Finally, researcher would like to thank all the academic staff from The University of Dodoma for their help since he arrived at the University.

DEDICATION

Special dedication to my family – my parents Mr. Elias Mfungo and Mrs. Zena Athmani for their kindness support, to my sisters, Ms. Zifa Mfungo and Ms. Aneth Mfungo for accompanied me through every effort on fulfillment of this thesis. Finally, this thesis is dedicated to my brothers' Mr. Jesse Mfungo and Mr. Evance Mfungo for their constant encouragement to accomplish the thesis work.

ABSTRACT

The main purpose of this dissertation was to evaluate the performance of Cassandra and HBase NoSQL Databases, that present at Column-oriented category on handling streaming data. The data set used for this evaluation were constructed with the help of the Twitter Streaming API. The environment which used to evaluate the performance of Cassandra and HBase on Streaming Data was Apache Spark with its ability to plot streaming data from source using Spark-R.

Several studies have been considered, and came out with evaluation metrics. Among the metrics found include computation time, memory used, read and write bytes, and CPU usage.

The benchmark performance of the two column family NoSQL Databases (Cassandra and HBase) were completed. The researcher, benchmark 4 different implementations by setting the time interval of 5seconds, 10 seconds, 5 minutes and 10 minutes for 10 iterations with 20 days.

The performance on two NoSQL databases were evaluated in terms of computation time where throughput and latency time were the metrics. Cassandra seem to have the overall good performance in write operation when the streaming workload increase compared to HBase while HBase show the overall low performance in computation for having high average latencies time particularly in writing operation. To have accuracy result, each test results were averaging to came out with average results.

TABLE OF CONTENTS

CERTIFICATION	i
DECLARATION AND COPYRIGHT	ii
ACKNOWLEDGEMENTS	iii
DEDICATION	iv
ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST APPENDICES	xii
LIST OF ABBREVIATION	xiii
CHAPTER ONE: INTRODUCTION	1
1.0 Introduction	1
1.1 Background	1
1.1.1 Relational Databases	1
1.1.2 NoSQL Databases	2
1.1.2.1 Categories of NoSQL Databases.....	2
1.1.2.1.1 Key -value	2
1.1.2.1.2 Document Database	3
1.1.2.1.3 Column Family	3
1.1.2.1.4 Graph Database	4
1.1.2.2 Apache Cassandra	5
1.1.2.3 Apache HBase.....	6
1.1.3 Apache Spark	7
1.1.3.1 Apache Spark Streaming.....	10
1.1.4 Statement of The Problem.....	11
1.1.5 Objective	13
1.1.5.1 Main Objective.....	13
1.1.5.2 Specific Objective	13
1.1.6 Research questions	13
1.1.7 Purpose of the study	13
1.1.8 Significance of The Study.....	14

1.1.9 The structure of the thesis	14
CHAPTER TWO	16
LITERATURE REVIEW.....	16
2.0 Introduction.....	16
2.1 Conceptual definition.....	16
2.2 Related Work	17
2.2.1 Tools for Ingestion	17
2.2.2 Data Analytic Layer.....	18
2.2.3 NoSQL Evaluation.....	19
2.2.4 Cassandra Vs HBase Evaluation.....	19
2.3 Research Gap	22
CHAPTER THREE: METHODOLOGY.....	23
3.0 Introduction.....	23
3.1 Research setting	23
3.2 Research Approach	24
3.3 Research Design.....	25
3.4 Research method and Data Collection	25
3.4.1 Literature review	25
3.4.2 Streaming Simulator.....	26
3.4.3 Experimental setup environment.....	27
3.4.3.1 IntelliJ IDEA setup.....	27
3.4.3.2 Apache Spark Configurations	28
3.4.3.3 Scala Installation	29
3.4.3.4 Twitter Streaming API Setup	30
3.4.3.5 Database Configurations	32
3.4.3.5.1 Cassandra	33
3.4.3.5.2 Apache HBase.....	35
3.4.6.2.1 HBase installation	36
3.4.4 Dataset.....	37
3.4.5 Data Size	37
3.4.6 Data received Rate	38
3.4.7 Experimental Scenario	38
3.4.7.1 How it works.....	40

3.5. Data processing and Analysis	42
3.5.1 Experimental analysis	42
3.5.1.1 Benchmark Results Data	42
3.5.1.1.1 Average benchmark results, triggered after every 5 seconds.....	42
3.5.1.1.2 Average benchmark results, triggered after every 10 seconds.....	43
3.5.1.1.3 Average benchmark results, triggered after every 5 minutes.....	43
3.5.1.1.4 Average benchmark results, triggered after every 10 minutes.....	44
3.5.1.1.5 Read (100%) only workload, triggered after every 5 seconds	44
3.5.1.1.6 Write (100%) only workload, triggered after every 5 seconds	45
3.5.1.1.7 Read (100%) only workload, triggered after every 10 seconds	45
3.5.1.1.8 Write (100%) only workload, triggered after every 10 seconds	46
3.5.2.2.9 Read (100%) only workload, triggered after every 5 minutes	47
3.5.2.2.10 Write (100%) only workload, triggered after every 5minutes	48
3.5.2.2.11 Read (100%) only workload, triggered after every 10 minutes	49
3.6 Ethical Consideration	50
3.7 External Validity	50
3.8 Economic Planning and Costs.....	51
3.8.1 Planning	51
3.8.2 Costs.....	52
3.8.2.1 Development Costs	52
3.8.2.2 Infrastructure Costs	52
CHAPTER FOUR: FINDINGS AND DISCUSSION.....	54
4.0 Introduction.....	54
4.1 Finding and discussion after Documents Analysis	54
4.2 Finding and discussion after Experimental Research study.....	55
4.3 Data received Rate	55
4.4 Benchmark results.....	56
CHAPTER FIVE: SUMMARY, CONCLUSION AND FUTURE WORK.....	64
5.0 Introduction.....	64
5.1 Summary of the research objectives	64
5.2 Conclusions.....	65
5.3 Future Work	67

REFERENCES	68
APPENDICES	74

LIST OF TABLES

Table 1. 1: Column-family database model	4
Table 2. 1: The comparison between Cassandra and HBase	21
Table 3. 1: Objective to appropriate Approaches.....	23
Table 3. 2: Metrics evaluations	26
Table 3. 3: Cassandra Configuration Details	33
Table 3. 4: Load Records-Average evaluation after 5seconds.....	43
Table 3. 5: Load Records-Average evaluation after 10 seconds.....	43
Table 3. 6: Load Records-Average evaluation after 10 minutes.....	44
Table 3. 7: Load Records-Average evaluation after 5 seconds.....	44
Table 3. 8: Load Records-Average evaluation after 5 seconds.....	45
Table 3. 9: Load Records-Average evaluation after 10 seconds.....	46
Table 3. 10: Load Records-Average evaluation after 10 seconds.....	47
Table 3. 11: Load Records-Average evaluation after 5 minutes.....	48
Table 3. 12: Load Records-Average evaluation after 5 minutes.....	49
Table 3. 13: Load Records-Average evaluation after 10 minutes.....	50
Table 3. 14: Development costs	52

LIST OF FIGURES

Figure 1. 1: Graph NoSQL Database	5
Figure 1. 2: Apache Cassandra Architecture.....	6
Figure 1. 3: Apache HBase Architecture	7
Figure 1. 4: Apache Sparks and its plugins.....	8
Figure 1. 5: Internal architecture of Apache Spark	8
Figure 1. 6: Schema of map-reduce	9
Figure 1. 7: Transformation of DStream.....	10
Figure 1. 8: DStream represent multiple RDD.....	11
Figure 3. 1: Strong Experiment Design.....	25
Figure 3. 2: Streaming data from different source to persistence storage.....	27
Figure 3. 3: OpenJDK installation	28
Figure 3. 4: Apache Spark setup	29
Figure 3. 5: Twitter credential information.....	31
Figure 3. 6: Spark Streaming model for processing the Twitter Stream.....	32
Figure 3. 7: Nodetool utility.....	34
Figure 3. 8: Starting HBase via Command tool	36
Figure 3. 9: Data Structure of tweets	37
Figure 3. 10: Simulation scenario for Cassandra	39
Figure 3. 11: Simulation scenario for HBase	39
Figure 4. 1: Evaluation metrics	55
Figure 4. 2: Receiving rate at interval of 5seconds for 11 minutes.....	56
Figure 4. 3: Read (100%) only workload: 5 seconds for 20 days	57
Figure 4. 4: Write (100%) only workload: every 5 seconds for 20 days	58
Figure 4. 5: Read (100%) only workload: every 10 seconds for 20 days	58
Figure 4. 6: Write (100%) only workload: every 10 seconds for 20 days	59
Figure 4. 7: Read (100%) only workload: every 5minutes for 20 days	60
Figure 4. 8: Read (100%) only workload: every 5minutes for 20 days	61
Figure 4. 9: Write (100%) only workload: every 10 minutes for 20 days	62
Figure 4. 10: Average Latency time after every 10 minutes in 10 iterations for 20 days	63
Figure 4. 11: Throughput after every 10 minutes in 10 iterations for 20 days	63

LIST APPENDICES

Appendix I: External Examiner Corrections..... 74

LIST OF ABBREVIATION

ACID	Atomicity, Consistency, Isolation and Durability
API	Application Program Interface
BASE	Basically Available, soft state, Eventual Consistency
CQL	Cassandra Query language
CPU	Central Processing Unit
HDFS	Hadoop Distributed File System
HTML	Hypertext Markup Language
NoSQL	Not only Structured Query Language
RDBMS	Relational Database Management System
SQL	Structured Query Language
UDOM	University of DODOMA
YCSB	Yahoo! Cloud Service Benchmark

CHAPTER ONE

INTRODUCTION

1.0 Introduction

Data is the crucial thing. We cannot avoid speaking of data while we need it in our daily life. Data can be stored in form of clustering or classification, supervised or unsupervised, and when processed, an important decision could be made from it. The relational database has been the vital choice for many professional when it comes to data management because of its ability on maintain ACID properties. As the technologies changes, data management professionals can store structured, semi-structured and unstructured data in a collective ways of data management tools called NoSQL. There are technologies which help to measure the flood velocity of data from different sources, Hadoop and Map-Reduce Technique work best in distributed system where it processes data in terms of batch system. Apache Storm and Apache Spark are new processing data specifically for streaming data.

1.1 Background

1.1.1 Relational Databases

These represent objects in tables which consist of tuples and fields. Relational database management systems are set of programs special for managing data and programs that are used to manipulate data and most of them consist of query language, data dictionary, memory management programs and storage programs(Sullivan, 2015). SQL is query language of all relational databases which perform both defining data structure and manipulation of data operations such as insert, update, delete even reading of data. Relational databases are faced by different challenge such as low volumes of read and write operations, low latency response times and low availability tendency (Sullivan, 2015) which has been solved by the

originated of NoSQL Database (Lakshman, Melkote, Liang, & Mayuram, 2016; Sullivan, 2015).

1.1.2 NoSQL Databases

Are designed to run in multiple server although this is not a necessary requirement (Tolerance, 2009). The benchmark motivations of NoSQL databases including the need for availability, scalability (Scale-Up), schema-less, simplicity and cost control(Sullivan, 2015). There are four categories of NoSQL Databases namely Key-Values, Document, Column-Family and Graph databases(Eric Redmond, 2012; Tolerance, 2009). Relational databases follow ACID properties, NOSQL databases follow the BASE properties (Sadalage & Fowler, 2012). All the NoSQL database must guarantee the CAP theorem where the relationship among availability, Consistency and persistence tolerance are stated (Eric Redmond, 2012; Lynch, 2014), the CAP theorem state that: “any networked shared-data system can have at most two of three desirable properties which are consistency (C), high availability (A) of that data (for updates) and tolerance to network partitions(P)” (Brewer, 2012).

Among major player of NoSQL Database includes, Google BigTable, HBase, Hypertable, Amazon Dynamo, Voldemort, Cassandra, Redis, CouchDB, MongoDB (Gokavarapu, 2010). The basic reasons why I chose this kind of database is because of its properties of schema-less and its use simple mechanism to store data in binary form.

1.1.2.1 Categories of NoSQL Databases

1.1.2.1.1 Key -value

The key value type basically, uses a hash table with a unique key point to a particular item of data (Makris & Tserpes, 2016) . The performance of key value database has

great impact due to its ability of caching mechanism that accompany the mappings. It has been designed for storing, retrieving and managing the data structure and associative arrays. It has schema-less properties and the value of data is opaque. Value in key-value (Sharma & Tim, 2015) databases are accessed by a key and the stored value can be images, binaries, videos, HTML, strings and other formats.

Key-value database has advantage on flexibility of data model, because data store does not enforce any structure on the data. Also, the architecture of the key-value favors high performance than relational databases because there is no need to perform join, union and lock. It does not need to search through every column to search for particular item because of using a key to search the location of the object.

1.1.2.1.2 Document Database

Document NoSQL databases are schema-less and flexible that can allow any type of document to load without understanding the inside structure of the document, thus no need of the prior knowledge of the data structure and the value in it (Sharma & Tim, 2015). The flexibility makes this kind of database mostly used in Agile development process. Some common standard for encoding used by document databases includes JSON, BSON and XML (Eric Redmond, 2012).

1.1.2.1.3 Column Family

Column-oriented NoSQL database (Sadalage & Fowler, 2012), are similar to relational database although are quietly difference. Data are stored in cells grouped in a column rather than as rows of data. Logically its columns are grouped into a family of similar column. Single column families can contain multiple column within. Thus, referencing during read and write always use column and not rows (Datastax, 2015).

Some basic concept to understand about column family are column families, super column and column.

- Column families – show how data stored in disk. Can contain multiple supple columns and column. All data with the same properties in a single column are grouped together in the same file.
- A super column is like a dictionary which contains other columns but no other super columns.
- A column is a column which contains timestamp, name of the value and value itself. The timestamp is what differentiate it with key/value databases.

Table 1. 1: Column-family database model

COLUMN FAMILY		
Column Name 1	Column Name 2	Column Name 2
Value _{1a} : timestamp _{1a}	Value _{2a} : timestamp _{2a}	Value _{3a} : timestamp _{3a}
Value _{2b} : timestamp _{2b}	Value _{2b} : timestamp _{2b}	Value _{3b} : timestamp _{3b}
Value _{1c} : timestamp _{1c}	Value _{2c} : timestamp _{2c}	Value _{3c} : timestamp _{3c}

Source: Sullivan, 2015

1.1.2.1.4 Graph Database

In Graph NoSQL Database, (Fiannaca & Huang, 2015; Sadalage & Fowler, 2012) focus is defining data with relation to other data present on database. Its format is quite different from that of SQL or Column-Family. Its structure use edge, nodes and properties of each. Graph database are good in data mining, monitoring of epidemic disease and finding relationship among objects (Eric Redmond, 2012). We simply say, they use nodes and edges to store and represent data, with the help of edge there is a relationship among nodes with defined properties.

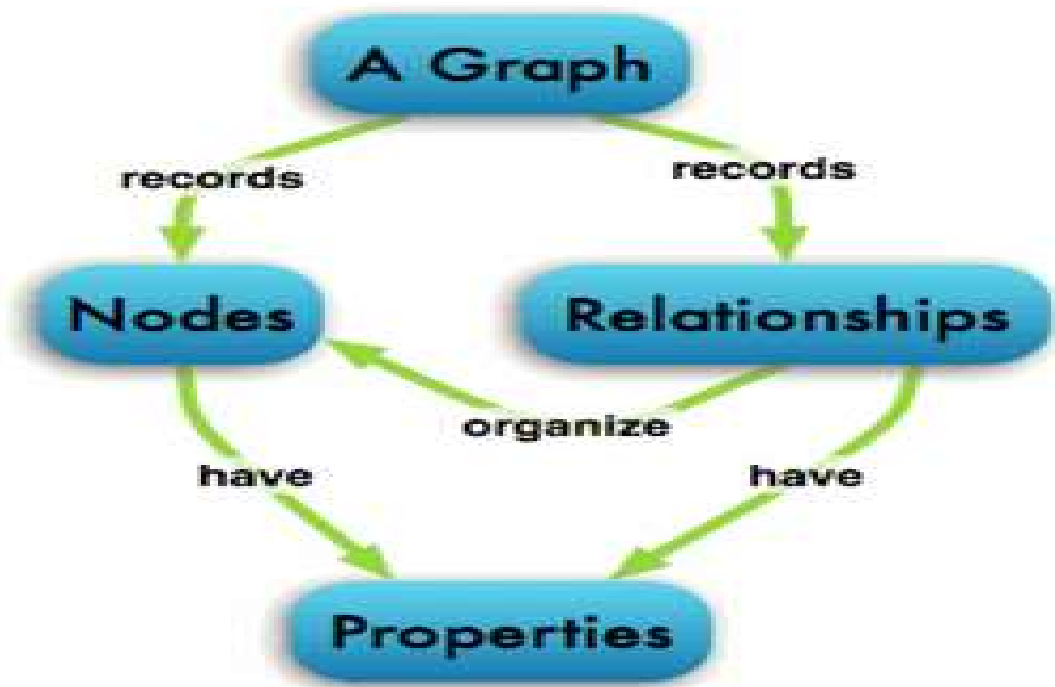


Figure 1. 1: Graph NoSQL Database

Source: Researcher data, 2017

Example of Graph database includes Neo4j, MeshBase, GiraffeDB

1.1.2.2 Apache Cassandra

Cassandra (Barata, 2015) is an open source distributed data store under Apache Software foundation which combine Google’s BigTable data model and Amazon’s Dynamo distributed system technologies.

It was first developed by Facebook in 2008 and in 2009 became one of Apache Incubator project (Barata, 2015). It has decentralized model architecture with no single point of failure. Cassandra support replication which improve the reliability of data, helps the performance by spreading the workload across multiple replicas, the update request is always more challengeable than write request due to updates problems in replicas of data. HBase also has Scalability properties which can enable

additional of new hardware and node without failure. It come with its own language called Cassandra Query language (CQL) and other language drivers for other programming language such as python, java and Scala. Cassandra also support Apache Pig, Apache Storm, Apache Spark and Hadoop Map. One advantage of Cassandra is the support of all fata format from structured, semi-structured and unstructured data.

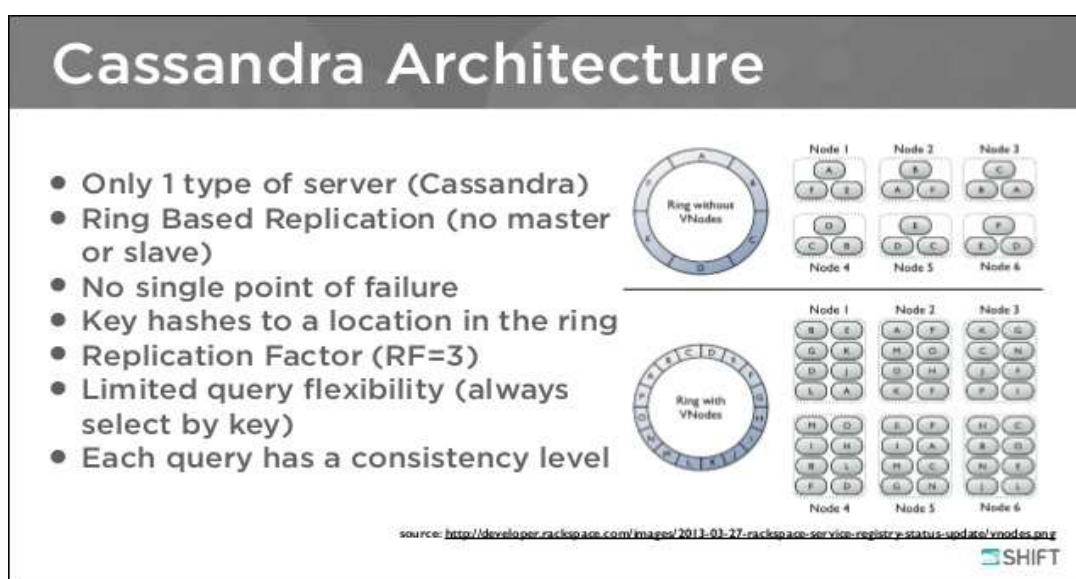


Figure 1. 2: Apache Cassandra Architecture

Source: Eric Redmond, 2012

1.1.2.3 Apache HBase

HBase, (Gokavarapu, 2010) is an open source running on top of Hadoop Distributed file system (HDFS) which become the Apache project in 2010. It is a column-oriented database (Gokavarapu, 2010) which based on Google's BigTable. To access HBase you can use command line or Scala API.

HBase contains different components but the two of the components are much prominent to developers, namely Master server and region Servers. Master Server used to assign region to the region servers while handling of load balance and

maintenance of schema changes and Region Servers are responsible for handling request particularly read and write requests.

Most companies dealing with Big Data problems use HBase. Facebook has been using HBase as a new messaging infrastructure (Eric Redmond, 2012), Stumbleupon use it for real-time data storage and analytics while Twitter use it for storing, monitoring the performance of data and data generation. Other companies using HBase includes Meetup, Ning, Yahoo!, and eBay (Eric Redmond, 2012).

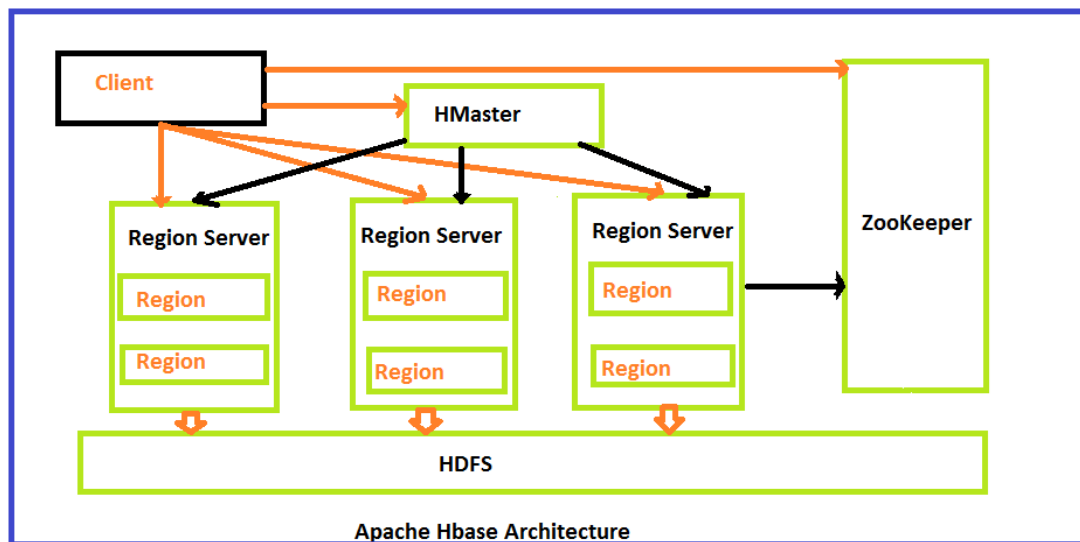


Figure 1. 3: Apache HBase Architecture

Source : “Apache HBase – Apache HBase™ Home,” 2017

1.1.3 Apache Spark

Apache Spark is an open source framework which developed at the university of Berkley and in 2013 entered the apache incubator. It has been designed for resilient distributed dataset (RDD) (Apache Spark, 2015).



Figure 1. 4: Apache Sparks and its plugins

Source : “Databricks - Making Big Data Simple,” 2017

Apache Spark has been built to communicate with languages like java, python, Scala and R. It can communicate with Apache Kafka, Twitter API, parquest, HBase, Cassandra, MongoDB and used for data streams, Machine learning, graphX and SparkSQL (Apache Spark, 2015).

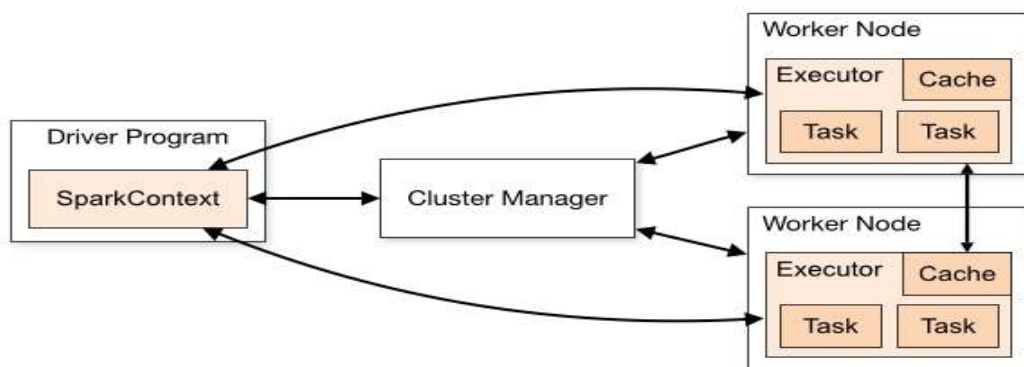


Figure 1. 5: Internal architecture of Apache Spark

Source : Apache spark, 2015

Spark has been developed using Scala Language thus it is compatible with any JVM language. As the compiled code analyzed, the resilient datasets are generated in

some amount. Simply the RDD can be termed as the carrying objects for data and operations.

Spark context allow the data from outside to be able to connect to the cluster manager. Also, Spark context area is responsible for creating instance of data stream and converting size batch data to units.

1. **Map Reduce** is a technique developed by Google for increasing computation process in big data problems (Yang, Dasdan, Hsiao, & Parker, 2007). The concept behind is by distributing the operation by map them over several clusters and reduce the data structure. By doing this makes an expensive operation to be with high parallizable.

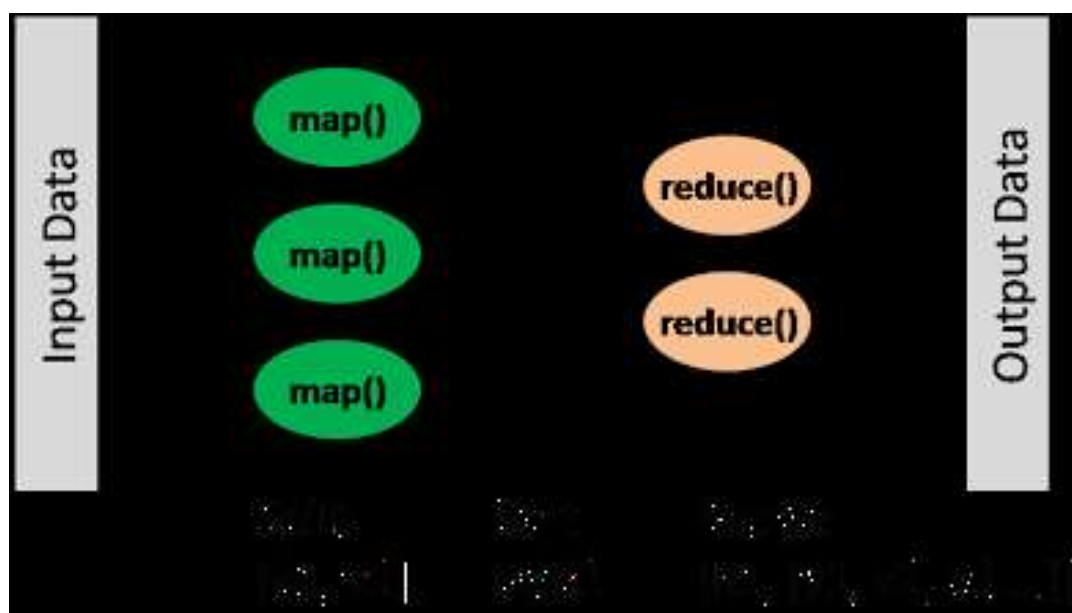


Figure 1. 6: Schema of map-reduce

Source- Researcher data

2. **RDD - Resilient Distributed Data**, Strict structure has been one of the disadvantage of map reduce pattern, due to ability of each dataset to do the same operation. To address this problem, apache Spark solve it by introducing resilient distributed data structure where it encapsulates chunks

of data and transform into a lightweight data structure (Apache Spark, 2015).

The RDD are backbone for Apache Spark since they help the data stream to be discretized by Spark streaming for processed

1.1.3.1 Apache Spark Streaming

Apache Spark was developed for many purpose, specifically for implementation of RDD (Guller, 2015). This framework can be used for streaming data, machine learning and graph processor.

Discretization is a primary abstraction that is provided by Spark streaming when working with data streams. When Apache Spark needs to handle real time data, it needs to split them into batches. Apache Spark implement discretized stream as a sequence of RDD. The interface for processing data stream is called Stream (discrete Stream)

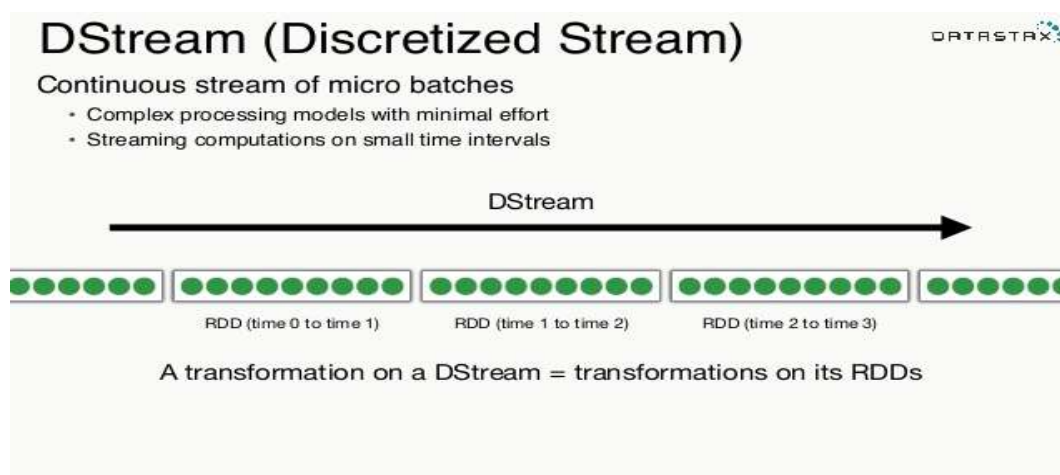


Figure 1. 7: Transformation of DStream

Source: Apache Spark, 2015

Computing; Always the start method begins stream computation. DStream represent all the stream in batches (Apache Spark, 2015) which are handled as RDD in Apache Spark.

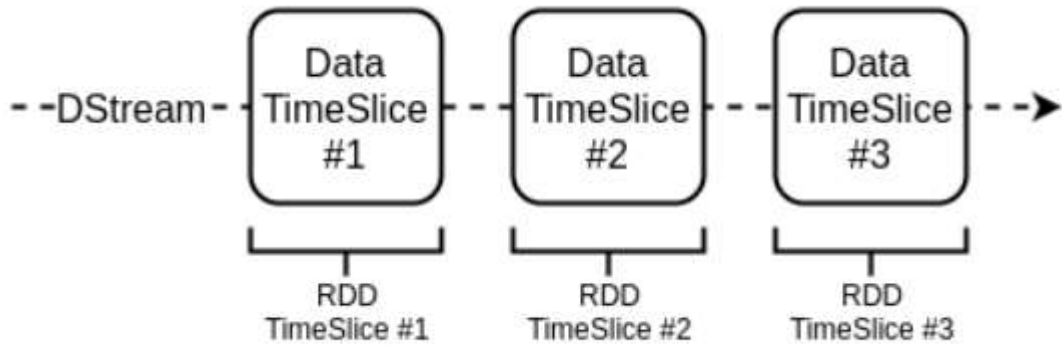


Figure 1. 8: DStream represent multiple RDD

Source: Researcher data

Several application benefit from dealing with streaming data as soon as it arrives (Karau & Konwinski, 2015). For example, the application which track number of coordinates as a mouse as it moving on web page, training machine learning model, tracking of the movement of tyres as the car move. Thus, decision makers in their business have to use Streaming API for analysis.

1.1.4 Statement of The Problem

“The lack of clear NoSQL database evaluation for the case of Data Streaming create a fear, anxiety and mislead in decision making for business success and for software developer during data analytics.”

Continuous flowing of data generated from different sources with different varieties are transferred in a form of the small size of kilobytes, typically these data are generated by web applications, e-commerce purchase, social networks information's, customers from financial trading centers, on line gaming playing. Since these data are generated continuously, they can be processed and used for analytics, thus the analytics information from streaming data can help people who work with data in their everyday and every time work for prediction of customers, sales trends and

weather forecasting, make better decision for their company, hospital, institution and organization to increase performance.

However, the dominant relational databases management systems which usually use SQL language are not designed for querying rapid and continuously flowing data (Li & Manoharan, 2013). These data normally come in different varieties such as structured, semi-structure and unstructured (Babcock & Babu, 2002; Dissertation, 2006; Terry, Goldberg, Nichols, & Oki, 1992). NoSQL are designed to overcome the limitation of RDBMS by allowing data access based on predefined access primitive with its features such as schema-less, scale-up (Li & Manoharan, 2013), to handle both streaming data and real time data (Gupta, Gupta, & Mohania, 2012).

Several studies have been conducted to evaluate the performance of NoSQL database on streaming data by using Hadoop. Example, a study conducted by (Aniceto et al., 2015) indicates that Cassandra has better performance of about 95% compared to other NoSQL Databases. Also, (Datastax, 2015; Gandini & Gribaudo, 2014) use Hadoop technology to compare HBase, MongoDB, Neo4j and Cassandra.

Despite of the benefits observed in apache Spark on streaming data, a study is hardly found that have evaluated the performance of NoSQL databases by integrating it with Spark framework. In response to the matter of streaming data, this study intended to evaluate the performance of NoSQL Database on streaming data by incorporate Cassandra and HBase as NoSQL Databases, with Apache Spark as data link and continuous data from twitter social network.

1.1.5 Objective

1.1.5.1 Main Objective

The main objective of this study is to evaluate the performance of NoSQL databases (Cassandra and HBase) from columnar family against streaming data.

1.1.5.2 Specific Objective

1. To identify performance metrics for evaluating Cassandra and HBase databases on Streaming Data.
2. To assessing the performance of Cassandra and HBase on Streaming data using Apache Spark
3. To propose which of the two NoSQL database is most suitable for streaming Data using Spark.

1.1.6 Research questions

1. What are the performance metrics for evaluating Cassandra and HBase on Streaming data?
2. How does Cassandra and HBase perform on streaming data using Apache Spark?
3. What is the appropriate NoSQL database for streaming data with Apache Spark.

1.1.7 Purpose of the study

The purpose of this experimental study was to report on the performance analysis methodology and results after evaluation by looking the correlation between NoSQL Databases (Cassandra and HBase) with continuously Streaming Data from social network information's generated by "twitter streaming API" with incorporation of Apache Spark. NoSQL Databases (Not only SQL) are defined as the class of

Database Management System (DBMS) that do not follow all the rules of a relational DBMS and cannot use ancient SQL to query data, it is usually used in very larger databases which are explicit at risk of performance downside caused by limitation of RDBMS. Additionally, in this study streaming data was defined as data that generated endlessly by thousands of data sources which usually send in the data records at the same time and in tiny size (order of kilobytes) – information from social networks.

1.1.8 Significance of The Study

The research might give info on the issue of NoSQL Databases technologies notably on the properties, categories, and characteristics. Further, this study would even be a review on the NoSQL information technologies gift. The study would be helpful to the Academician and Students as this study enhance the information on NoSQL information, Apache Spark, and Apache Storm technologies. Furthermore, this study would be beneficial to the system administrators and software developers as the study would provide necessary information regarding the performance on NoSQL databases on Streaming Data, additionally facilitate analyst on choosing the most effective suit technology for fast result on data analytics throughout decision making. Also, this study would offer a baseline info regarding with performance of NoSQL database particularly those from columnar family on streaming data using Apache Spark and Apache Storm technologies.

1.1.9 The structure of the thesis

The rest of the thesis is organized in the following chronological order; chapter two gave enough details on what others have said about NOSQL databases evaluation, Apache Spark, ingestion tools, data analytics layers and Cassandra vs HBase

evaluation. Chapter three covered the methodology part (experimental set-up, implementation, analysis and planning and cost). Chapter four is about findings and conclusion and the last chapter explained about summary, conclusion and future work on the field of NoSQL Database evaluation.

CHAPTER TWO

LITERATURE REVIEW

2.0 Introduction

A literature review discusses revealed info in a very explicit discipline at intervals of a period. It provides a new interpretation of previous material or mix new with previous interpretations. Depending on matters, the literature review could valuate the supply and recommendation of reader on the foremost pertinent or relevant. The focus of a literature review is to summarize and synthesize the arguments and ideas of others while not adding new contributions.

In this chapter, critical review and summary of various concepts related to the streaming data and database as well as various empirical literatures which concerned with evaluating NoSQL Database in terms of throughput, latency and Scalability. The chapter categories sub-themes as ingestion tools for those tools used for data captured, an integrated layer as data link layer, NoSQL database evaluation, the evaluation of HBase and Apache Cassandra as our focus of this study, and lastly is the research gap found after reading several literatures. By review different studies, the objective (1) answered from this chapter.

2.1 Conceptual definition

The following definition used in this study as it written unless defined otherwise by the researcher.

Apache Spark; is an open-source cluster computing framework for big data processing. It has emerged as the next generation big data processing engine, overtaking Hadoop MapReduce computation strategies of big data. (Shanahan & Dai, 2015)

Spark Streaming is an extension of the core Spark API; it allows integration of real-time data from disparate event streams (Ranjan, 2014). But in this study streaming data was defined as data that generated endlessly by thousands of tweets which usually send in the data records via Spark at the same time and in tiny size (order of kilobytes) which are captured after every 5 seconds, 10 seconds, 5 minutes and 10 minutes.

Apache Cassandra is an open source distributed database system that is designed for storing and managing large amounts of data across commodity servers. Can serve both real-time operational data store and streaming data (Apache Software Foundation).

Apache HBase is a column-oriented key/value data store built to run on top of the Hadoop Distributed File System (HDFS) (Apache, 2013).

Hadoop is an open source, Java-based programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment (Apache Software Foundation)

2.2 Related Work

2.2.1 Tools for Ingestion

Gathering of information has been a vital part for any developing and developed business companies, institution and organizations. Different tools have been used to capture a real-time streaming and offline data. Wireshark (“Wireshark · Go Deep.,” 2017) tool has been doing tremendous in capturing data from live network and file on disk. Metasploit (Holik & Horalek, 2015; “Penetration Testing Software | Metasploit,” 2017) used to capture data for security purpose specifically for research

and penetration test of the live or simulation test (Holik & Horalek, 2015; Nicoll, 2011) while snort used for intrusion detection for live network(Roesch, 1999). While most of the evaluation of NoSQL Databases have been done using Yahoo Cloud service Benchmark (YCSB) (Barata, 2014; Cooper, 2010, Gandini & Gribaudo, 2014, 2014; Sergey Bushik, 2012; Veronika Abramova, 2014) as their data ingestion tool based on cloud computing services. In this study, a twitter sentimental analysis mechanism for data ingestion from twitter social network (Agarwal, 2011a, 2011b; Go, Huang, & Bhayani, 2009) was used. Researcher decided to use twitter because it has become the most social network for people to express opinions and views on different sectors which generate the burst amount of volume of data per time. The study conducted by (Adams & McCorkindale, 2013; Wang, Can, Kazemzadeh, Bar, & Narayanan, 2012) use twitter real-time sentimental analysis for U.S presidential election.

2.2.2 Data Analytic Layer

An integrated layer for ingestion streams, storage data, user interface and deep analytics is a crucial layer which use different technologies. IBM Streams (“IBM Streams,” 2016) is a platforms analytics tools used to correlate and analyze information which comes from data streams sources. (Nabi, Wagle, & Bouillet, 2015) conducted an integration of IBM infosphere with YARN on NoSQL Database tried to capture streaming data, (Yulevich, Pyasik, & Gorelik, 2012) use its in detection of motion data. (Biem et al., 2010; Gedik & Andrade, 2012) use infosphere on streaming data. (Taylor, 2012) specify that a data link layer “Oracle Real Time Decision” can improve the quality of organization decision. The most useful Data Analytics layer used for evaluation of NoSQL Database is Hadoop (White, 2012) which provide a Scalable and reliable platform for both storage and analysis. Hadoop

enable several processing model such as YARN (Kumar Vavilapalli et al., 2013) and MapReduce (Eric Redmond, 2012) which access data by batch processing.

Although, Hadoop is most useful for evaluation of streaming data (Gokavarapu, 2010) but it take a lot of time during data process, thus, in this study, the researcher was using an open source platform from Apache foundation which are developed specifically for real time data processing capability, called Apache Spark.

2.2.3 NoSQL Evaluation

A study performed (Datastax, 2015) on NoSQL Databases when evaluation CoachBase, MongoDB, HBase and Cassandra and results showed that Cassandra was better in terms of performance compared to the rest. (Klein, 2015) a case study of NoSQL Database evaluation among Cassandra, MongoDB and Riak resulted on Cassandra to have best throughput performance with highest latency while MongoDB was lower in performance throughput. Other authors who studied about evaluation of NoSQL Database include (Fiannaca & Huang, 2015; Gandini & Gribaudo, 2014; Kalakanti, 2015; Sharma & Tim, 2015; Sukhdev, n.d.; Veronika Abramova, 2014). Although the evaluation of NoSQL database has been conducted with many research or academician, but this study found a performance comparison between Cassandra and HBase on Streaming Data using special distributed integrated data link for streaming data namely Apache Spark.

2.2.4 Cassandra Vs HBase Evaluation

Cassandra was developed by Facebook as an open source distributed database management system in july,2008 (Cassandra, 2016) and later become part of Apache Software foundation in march,2009 (Cassandra, 2016). Apache HBase (“Apache HBase – Apache HBase™ Home,” 2017) is a distributed NoSQL database build on

top of Hadoop. Unlike relational databases, Cassandra store data in structured, semi-structured and unstructured format (Barata, 2015) without affecting the performance of database. Cassandra is Scalable, high available and fast (Barata, 2015) in query execution. Compared to MongoDB and PostgreSQL, Cassandra was indicated to be the best database (Fiannaca & Huang, 2015) where their solutions rely on RAM and CPU cores. The study conducted in 2015 towards the major NoSQL Databases (Cassandra, CouchBase, HBase and MongoDB) (Datastax, 2015), Apache Cassandra was the best in both throughput by workload and Average Latency by workload. (Gokavarapu & Qiu, n.d.) compare HBase and Cassandra in terms of language written, license, protocol, tradeoff and usage, table 1 (Ayush, 2014) indicate some of the comparison between Cassandra and HBase. It was observed that HBase was better on Optimized Batching compared to Cassandra by 8% while Cassandra has higher latency by 12% (Kalakanti, 2015). In this study, researcher compared their throughput performance on handling streaming data.

Table 2. 1: The comparison between Cassandra and HBase

Parameter	HBase	Cassandra
Database type	Column Oriented Data Store	Column oriented Data store
Development language	Java	Java
License	Open Source	Open source
Works on Operating system	Linux, Unix, Windows	BSD, Linux, OS X, Windows
Database schema Used	Schema-less	Schema-Less
Predefined data types	Yes	Yes
Secondary indexes	No	Restricted
Structure Query language	No	No
CAP Theorem	Consistency, Availability	Availability, Partition Tolerance
Application Programming Interface	Java API, REST HTTP API, Thrift	Proprietary protocol
Consistency	Immediate	Immediate and eventual
Mainly used for	Read	Write
Apache Spark	Through use of Zookeeper and Hadoop	Datastax
Single Field Indexes	Yes	Yes
Multi key index	No	No
Partitioning	Dynamic	Sharding
Rebalancing of Nodes in Failure	Automatic	Automatic
Compression of data	Yes	yes
Language used for programming	Java, Python and Scala	Java, Python, and Scala
Triggers	No	No
Foreign keys	No	No
JOIN Concept	No	No
Transaction	No	No

Concurrency	Yes	Yes
Durability	Yes	Yes

Several metrics has been used to evaluate database system includes throughput, latencies and response time (Datastax, 2015) , RAM and CPU usage (Fiannaca & Huang, 2015). This research based on checking the I/O performance of the database where throughput and latency used as testing metrics.

2.3 Research Gap

Nowadays, NoSQL database has shown to have more advantages of use compared to RDBMS (Li & Manoharan, 2013), this indicates that, NoSQL databases should be adapted and used in several sectors. While the continuous flowing of data is essential to be measured and stored in NoSQL databases, several studies reviewed show that the data link technology used are not designed specifically for measuring real-time and continuous flowing of data (Eric Redmond, 2012; Kumar Vavilapalli et al., 2013; White, 2012). Question of whether the evaluation of NoSQL database using Hadoop technology have limited effect on streaming data has been elaborated. In this dissertation, researcher decided to evaluate the performance of NoSQL databases (Cassandra and HBase) on streaming data by means of Apache Spark as data link layer.

CHAPTER THREE

METHODOLOGY

3.0 Introduction

It is important to know the environmental details where the experiment has been conducted to come out with the valid result. The chapter, gives details about Study setting, Design research, Approach used, Planning and Cost of the study, experiment setup, and implementation. This chapter evaluated the metrics chosen in chapter 2 thus objective (1) and (2) considered to be accomplished.

Table 3. 1: Objective to appropriate Approaches

Questions	Objectives	Approach
What are the performance metrics for evaluating Cassandra and HBase on Streaming data?	To identify performance metrics for evaluating Cassandra and HBase databases on Streaming Data.	Literature review (Document analysis)
How does Cassandra and HBase perform on streaming data using Apache Spark?	To assessing the performance of Cassandra and HBase on Streaming data using Apache Spark	experiment
What is the appropriate NoSQL database for streaming data with Apache Spark.	To propose which of the two NoSQL database is most suitable for streaming Data using Spark.	experiment

3.1 Research setting

Study has been carried out at Computer Laboratory found at Ruaha Catholic University. Ruaha Catholic University is a private institution which provide academic service to the public. Provision of academic service and presence of the Computer laboratory was one of the reason why the researcher chosen that

institution. In this study, researcher chose Ubuntu 17.04 as operating system for platform. All the processes have been tested on 64-bit Ubuntu 17.04 OS and may differ on another Linux kernel-based OS. Other tools include; Apache Spark Version 1.6 pre-built for Apache Hadoop 2.6, Scala language version 2.10.5, Apache Cassandra Version 2.0.6, Apache HBase version 1.2.6, Java-8-openjdk.

3.2 Research Approach

This study involved generation and computation of massive volume of data which came from Twitter social network to a standalone platform. To deal with these continuous flow of data, a quantitative research approach as suggested by (Creswell, 2014; L. Christensen, 2015) when deals with numerical and countable data was used. Since Quantitative approach are used to find the relationship between variables and constants, this brings a reasonable means for the same approach to be used in this study when evaluating the performance of two databases.

Since, the study involved the calculation and finding the average of each experiment and control group, the descriptive statistical approach as part of quantitative approach was used. The descriptive statistics was used, and numerical data presented in both numeric and graphs using frequency distribution and scatter graph. Frequency distribution used to show the uniqueness of data rank values and the frequencies of both databases used for evaluations. Also, researcher used scatter graph as evaluator mechanism to bring a clear graphical representation of the relationship between databases on handling streaming data. Only when mentioned a researcher used a line graph to show statistical significant between the databases.

3.3 Research Design

In this study, the strong experimental design has used. The reason behind was due to giving a researcher control over the situations in terms of databases selected, in terms of which gets treatment condition and in terms of the amount of treatment condition that each database received. This can be classified as having controlled experiments in the researcher management by having confidence in the relations observed between the independent (throughput and latency) and dependent variable (iteration time).

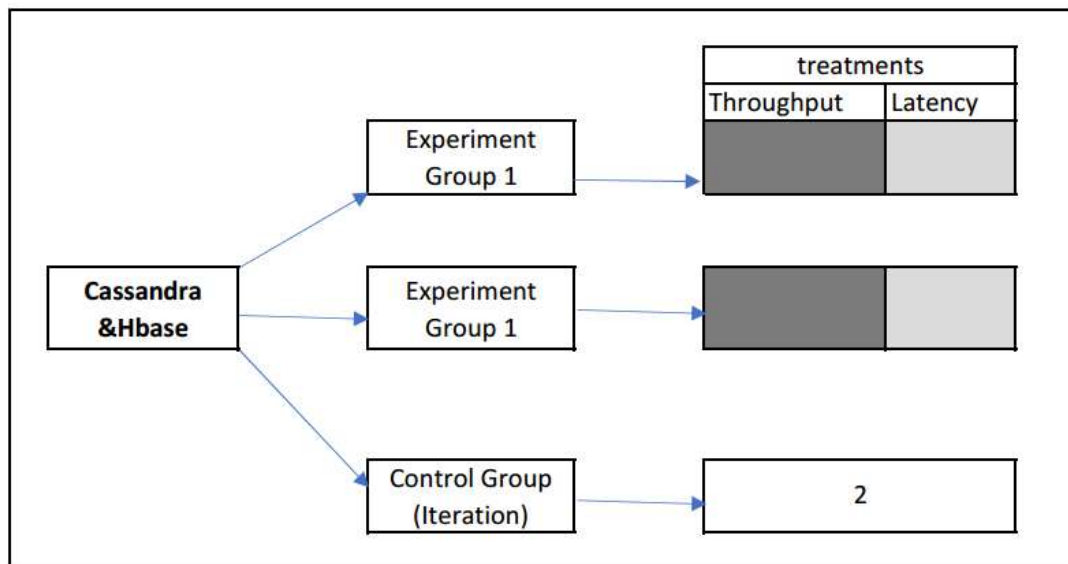


Figure 3. 1: Strong Experiment Design

Source: Researcher data

3.4 Research method and Data Collection

3.4.1 Literature review

To address question (1) several studies were reviewed to determine the performance metrics for HBase and Apache Cassandra databases. Recent and relevant studies from different journals, papers, thesis, articles have been passed through for a purpose of metrics searching. Some of the key terms observed includes SQL,

NoSQL, HBase, Cassandra, performance evaluation, performance comparison and database ranking.

The following table indicated number of performance metrics as used by different researcher to evaluate database

Table 3. 2: Metrics evaluations

METRICS	SQL DATABASES			NoSQL DATABASES		
	MySQL	Postgre SQL	SQL Server	Cassandra	MongoDB	HBase
Throughput	13	12	10	15	17	15
Latency	14	14	17	21	19	23
Response time	9	8	46	12	16	11
CPU usage	15	16	15	5	7	4

Source: Researcher data

3.4.2 Streaming Simulator

To answer Question (2) and (3), data must be collected from streaming simulation. To help simulation of streaming data from Twitter Streaming API, Apache Spark was used, since it has mechanism to deals with streaming data. (Maarala & Rautiainen, 2015). Data source was from Twitter Social Network with Twitter API (Benhardus & Kalita, 2013) due to its ability to generate huge amount of data at a time (Bifet & Frank, 2010). Captured data was stored in either Cassandra or HBase database using Apache Spark data link layer and analyzed.



Figure 3. 2: Streaming data from different source to persistence storage

(Guller, 2015)

Several machine can be used to simulate experiment but for the case of this study, computer has been used to simulator machine (Jacobs, 1993) which can perform and manage scientist experiments.

3.4.3 Experimental setup environment

All testing performed on standalone platform which was reasonable for testing NoSQL databases. To minimize the effect of CPU, Memory usage and I/O variability, researcher performed each iteration on 20 successively days. The tests conducted on a machine with 8 GB RAM, 2 CPU Cores, a single volume of 500 GB of HDD local storage. Environmental tools used for coding was IntelliJ IDEA 3.0 Community edition, Apache Spark 1.6.0 as a data link layer, Scala 2.11.5 as a programming language, twitter Streaming API as congestion layer and Cassandra 3.10, HBase 0.92.0 as database storage.

3.4.3.1 IntelliJ IDEA setup

Intellij IDEA Community was chosen as a platform tools for connecting different packages, software's, gateways, libraries by using Scala language. The minimum

requirements which was considered for installing IntelliJ IDEA was 8GB RAM,500GB hard disk and 1024 * 768 screen resolution.

IntelliJ IDEA Community was downloaded from its website and unpacked to the /opt directory

```
sudo tar xf -*.tar.gz -C /opt/
```

to run IntelliJ Idea for the first time always switch to /bin directory cd opt/-*/bin and run from there by type idea.sh

3.4.3.2 Apache Spark Configurations

Spark processes run under Java Virtual Machine (JVM) thus java is pre-installed before Apache Spark in the machine. The following command are used to install Java in Ubuntu machine

```
$sudo add-apt-repository ppa:webupd8team/java
$sudo apt-get update
```

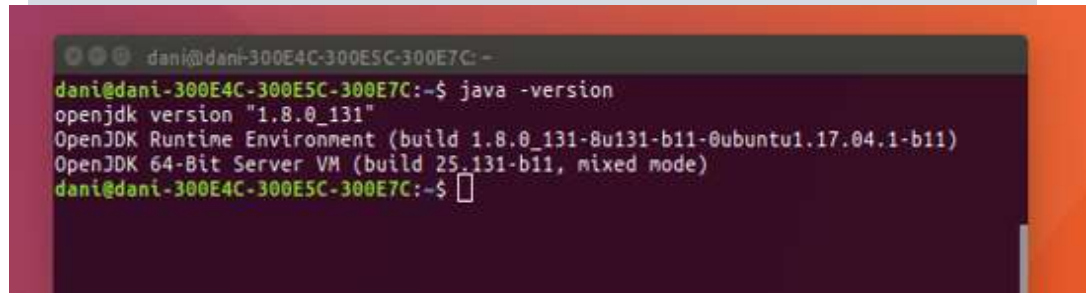
A terminal window screenshot showing the installation of OpenJDK. The prompt is 'dani@dani-300E4C-300E5C-300E7C:~'. The user enters 'java -version' and the output is: 'openjdk version "1.8.0_131"', 'OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-0ubuntu1.17.04.1-b11)', and 'OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)'. The prompt returns to 'dani@dani-300E4C-300E5C-300E7C:~\$'.

Figure 3. 3: OpenJDK installation

Source: Researcher data, 2017

Because Apache Spark is written in Scala language, the Scala should be installed first. Thus, the researcher installed Scala version 2.10.5 and set its folder to the opt directory. The following are commands used to install Scala in ubuntu 16.04.

note: In order to install Spark, we need to make sure, that Java 7+ and Scala 2.10.x is up and running

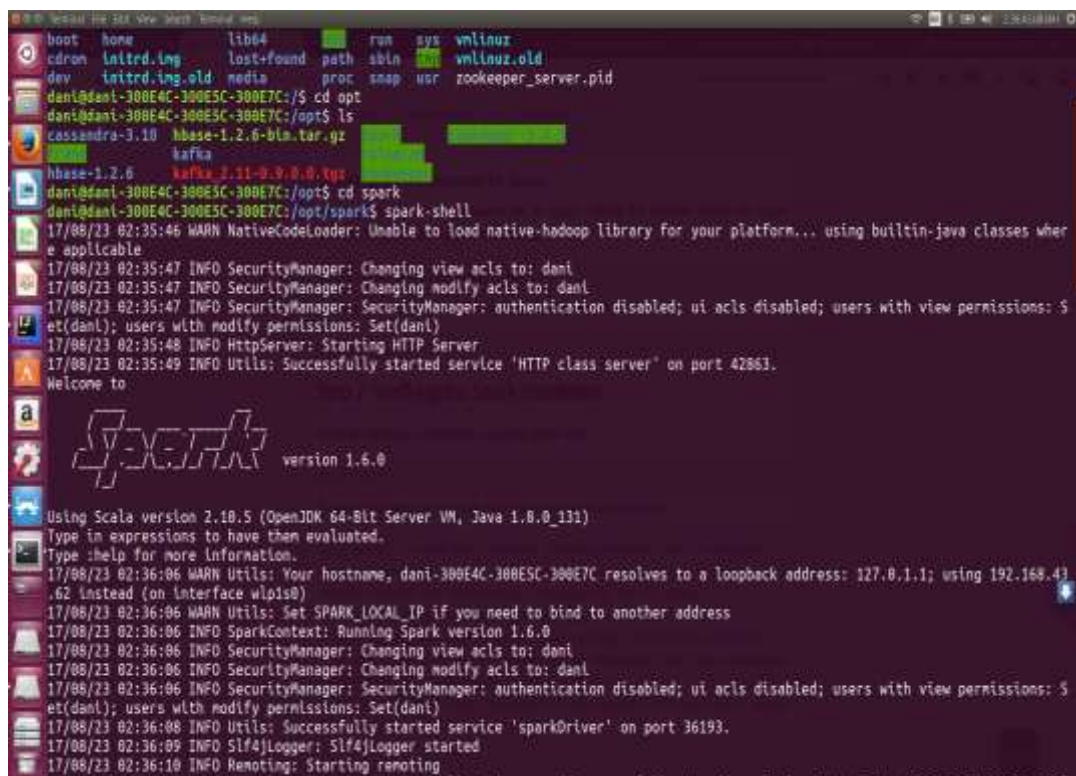
3.4.3.3 Scala Installation

```
sudo apt-get remove Scala-library Scala
wget http://www.scala-lang.org/files/archive/Scala-2.10.5.deb
sudo dpkg -i Scala-2.11.5.deb
sudo apt-get update
sudo apt-get install Scala
```

After install Scala, researcher set up the its path to the /opt directory with the following command

```
export SCALA_HOME=/opt/Scala/Scala-2.10.5
export PATH=$SCALA_HOME/bin:$PATH
```

After download Apache Spark, it must be extracted to the preferred directory, where in this project the /opt directory was chosen throughout. It is important to set the path for Spark directory



```
boot home lib64 run sys vmlinuz
cdrom initrd.img lost-found path sbin vmlinuz.old
dev initrd.img.old media proc snap usr zookeeper_server.pid
dani@dani-308E4C-308E5C-308E7C:/$ cd opt
dani@dani-308E4C-308E5C-308E7C:/opt$ ls
cassandra-3.10 hbase-1.2.6-bin.tar.gz
kafka
hbase-1.2.6 kafka_2.11-0.9.0.0.tgz
dani@dani-308E4C-308E5C-308E7C:/opt$ cd spark
dani@dani-308E4C-308E5C-308E7C:/opt/spark$ spark-shell
17/08/23 02:35:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/08/23 02:35:47 INFO SecurityManager: Changing view acls to: dani
17/08/23 02:35:47 INFO SecurityManager: Changing modify acls to: dani
17/08/23 02:35:47 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(et(dani)); users with modify permissions: Set(dani)
17/08/23 02:35:48 INFO HttpServer: Starting HTTP Server
17/08/23 02:35:49 INFO Utils: Successfully started service 'HTTP class server' on port 42863.
Welcome to

Spark version 1.6.0

Using Scala version 2.10.5 (OpenJDK 64-Bit Server VM, Java 1.8.0_131)
Type in expressions to have them evaluated.
Type :help for more information.
17/08/23 02:36:06 WARN Utils: Your hostname, dani-308E4C-308E5C-308E7C resolves to a loopback address: 127.0.1.1; using 192.168.43.62 instead (on interface wlois0)
17/08/23 02:36:06 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/08/23 02:36:06 INFO SparkContext: Running Spark version 1.6.0
17/08/23 02:36:06 INFO SecurityManager: Changing view acls to: dani
17/08/23 02:36:06 INFO SecurityManager: Changing modify acls to: dani
17/08/23 02:36:06 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(et(dani)); users with modify permissions: Set(dani)
17/08/23 02:36:08 INFO Utils: Successfully started service 'sparkDriver' on port 36193.
17/08/23 02:36:09 INFO Slf4jLogger: Slf4jLogger started
17/08/23 02:36:10 INFO Remoting: Starting remoting
```

Figure 3. 4: Apache Spark setup

Source: Research data

3.4.3.4 Twitter Streaming API Setup

Researcher use Twitter Streaming API since it provides, RESTful and Streaming data API endpoint in order to retrieve published data. REST API used to obtain specific search and posted new tweets while the Twitter Streaming API was used to retrieve every new tweet as soon as it is published. This process has been chosen for security purpose as (Benhardus & Kalita, 2013; Morstatter, Pfeffer, Liu, & Carley, 2013) suggested.

Twitter4j library was used in this experiment to connect with twitter streaming API where streams were accessed via HTTP POST request, by filtering user topic and English language. The Authentication methods (OAuth) must be used to provide authorized access to twitter API. Twitter4j was implemented in Scala by using Maven as shown below.

```
<!-- https://mvnrepository.com/artifact/org.twitter4j/twitter4j-core -->
    <dependency>
      <groupId>org.twitter4j</groupId>
      <artifactId>twitter4j-core</artifactId>
      <version>4.0.6</version>
    </dependency>

<!-- https://mvnrepository.com/artifact/org.twitter4j/twitter4j-stream -->
    <dependency>
      <groupId>org.twitter4j</groupId>
      <artifactId>twitter4j-stream</artifactId>
      <version>4.0.6</version>
    </dependency>
```

To prevent misuse of information from twitter, there are some credential information about the user which are taken and must be included in the application program in order to receiver streaming of data from twitter.

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application

Consumer Key (API Key)	iyNveLi***** ****ETG BGFST
Consumer Secret (API Secret)	upj567Gtdh ***5tD*****FHVIK87Y* ****ETG BGFST
Access Level	Read,write, and direct messages (modify app permissions)
Owner	Dani Mfungo
Owner ID	24*****43

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	24 *****43 T9wQRSTdhftrh*****jfDGT*****Y5
Access Token Secret	OjoDTGFRsgdrF*****JJ5327FCVxmmh****
Access Level	Read, write,and direct messages
Owner	Dani Mfungo
Owner ID	24*****43

Figure 3. 5: Twitter credential information

Source: Researcher data, 2017

The architecture of Spark Streaming Configured for this experiment process is seen in figure 3.6.

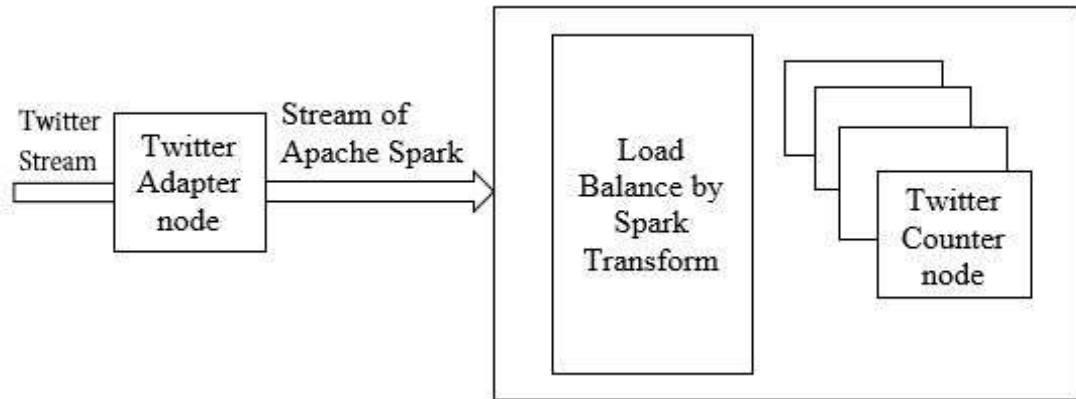


Figure 3. 6: Spark Streaming model for processing the Twitter Stream

Source: Researcher data, 2017

Tweets were received and temporary stored inside the local queue, the writing operation is triggered by considering the setup time interval. Time was set at interval of 5sec, 10sec, 5min and 10 minutes. Once the time end, it automatically writes the tweet from queue to the database. Although the interval may seem as batch process by wrote the tweet at once, but the time is very minimum for batching operation also to observe the writing speed of each database in small amount of time interval and overcome the fast writing as much speed as the tweets received at fast rate.

3.4.3.5 Database Configurations

The new version of both Apache Cassandra and Apache HBase was chosen and selected among NoSQL Databases for benchmarking and well configured as required. Each of the two databases were installed on top of the Ubuntu 17.04 Operating system. While each database was configured to work as a single cluster

where replication was not considered in the process unless said, they both received a Spark streaming of data from Twitter social network.

3.4.3.5.1 Cassandra

A single node Cassandra cluster was setup to receive Spark Streaming with different interval of time for consistent hashing and receiving tweets to a node, Spark Streaming was bundled with Apache Cassandra 2.0.6

Cassandra was installed by using a tarball from the Cassandra website. The murmu3partitioner partitioning strategy was used to improve the performance of the system by implementing a technique known as Consistent hashing.

Only one copy of data of each row on one node was maintained by setting the replication factor to 1. The Status of the node was observed and monitored using nodetool utility. Table 3.3 provide important configuration details of Apache Cassandra.

Table 3. 3: Cassandra Configuration Details

Parameter	Values
Concurrent read	Default (32)
Concurrent write	Default (32)
Initial token	Token generated based on Mumu3Partition hash values
Partition	Default (mumur3partition)
Key_cache_size_in_mb	Default 100MB
Seed provider	Spark streaming data

Cassandra was downloaded from its website cassandra.apache.org/download where the version 3.10 was used for the purpose of this experiment. The /opt directory was

used to extract Cassandra package and use .bashrc command to set the path for Apache Cassandra database.


```
export CASSANDRA_HOME=/opt/cassandra-3.10
export PATH=$CASSANDRA_HOME/bin:$PATH
```

Cassandra must be started from terminal by type

```
cd $CASSANDRA_HOME/bin
```

it's better to confirm if Cassandra service is up and running by checking the status of cluster by using nodetool utility where UN means up and running.

```
$ sudo nodetool status
```



```
CQLSH: command not found
dani@dani-300E4C-300E5C-300E7C:/opt/cassandra-3.10/bin$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> EXIT
dani@dani-300E4C-300E5C-300E7C:/opt/cassandra-3.10/bin$ sudo service cassandra status
[sudo] password for dani:
Unit cassandra.service could not be found.
dani@dani-300E4C-300E5C-300E7C:/opt/cassandra-3.10/bin$ nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
-- State=Normal/Leaving/Joining/Moving
-- Address      Load       Tokens     Owns (effective)  Host ID                               Rack
UN 127.0.0.1    119.15 KiB 256        100.0%            f4ef7283-8bd5-4363-a9cd-d2a0c2cf8222 rack1
dani@dani-300E4C-300E5C-300E7C:/opt/cassandra-3.10/bin$
```

Figure 3. 7: Nodetool utility

Source: Researcher data, 2017

for the case of this research a keyspace called “cycling” was created with the table named “tweets” with name id,tweet” and timein.

```
Create KEYSPACE cycling WITH REPLICATION =  
{'class':'simpleStrategy','replication-factor' 1};  
  
CREATE table tweets(  
  id varchar,  
  tweet varchar  
  timein varachar  
  PRIMARY KEY(id));
```

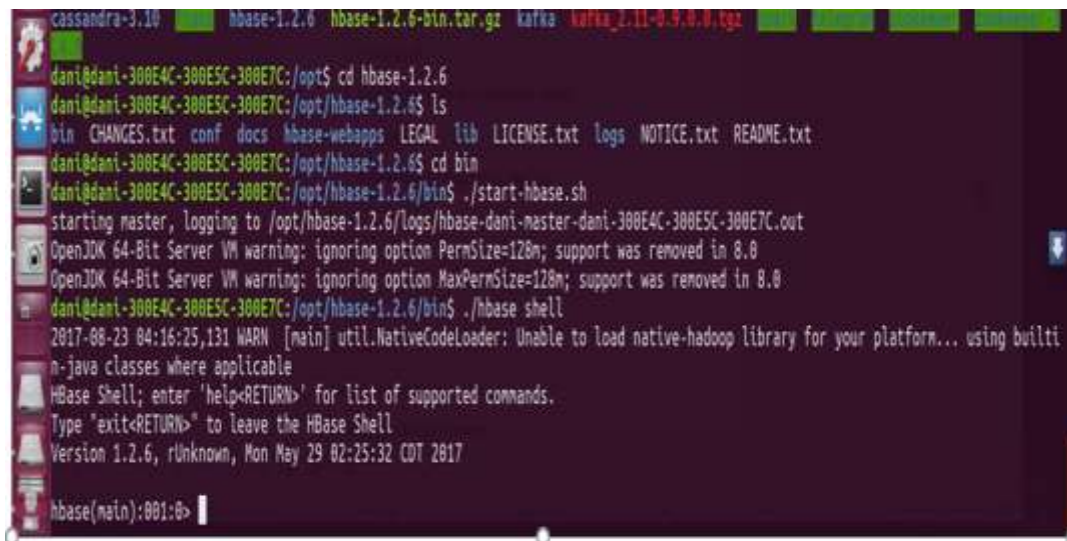
Cassandra connector from Datastax was used to connect Cassandra database and Apache Spark with the help of Spark class known as Streaming Context which allow a Spark streaming from twitter to connect with Spark cluster. This help to get data from and to write to the Cassandra databases.

3.4.3.5.2 Apache HBase

The HBase cluster was setup to receive streaming of Data from Twitter social network using Spark streaming. The whole configuration optimized by Zookeeper and HDFS to run Apache HBase 0.92.0 on ubuntu 17.04 OS. It was split as shown below.

3.4.6.2.1 HBase installation

Download the hbase zip file from apache website
move the downloaded file to the /opt directory and extract them
open the terminal and move to conf folder
cd /opt/Hbase-1.0.0/conf
researcher opened and edit the hbase_env.sh
[dani@dani](#):/opt/hbase-1.0.0/conf \$vi hbase-env.sh
set the hbase path
to start the HBase service we use the following command
.start-hbase.sh
to start the hbase shell using command ./hbase shell



```
cassandra-3.10 hbase-1.2.6 hbase-1.2.6-bin.tar.gz kafka kafka_2.11-0.9.0.0.tgz
dani@dani-300E4C-300ESC-300E7C:/opt$ cd hbase-1.2.6
dani@dani-300E4C-300ESC-300E7C:/opt/hbase-1.2.6$ ls
bin CHANGES.txt conf docs hbase-webapps LEGAL lib LICENSE.txt logs NOTICE.txt README.txt
dani@dani-300E4C-300ESC-300E7C:/opt/hbase-1.2.6$ cd bin
dani@dani-300E4C-300ESC-300E7C:/opt/hbase-1.2.6/bin$ ./start-hbase.sh
starting master, logging to /opt/hbase-1.2.6/logs/hbase-dani-master-dani-300E4C-300ESC-300E7C.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
dani@dani-300E4C-300ESC-300E7C:/opt/hbase-1.2.6/bin$ ./hbase shell
2017-08-23 04:16:25,131 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```

Figure 3. 8: Starting HBase via Command tool

Source: Researcher data, 2017

Note: before starting the HBase, Hadoop service and Zookeeper were started, and a Spark-on-HBase connector was used to connect between Apache Spark and HBase.

3.4.4 Dataset

The researcher, construct his own dataset which fit the need of the research when considering the structure and nature of the storage. Twitter contain a lot of information's, and this has been proved by (Agarwal, 2011a) on his study. Some of the information per single tweet are language, coordinates, like and unlike, number of likes, number of re-tweets, username, user id, description follower and image. Only English text start with (#) was taken because majority of users use it to connect the idea of the same interest.

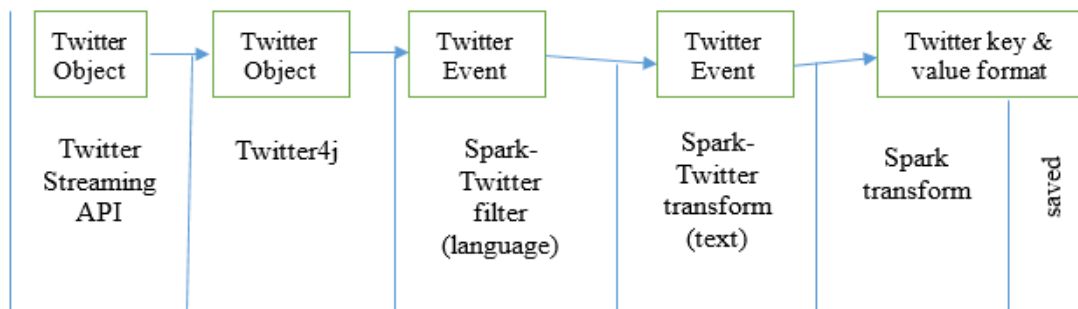


Figure 3. 9: Data Structure of tweets

Source: Researcher data, 2017

3.4.5 Data Size

Due to receiving only English text which starts with hashtags, Researcher limit the content of the tweet. According to UTF-8 characters, a character is represented by 32 bits which is the same as 4 bytes. Thus, if at once, tweet of maximum length of 120 characters received means a size of 800 bytes, so for average of 41 events per each 1ms I expected to have

$$800\text{bytes} * 41$$

$$=32800\text{bytes/events/ms}$$

Thus, to understand the actual data size of the average tweet size

$$\text{Average tweet size} = \frac{\text{Used Database Space}}{\text{Total number of tweets in database}}$$

3.4.6 Data received Rate

The figure 4-2 and 4-3 show the received rate of twitter as it passed through Spark Streaming via twitter Streaming API. The figures indicate events when triggered after 5seconds and the process were run for 8 minutes and 17seconds only with the average of 7.45*events/sec* and 11 minutes with the average receiving rate of 8.24 events/sec.

3.4.7 Experimental Scenario

The metrics performance identified at objective (1) help to go further with this study where the following scenario were used; Data source from twitter was collected, pass through Spark streaming and later stored at Cassandra or HBase. The write and read operations occurred at both databases. The performance of both databases was evaluated by Spark-R as the streaming data continuous.

Apache Spark with Spark-R

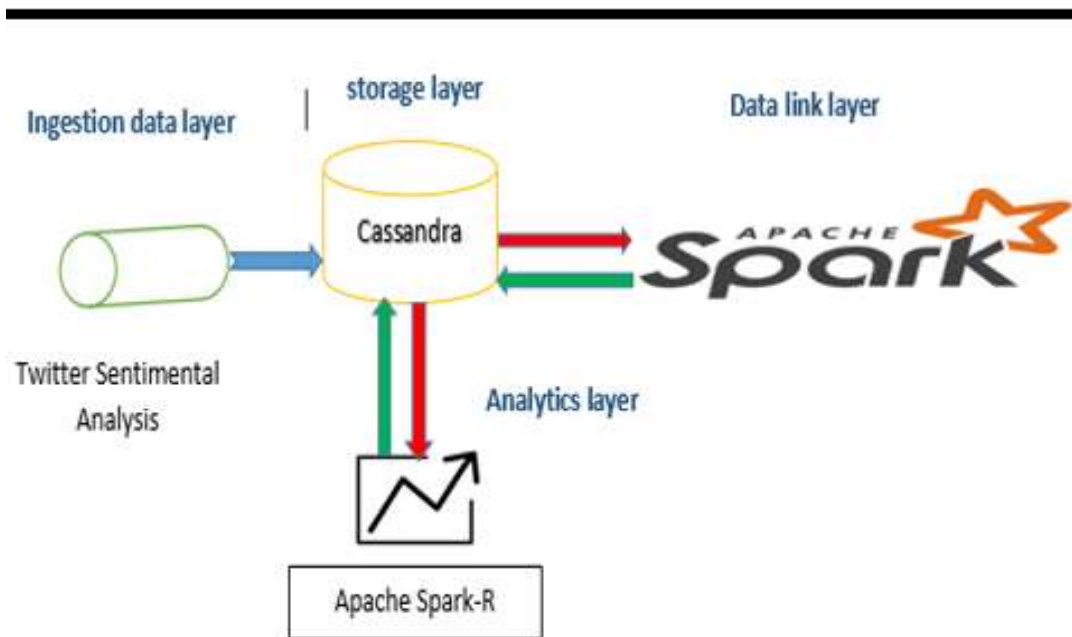


Figure 3. 10: Simulation scenario for Cassandra

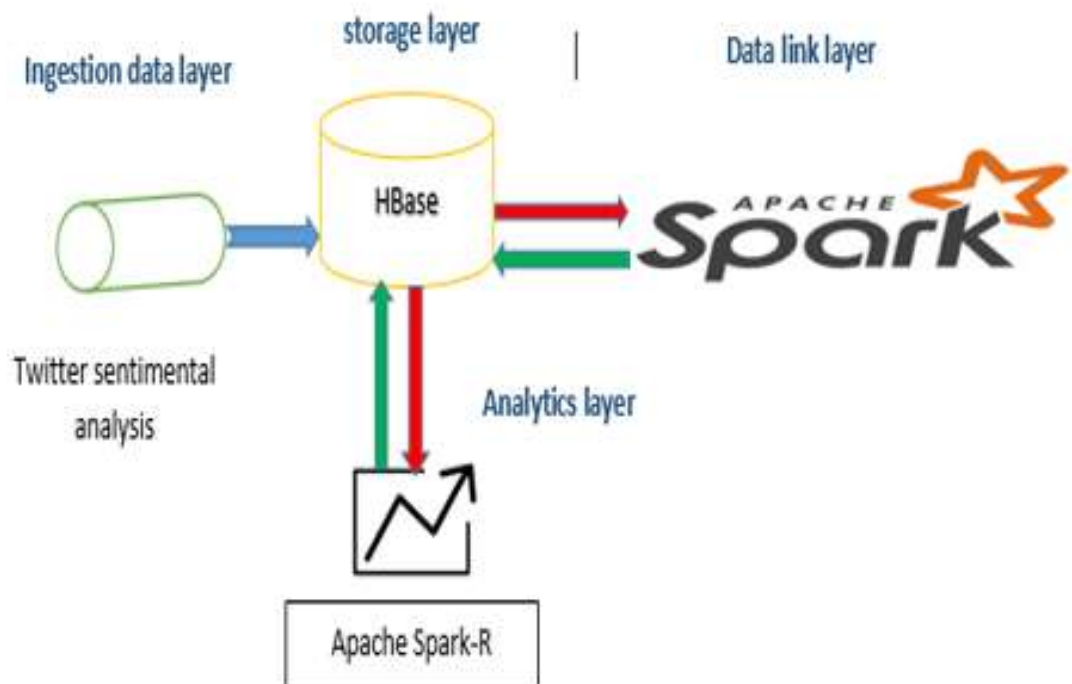


Figure 3. 11: Simulation scenario for HBase

3.4.7.1 How it works

The basic idea for Spark and Cassandra was to deploy both Spark and Cassandra cluster in a single computer simulation which has been done and came out with the results. Cassandra used for purpose of storing streaming data which pass through Spark. Spark Worker nodes mapped to Cassandra and do a data process.

Spark Streaming section are designed to handle Streaming Data from Twitter Streaming API. When the tweets arrive, are served first in spark memory to overcome the issue of data loss which could be caused by network traffic or network fractuation. Spark Worker understand how to read data from shuffle.

A transformation mechanism allows operations of streaming data from shuffle, after that data are served into Cassandra for further analysis.

A very simple schema for Cassandra are created for storing Streaming data.

```
Create KEYSPACE cycling WITH REPLICATION =  
{'class': 'simpleStrategy', 'replication-factor': 1};  
CREATE table tweets (  
  id varchar,  
  tweet varchar  
  timein timestamp  
  PRIMARY KEY(id));
```

The table cycling is quietly simple with three columns, where the tweet column used for tweets as it arrives and controlled with a unique timein identifier which is essential for clustering key as the data stored in Cassandra.

```
val now = new Date ();  
sc.cassandraTable ("cycling", "tweets")  
  .select ("tweet", "id")  
  .where ("timein", now)  
  .filter (includedStatuses contains .getLang ()==lang)
```

```

    . keyBy (row => (row.getString ("tweet")))
    . map { case (key, value) => (key, value.getInt ("id")) }
    . reduceByKey(_ + _)
    . map { case (tweet) => (tweet, now) }
    . saveToCassandra("cycling", "tweet")

```

Scala code above produce reasonably results with integration of Spark and Cassandra.

```

val hbaseConfiguration = (hbaseConfigFileName: String, tweet: String) =>
{
    val hbaseConfiguration = HBaseConfiguration.create()
    hbaseConfiguration.addResource(hbaseConfigFileName)
    hbaseConfiguration.set(TableInputFormat.INPUT_TABLE, tweet)
    hbaseConfiguration
}
val rdd = sparkContext.newAPIHadoopRDD(
    hbaseConfiguration("/opt/hbase/hbase-site.xml", "tweets"),
    classOf[TableInputFormat],
    classOf[ImmutableBytesWritable],
    classOf[Result]
)
import scala.collection.JavaConverters._
rdd
    .map(tuple => tuple._2)
    .map(result => result.getColumn("tweet".getBytes(),
    "columnQualifier".getBytes()))
    .map(keyValues => {
    keyValues.asScala.reduceLeft {
    (a, b) => if (a.getTimestamp > b.getTimestamp) a else b
    }.getValue
    })

```

To enable apache Spark with HBase, the RDD construction in Spark has been configured by setting appropriate configuration measure in hbase-site.xml to read table “tweet”. By having RDD, made easy for HBase operations to handle Streaming from Spark.

The RDD load the data to and from the table “tweet” as “ImmutableBytesWritable” to obtain the exactly number of tweets.

Load an RDD of (ImmutableBytesWritable,tweet)

The getColumnCells return all the collections of tweets with the timestamps when the method getColumn executed.

3.5. Data processing and Analysis

In this study, real time streaming data was processed by using Apache Spark and analyzed using Apache Spark-R and Microsoft Spreadsheet.

3.5.1 Experimental analysis

The experimental method used to find the relationship between Cassandra and HBase by evaluating them on handling Spark streaming data when considered Throughput and Latency time as metrics. The analysis was based on finding the average in both write and read performance when the script set to a particular time. Later, the experimental analysis based on only read or write performance of the databases are taken.

3.5.1.1 Benchmark Results Data

3.5.1.1.1 Average benchmark results, triggered after every 5 seconds

The average results for Cassandra and HBase with a trigger time of every 5 seconds, indicate that Cassandra to have a higher latency time of 6.08 ms and HBase have 2.99 ms.

Table 3. 4: Load Records-Average evaluation after 5seconds

Load Records-Average evaluation after 5 sec		
Database	Throughput	Average latency
Cassandra	17577.0789	6.08871577
HBase	37577.30471	2.99781237

3.5.1.1.2 Average benchmark results, triggered after every 10 seconds

The Average results for 20 days workload which evaluated the throughput and latencies of Cassandra and HBase. The trigger time was 10 seconds after each iteration. As section 3.5.1.1.1 show, Cassandra has highest throughput as compared to HBase.

Table 3. 5: Load Records-Average evaluation after 10 seconds

Load Records-Average evaluation after 10 seconds		
Database	Throughput	Average latency
Cassandra	22255.2778	5.401741321
HBase	75156.60958	3.867572121

3.5.1.1.3 Average benchmark results, triggered after every 5 minutes

When the load increase leads to the increase in latencies to both read and write of Cassandra and HBase. The average Cassandra has 3338921 operations per seconds while HBase has 2 times than of Cassandra.

Table 0-1: Load Records-Average evaluation after 5 minutes

Load Records-Average evaluation after 5 minutes		
Database	Throughput	Average latency
Cassandra	3338291.671	81.26119815
HBase	75156.60958	52.3681736

3.5.1.1.4 Average benchmark results, triggered after every 10 minutes

The average of data collected after every 10 minutes for both 100 read and write in 20 days are indicated at table 3.6.

Table 3. 6: Load Records-Average evaluation after 10 minutes

Load Records-Average evaluation after 10 minutes		
Database	Throughput	Average latency
Cassandra	6475291.78	83.3387891
HBase	22371457.48	72.66789101

3.5.1.1.5 Read (100%) only workload, triggered after every 5 seconds

A total number of 10 iterations was used to collect data in which the trigger time was set after every 5 seconds. This procedure was conducted for almost 20 days successively. Heavy read was performed in both Cassandra and HBase for 100% as data received.

Table 3. 7: Load Records-Average evaluation after 5 seconds

Average throughput and latency after every 5sec for 20 days (read)				
database	Iteration number	Throughput	Average Latency	target throughput
Cassandra	2	1890.529001	2.196017567	2000
	4	3767.94411	2.887767321	4000
	6	5978.80712	4.345678912	6000
	8	7841.99431	6.126189933	8000
	10	9852.15736	6.930014901	10000
HBase	2	1752.00131	1.63291145	2000
	4	3091.22791	1.85971245	4000
	6	5293.11349	1.36371143	6000
	8	5163.18349	2.77391284	8000
	10	5401.17933	2.93374995	10000

3.5.1.1.6 Write (100%) only workload, triggered after every 5 seconds

As in section 3.5.1.1.5, data was written as it came directly from spark streaming. Apache Spark has the capability of storing number of data into memory until the trigger time after every 5 seconds reached. Both throughput and latencies are collected and analyzed.

Table 3. 8: Load Records-Average evaluation after 5 seconds

Average throughput and latency after every 5sec for 20 days (write)				
Database	Iteration number	Throughput	Average Latency	Target throughput
Cassandra	2	1890.529001	1.798087726	2000
	4	3767.94411	2.483267381	4000
	6	5978.80712	3.547878982	6000
	8	7841.99431	6.126789945	8000
	10	9852.15736	6.930014901	10000
HBase	2	1752.00131	10.33891175	2000
	4	3091.22791	10.45979243	4000
	6	5293.11349	10.46379946	6000
	8	5163.18349	10.77393284	8000
	10	5401.17933	10.98374997	10000

3.5.1.1.7 Read (100%) only workload, triggered after every 10 seconds

To insure consistency of data collection, data load was increased to 2 times the collected data as indicated in sections 3.5.1.1.6 and 3.5.1.1.7. Trigger time for read data was set after every 10 seconds. Number of iteration was 10 for both Cassandra and HBase.

Table 3. 9: Load Records-Average evaluation after 10 seconds**Average throughput and latency after every 10sec for 20 days (read)**

Database	Iteration number	Throughput	Average Latency	Target throughput
Cassandra	2	498.529001	7.814	500
	4	997.94411	14.564	1000
	6	1433.80712	13.897	1500
	8	1977.99431	12.987	2000
	10	2496.00131	12.0123	2500
HBase	2	482.9345	26.987	500
	4	911.4378	28.154	1000
	6	1230.437	24.873	1500
	8	1732.9834	28.983	2000
	10	1798.6921	29.342	2500

3.5.1.1.8 Write (100%) only workload, triggered after every 10 seconds

The write operations were performed as soon after the data was arrived in the Streaming platform from Twitter Streaming API. Each time the script trigger, the write operation was performed. Throughput and average latencies was collected for both Cassandra and HBase.

Table 3. 10: Load Records-Average evaluation after 10 seconds

Average throughput and latency after every 10sec for 20days (write)				
Database	Iteration number	Throughput	Average Latency	target throughput
Cassandra	2	498.529001	1.423	500
	4	997.94411	3.335	1000
	6	1433.80712	5.768	1500
	8	1977.99431	4.66	2000
	10	2496.00131	5.01	2500
HBase	2	482.9345	12.994	500
	4	911.4378	14.967	1000
	6	1230.437	11.994	1500
	8	1732.9834	11.989	2000
	10	1798.6921	12.013	2500

3.5.2.2.9 Read (100%) only workload, triggered after every 5 minutes

When the workload increased by 50% and collected after every 5 minutes for 20 days successively, the performance of Cassandra and HBase was affected. Read operations was conducted in each 5 minute obtained average throughput and Latencies for both Cassandra and HBase on Spark Streaming data.

Table 3. 11: Load Records-Average evaluation after 5 minutes

Average throughput and latency after every 5min for 20days (read)				
database	Iteration number	Throughput	Average Latency	target throughput
Cassandra	2	1899.918	2.9287	2000
	4	3973.129	4.43995	4000
	6	5801.872	6.3994	6000
	8	7901.538	8.7892	8000
	10	8678.153	11.6794	10000
HBase	2	1955.99	1.3891	2000
	4	3768.635	1.6756	4000
	6	5601.652	1.8716	6000
	8	7583.678	1.9997	8000
	10	9101.647	1.9998	10000

3.5.2.2.10 Write (100%) only workload, triggered after every 5minutes

Data was collected and analyzed for write operation after every 5 minutes for 20 days successively. Spark control the increase of size in memory before writing them to the database. When the trigger time of 5 minutes reached, the write operation occurs. Collected data in table below indicate the data collected and average throughput and latencies for both Cassandra and HBase.

Table 3. 12: Load Records-Average evaluation after 5 minutes

Average throughput and latency after every 5min for 20days (write)				
Database	Iteration number	Throughput	Average Latency	target throughput
Cassandra	2	1899.918	1.391	2000
	4	3973.129	3.335	4000
	6	5801.872	5.689	6000
	8	7901.538	4.115	8000
	10	8678.153	6.325	10000
HBase	2	1955.99	11.998	2000
	4	3768.635	13.856	4000
	6	5601.652	20.258	6000
	8	7583.678	21.965	8000
	10	9101.647	23.756	10000

3.5.2.2.11 Read (100%) only workload, triggered after every 10 minutes

To maintain validity, workload was increased to 100% with read operations for 100%. Average throughput and average latencies was collected for both Cassandra and HBase. In each iteration, average was taken and recorded.

Table 3. 13: Load Records-Average evaluation after 10 minutes

Average throughput and latency after every 10min for 20days (read)				
Database	Iteration number	Throughput	Average Latency	target throughput
Cassandra	2	1999.216	21.3287	2000
	4	3938.5487	24.2395	4000
	6	5789.0215	25.0213	6000
	8	7899.3284	29.4586	8000
	10	9972.6589	34.8564	10000
HBase	2	1787.2567	57.695	2000
	4	3397.3201	61.562	4000
	6	5200.1275	69.458	6000
	8	7101.3615	77.0985	8000
	10	9081.2548	87.0536	10000

3.6 Ethical Consideration

Gathered data particularly from twitter social network were used for the intention of this study only and not otherwise, and only disclosure data to the public were taken. The researcher also uses his own account authentication to fetch streaming data from Twitter streaming API.

3.7 External Validity

According to (Cozby, 2015) the validity of research dissertation based on how well the instruments set at measuring the variables at a given study. For the purpose of this study, the experimental tools were set in appropriate way not to alter the results. The study was Strong experimental design which based on external validity. Each data was collected as the corresponding time match and averaging due to iteration time. To maintain an external validity, the results were compared often and later an average was be taken for each round.

3.8 Economic Planning and Costs

3.8.1 Planning

In order to achieve all the objectives as mentioned in chapter (1), all the tasks have been mentioned and estimation has been calculated. The Gantt diagram show in detail how the process has been performed since the beginning. The project was started in March 2017.

At the beginning the research started with requirement analysis. The IDEA was to evaluate all NoSQL Databases on streaming data but due to time and cost limitation, only two Database were evaluated. The evaluation process was to run Cassandra and HBase on top of Apache Spark.

The next step was to go through various literature review in order to come out with suitable metrics for evaluation.

Steps after identification of metrics, was to design the architecture of the experiments and setting the main modules, configuring Apache Spark, Cassandra and HBase. Process has gone into three different stages.

- Development of the Ingestion Data link layer (Stream processing application)
- Development of Data Link Layer (Spark application using Scala Language)
- Development of persistence storage focused on Cassandra and HBase.

All those stages involved in depth understanding of technologies and developing the simulating application.

The final step of the research project was to perform the experiments and evaluate Cassandra and HBase in different time interval with different data size due to streaming time while using a single node to each database.

3.8.2 Costs

The cost is divided into two parts, which are development costs and infrastructure costs which include cost for both hardware and software.

3.8.2.1 Development Costs

Programming task is the job which can be done by a Software engineer. To write Scala code, Spark Code require in depth understanding and knowledge in each language.

Thus, the average Cost for this thesis was approximately \$10000 per month, for 12 hours/day programme.

Table 3. 14: Development costs

Cost per month	Number of Months	Total
\$10000	4	\$40000

3.8.2.2 Infrastructure Costs

This can be divided in both software and hardware costs.

For this research, all software costs are free of charge since I used open source software's. The list below indicates software.

- Apache Spark
- Apache Cassandra
- Apache HBase

- IntelliJ IDE community Edition – Scala IDE

The hardware Cost was also zero since I used Computer present at Ruaha Catholic University Computer Laboratory.

CHAPTER FOUR

FINDINGS AND DISCUSSION

4.0 Introduction

The chapter presents the findings and discussion for each question and objectives.

4.1 Finding and discussion after Documents Analysis

The following question were used to came out with evaluation metrics

What are the performance metrics for evaluating Cassandra and HBase on Streaming data?

In order to identify performance metrics used to evaluate Cassandra and HBase Databases, a deep learning of papers, thesis, journal was used. Researcher categories the evaluation in two groups: Relational and non-Relational databases. The metrics used are throughput and latency time and used for these results. The frequency distribution in figure 4-1 indicates and rank evaluated metrics. Because throughput and latency has been used frequently and for a larger number during database evaluation, this was the reason researcher use these two as evaluation metrics.

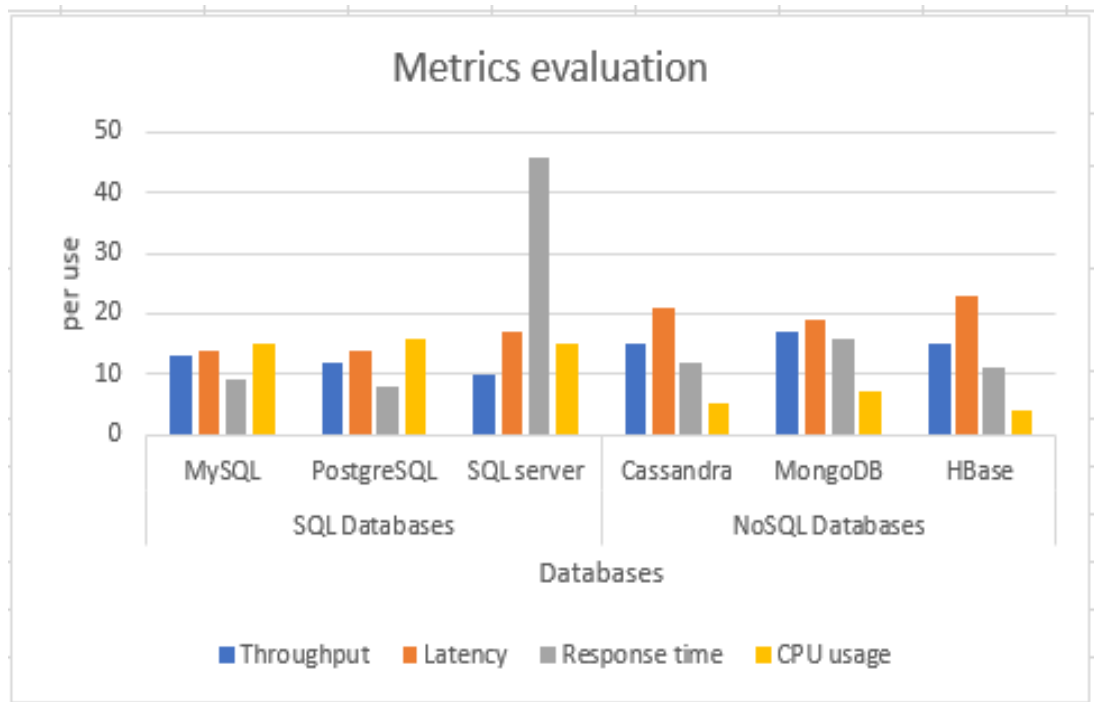


Figure 4. 1: Evaluation metrics

Source: Researcher data, 2017

4.2 Finding and discussion after Experimental Research study

By considering the nature of the research, only two core parameters were considered, namely throughput and latency time for both read and write operations. In case of in-memory usage, input and output operations, the first approach was to consider the streaming of Tweeter data after every 5 sec, 10 sec, 5 min and 10 minutes as indicated from figure 4-4 to figure 4-10.

4.3 Data received Rate

The figure 4-2 and 4-3 show the received rate of twitter as it passed through Spark Streaming via twitter Streaming API. The figures show the events when triggered after 5seconds and the process were run for 8minutes and 17 seconds only with the average of 7.45events/sec and 11 minutes with the average receiving rate of 8.24events/sec

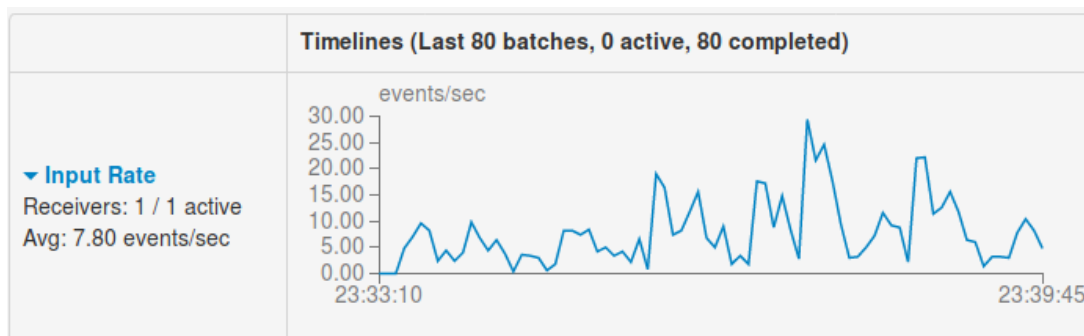


Figure 0-1: Receiving rate at interval of 5seconds for 8 minutes

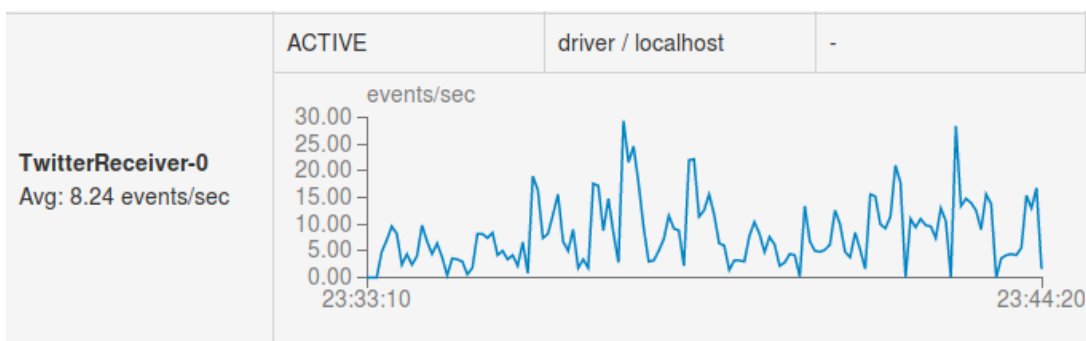


Figure 4. 2: Receiving rate at interval of 5seconds for 11 minutes

4.4 Benchmark results

Experiments were run on both Apache Cassandra and Apache HBase using Spark-R and with Scala script which helped to automate the process. Four different time intervals were used to run the workload into five different iteration. Only the workloads with 10 seconds interval time were set with target throughput of 500,1000, 1500,2000 and 2500 ops/sec for both read and write operations while others, 5sec,5min and 10min were set with target throughput of 2000,4000,6000,8000 and 10000. For each time interval, there was 5 iteration which was set as 2 iterations,4,6,8 and 10 iterations runs for 20 days which act as a trigger to allow a better read and write performance.

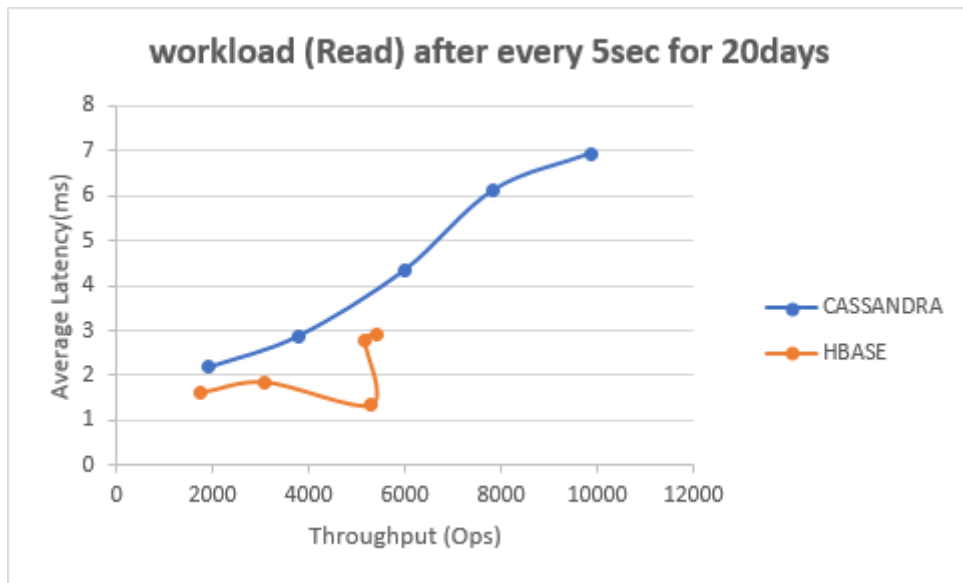


Figure 4. 3: Read (100%) only workload: 5 seconds for 20 days

Cassandra's average Latency increase relatively exponential with increase in throughput. I can conclude that when Cassandra achieve overall throughput it also delivers highest average latencies which is 50% than that of HBase. The read performance of the HBase has improved with increase in throughput and this could be due to its to commit in memory. Thus, as the number of iteration increase the throughput stick to 30% while having low latency time.



Figure 4. 4: Write (100%) only workload: every 5 seconds for 20 days

Cassandra lives up to its name and ready to perform write with considerably less latency than its read. Its performance increase as the workload increase although there a slightly increase in latency. HBase on the other hand, show a linear increase in throughput until when it reaches to 40% of the workload where it stacks on that mark as the workload increase. Although, HBase show better consistency on the throughput at the begging of the process but it has been starting with higher average latencies and this could be because of flush commit process which force the synchronous flush of the write buffer.



Figure 4. 5: Read (100%) only workload: every 10 seconds for 20 days

With 20% operations added as workload where by the trigger mechanism for read happened after every 10 seconds. Cassandra (2458 ops/sec) still has highest throughput while HBase shows the least throughput which is almost similar to the

experiment conducted after every 5 seconds. When reading streaming data, Cassandra seems to have highest latency (14ms) before start to decrease as the workload increase. HBase continue to face the same problem of highest latency as write-only workload in 5seconds-write although it shows good sign as the latency stop increasing at (24ms) as the workload increase.

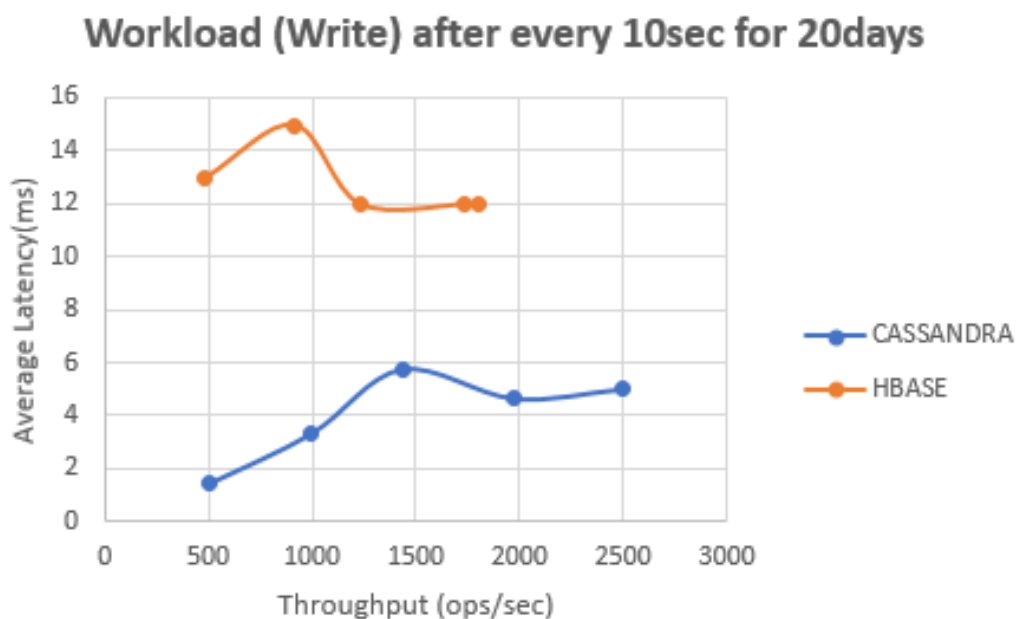


Figure 4. 6: Write (100%) only workload: every 10 seconds for 20 days

In this experiment, the overall performance of Apache Cassandra continues to show good performance as the workload increase in write (100%) operation. Cassandra suffer from increase in average latencies time as the workload increase. Workload performance on HBase is slightly improving for small set of data and stop at (1879ops/sec) while its average latencies time decrease for small amount from 15ms to 12.9 ms which maintained that level as the workload increase.

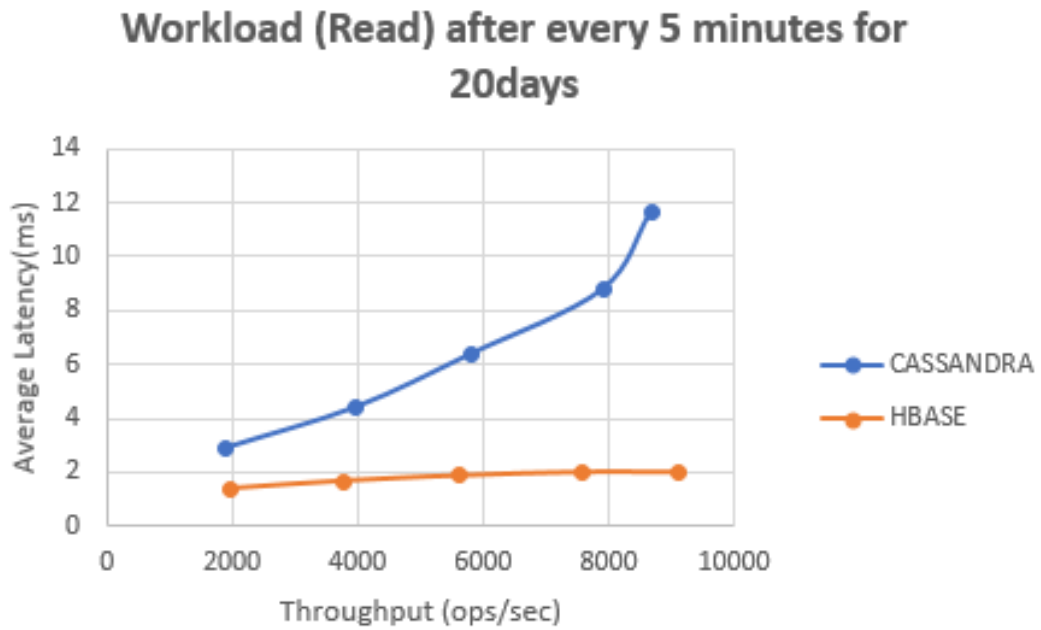


Figure 4. 7: Read (100%) only workload: every 5minutes for 20 days

Heavy read on a very larger data set, HBase show a best performance on both throughput and Latency time. Only the average latencies of 2ms maintained (increase linearly) for the whole workload from day one to the last day, this is one of its advantages in read operation (Jiang, 2012). The performance of Cassandra was affected by the exponential increase in latencies time. By 98% the performance of HBase was better compare to that of Cassandra on handling streaming data with the read of data after every 5 seconds in 20 consecutive days.



Figure 4. 8: Read (100%) only workload: every 5minutes for 20 days

In this experiment, Cassandra (9200ops/sec) still highest throughput with low latency time compared to HBase which has high throughput with highest latency time. Cassandra (5msec) has smallest latency compared to HBase (24ms). Cassandra has slightly increase in average latencies time when doing write operation compare to how HBase increase.

Workload (Write) after every 10 minutes for 20days

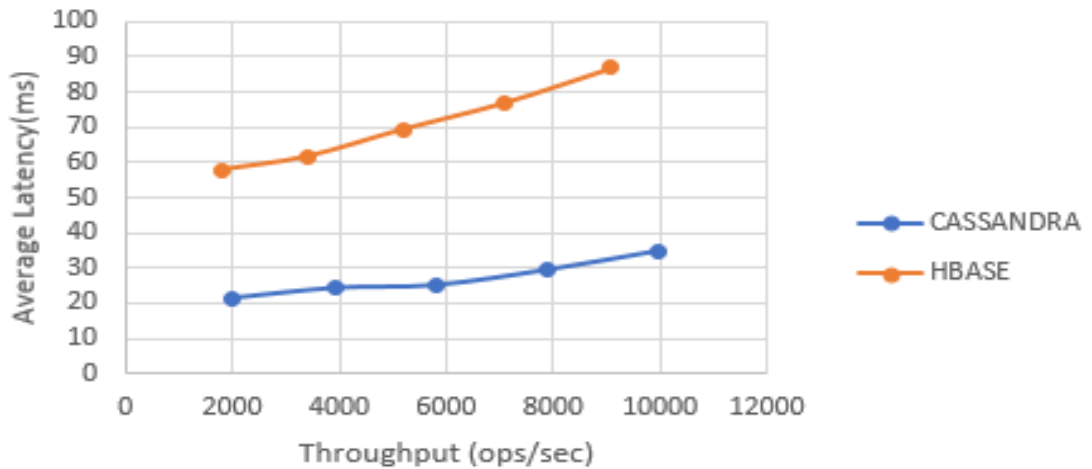


Figure 4. 9: Write (100%) only workload: every 10 minutes for 20 days

The last experiment was the read and write (100%) operations whereby it was desired to prove the performance of HBase over that of Cassandra. HBase show poor performance compared to Cassandra in terms of the response time since it delays are very high (over 50%) to that of Cassandra. Cassandra had the best throughput performance for large data sets, approximately twice the performance of that HBase had. Figure 4.10 and 4.11 show the individual graph for response time and throughput for every 10 minutes in 10 iteration nodes for 20 days on handling streaming data.

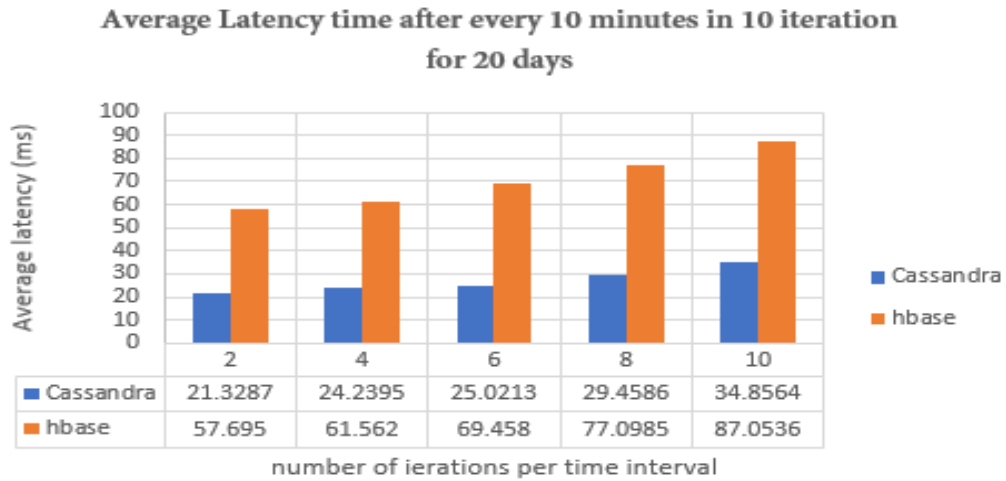


Figure 4. 10: Average Latency time after every 10 minutes in 10 iterations for 20 days

The average latencies time between Cassandra and HBase in general show that, Cassandra has lowest average latencies time compared to HBase as the workload increases. While the increase of latencies time in Cassandra was almost linear, in HBase was exponential increment of average latencies time.

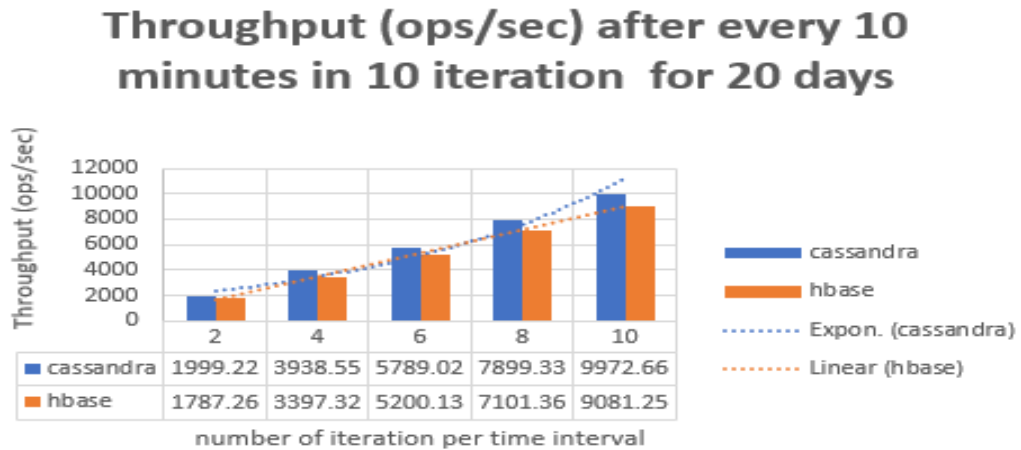


Figure 4. 11: Throughput after every 10 minutes in 10 iterations for 20 days

As the workload increase by 100% of streaming data flowing from Twitter Streaming API, there were high increase of throughput in Cassandra compared to that of HBase database.

CHAPTER FIVE

SUMMARY, CONCLUSION AND FUTURE WORK

5.0 Introduction

This chapter looks into the dissertation objectives and establishes to what extent the study has achieved.

5.1 Summary of the research objectives

1. To identify performance metrics for evaluating Cassandra and HBase databases on Streaming Data.
2. To assessing the performance of Cassandra and HBase on Streaming data using apache Spark.
3. To propose which of the two NoSQL database is most suitable for streaming data using Spark.

The summary of objectives are summaries below

- 1. To identify performance metrics for evaluating Cassandra and HBase databases on Streaming Data.**

Documents analysis were used to come out with evaluated metrics. Both NoSQL and SQL databases were evaluated. NoSQL databases includes Cassandra, HBase and MongoDB where by SQL databases includes SQLServer, MySQL and PostgreSQL. Selected metrics for evaluation was throughput and average latencies time.

- 2. To assessing the performance of Cassandra and HBase on Streaming data using apache Spark.**

Different number of iteration were used to evaluate the databases. Throughput and average latencies for both Cassandra and HBase were recorded and analyzed. A

trigger time was set to 5 seconds, 10 seconds, 5 minutes and 10 minutes. By average Cassandra was observed to have high performance on writing operation compared to HBase. Apache Spark Streaming was used as platform for evaluation while the Twitter Streaming API used as ingestion tool.

3.To propose which of the two NoSQL database is most suitable for streaming data using Spark

Apache Cassandra was found to be best on handling Streaming data using Apache Spark, it found to have the average of 69.14% better throughput compare to 30.399% that of HBase in all test performed in this dissertation. As the streaming data increase, throughput was found to increase while decreasing in average latencies time by 1.07%. For the case of continuous flowing of data and real-time analysis, Researcher suggest the use of Apache Cassandra.

5.2 Conclusions

Researcher sought that the choice of Databases to be used depend on the needs of the user. Studies has been conducted to compare databases with various opinion, reasons and needs. A study conducted by (Li & Manoharan, 2013) was about finding performance comparison between SQL and NoSQL. (Li & Manoharan, 2013) concluded that not in all cases NoSQL databases perform better than SQL in terms of read, write and delete operations.

The evaluation of NoSQL database has been conducted by (Ayush, 2014) by integrating with Hadoop as analytics tool. The study (Ayush, 2014) found Hadoop-MongoDB was not efficient in terms of read while Hadoop-Cassandra was more stable in write operation, the only limitation to them was Hadoop not to be designed

for analytics. According to (Sergey Bushik, 2012) found that Cassandra write operation was good in write operation when compared to HBase on processing batch data.

The findings in this study is much like that of (Ayush, 2014; Sergey Bushik, 2012) findings on the comparison of NoSQL databases, the only difference is that, this study was dealing with spark streaming data. Researcher chose Apache Spark as the framework to be on top to evaluate both Cassandra and HBase databases, because of its capability on handling Streaming Data. For the best comparison, researcher thought that common factors must be considered when you evaluate databases. In this study, only NoSQL database from column-family are taken.

Within the experiment, Researcher have managed to use Twitter Streaming API and constructed a dataset which receive tweets from twitter social network. Researcher went through several studies included literature review to accomplish objective (1) while chapter (3) and (4) used for the rest of objectives, therefore the main objective of this study can be considered accomplished.

After experiment, results show that Spark-Cassandra performed best when it comes write operations because as the streaming workload increase, its average latencies time was continuing to be minimal. NoSQL databases are considered to be fast, but Spark-Cassandra was not efficient in read streaming data operation. Spark-HBase did well when it comes to read operation as the workload increase. Therefore, suggest the use of Apache Cassandra on handling Streaming data is being suggested when apache Spark is used as analytic platform.

5.3 Future Work

Much work could be done on this experiment but due to time and financial constraints limitation, only few experiments were performed. The test was conducted for only 20 successively days and provided the interesting results on database arena, but much more work can be done to further study.

Other researcher can continue to do this experiment by involve other NoSQL Databases from Document, Key-Value and Graph database as this study deal only with Column-family database category. Also, the study can go further by evaluating NoSQL using Apache Storm as data link layer for streaming and real-time data, Flume or Kafka as data ingestion tool. This analysis can also be done by comparing the data partitioning capability of Spark-Cassandra and Spark-HBase in heterogeneous network when handling streaming data from different sources.

REFERENCES

- Adams, A., & McCorkindale, T. (2013), Dialogue and transparency: A content analysis of how the 2012 presidential candidates used twitter. *Public Relations Review*, 39(4), 357–359. <https://doi.org/10.1016/j.pubrev.2013.07.016>
- Agarwal. (2011a), Sentiment analysis of Twitter data. *Association for Computational Linguistics*, 30–38. Retrieved from <http://dl.acm.org/citation.cfm?id=2021109.2021114%5Cnpapers3://publication/uuid/83CA53FE-43D1-4BD5-BCF2-D55B82CF0F99>
- Agarwal. (2011b), Sentiment Analysis of Twitter Data. *Proceedings of the Workshop on Languages in Social Media*, (June), 30–38. Retrieved from <http://dl.acm.org/citation.cfm?id=2021109.2021114%5Cnpapers3://publication/uuid/83CA53FE-43D1-4BD5-BCF2-D55B82CF0F99%5Cnhttp://dl.acm.org/citation.cfm?id=2021109.2021114>
- Aniceto, R., Xavier, R., Guimarães, V., Hondo, F., Holanda, M., Walter, M. E., & Lifschitz, S. (2015), Evaluating the cassandra NoSQL database approach for genomic data persistency. *International Journal of Genomics*, 2015. <https://doi.org/10.1155/2015/502795>
- Apache. (2013), HBase. *The Apache Software Foundation*. Retrieved from <http://hbase.apache.org/>
- Apache HBase – Apache HBase™ Home. (2017), Retrieved from <http://hbase.apache.org/>
- Apache Spark. (2015), Apache Spark™ - Lightning-Fast Cluster Computing. Retrieved from <http://spark.apache.org/>
- Ayush. (2014), Performance Analysis of NoSQL Databases with Hadoop Integration, (July).
- Babcock, B., & Babu. (2002), Models and Issues in Data Stream Systems. *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 1–16. <https://doi.org/10.1145/543614.543615>
- Barata. (2014), YCSB and TPC-H: Big data and decision support benchmarks. *Proceedings - 2014 IEEE International Congress on Big Data, BigData Congress 2014*, 800–801. <https://doi.org/10.1109/BigData.Congress.2014.128>
- Barata, M. (2015), Cassandra : what it does and what it does not and benchmarking. *Int. J. Business Process Integratoin and Management*, 7(4), 364–371. <https://doi.org/10.1504/IJBPIIM.2015.073658>

- Benhardus, J., & Kalita, J. (2013), Streaming trend detection in twitter. *International Journal of Web Based Communities*, 9(1), 122–139. <https://doi.org/10.1504/IJWBC.2013.051298>
- Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., ... Moran, C. (2010), IBM Infosphere Streams for Scalable, Real-time, Intelligent Transportation Services. *Sigmod*, 1093–1104. <https://doi.org/10.1145/1807167.1807291>
- Bifet, A., & Frank, E. (2010), Sentiment knowledge discovery in Twitter streaming data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 6332 LNAI, pp. 1–15). https://doi.org/10.1007/978-3-642-16184-1_1
- Brewer, E. (2012), CAP twelve years later: How the “rules” have changed. *Computer*, 45(2), 23–29. <https://doi.org/10.1109/MC.2012.37>
- Cassandra, A. A. (2016), Apache Cassandra TM 2.2, (July 2008).
- Cooper. (2010), Benchmarking cloud serving systems with YCSB. *Proceedings of the 1st ACM Symposium on Cloud Computing - SoCC '10*, 143–154. <https://doi.org/10.1145/1807128.1807152>
- Cozby, P. C. (2015), *Methods in Behavioural Research. 12th Ed.*
- Creswell, J. W. (2014), *research design.*
- Databricks - Making Big Data Simple. (2017), Retrieved July 29, 2017, from <https://databricks.com/>
- Datastax. (2015), Benchmarking Top NoSQL Databases, (April), 13. Retrieved from <http://www.datastax.com/wp-content/uploads/2013/02/WP-Benchmarking-Top-NoSQL-Databases.pdf>
- Dissertation, A. (2006), February 2006 c Copyright by Arvind Arasu 2006 All Rights Reserved ii, (February).
- Eric Redmond, J. R. W. (2012), *Seven Databases in Seven Weeks A Guide to Modern Databases and the NoSQL Movement.*
- Fiannaca, A. J., & Huang, J. (2015), Benchmarking of Relational and NoSQL Databases to Determine Constraints for Querying Robot Execution Logs.
- Gandini, A., & Gribaudo. (2014), Performance evaluation of NoSQL databases. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8721 LNCS, 16–29. https://doi.org/10.1007/978-3-319-10885-8_2

- Gedik, B., & Andrade, H. (2012), A model-based framework for building extensible, high performance stream processing middleware and programming language for IBM InfoSphere Streams. *Software - Practice and Experience*, 42(11), 1363–1391. <https://doi.org/10.1002/spe.1139>
- Go, A., Huang, L., & Bhayani, R. (2009), Twitter Sentiment Analysis. *Entropy*, (June), 17. https://doi.org/10.1007/978-3-642-35176-1_32
- Gokavarapu. (2010), Exploring Cassandra and HBase with BigTable Model. *Pragmatic Programming Techniques*. Retrieved from <http://horicky.blogspot.com/2010/10/bigtable-model-with-cassandra-and-hbase.html>
- Gokavarapu, H., & Qiu, J. (n.d.). Exploring Cassandra and HBase with BigTable Model.
- Guller, M. (2015), *Big Data Analytics with Spark*.
- Gupta, R., Gupta, H., & Mohania, M. (2012), Cloud computing and big data analytics: What is new from databases perspective? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7678 LNCS, pp. 42–61). https://doi.org/10.1007/978-3-642-35542-4_5
- Holik, F., & Horalek. (2015), Effective penetration testing with Metasploit framework and methodologies. In *CINTI 2014 - 15th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings* (pp. 237–242). <https://doi.org/10.1109/CINTI.2014.7028682>
- IBM Streams. (2016), Retrieved November 9, 2016, from <http://www-03.ibm.com/software/products/en/ibm-streams>
- Jacobs, R. (1993), Computer Simulation in Management Science - Third Edition (Book). *European Journal of Operational Research*, 71(1), 137–138. Retrieved from <http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=8512200&site=ehost-live>
- Jiang, Y. (2012), *Hbase Administration Cookbook*. Retrieved from <http://books.google.com/books?id=QDZm1juBH1YC&pgis=1>
- Kalakanti. (2015), A comprehensive evaluation of NoSQL datastores in the context of historians and sensor data analysis. *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 1797–1806. <https://doi.org/10.1109/BigData.2015.7363952>
- Karau, H., & Konwinski. (2015), *Learning Spark*.
- Klein. (2015), Performance Evaluation of NoSQL Databases: A Case Study. *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, 5–10. <https://doi.org/10.1145/2694730.2694731>

- Kumar Vavilapalli, V., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... Baldeschwieler, E. (2013), Apache Hadoop YARN: Yet Another Resource Negotiator. *SOCC '13 Proceedings of the 4th Annual Symposium on Cloud Computing*, 13, 1–3. <https://doi.org/10.1145/2523616.2523633>
- L. Christensen, R. J. (2015), *Research Methods, Design and Analysis- Twelfth Edition*.
- Lakshman, S., Melkote, S., Liang, J., & Mayuram, R. (2016), Nitro: A Fast, Scalable In-Memory Storage Engine for NoSQL Global Secondary Index, 9(13), 1413–1424.
- Li, Y., & Manoharan, S. (2013), A performance comparison of SQL and NoSQL databases. In *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings* (pp. 15–19). <https://doi.org/10.1109/PACRIM.2013.6625441>
- Lynch. (2014), Perspectives on the CAP Theorem Accessed Detailed Terms Perspectives on the CAP Theorem, 0–10.
- Maarala, A. I., & Rautiainen. (2015), Low latency analytics for streaming traffic data with Apache Spark. In *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015* (pp. 2855–2858). <https://doi.org/10.1109/BigData.2015.7364101>
- Makris, A., & Tserpes, D. (2016), A Classification of NoSQL Data Stores Based on Key Design Characteristics. In *Procedia Computer Science* (Vol. 97). <https://doi.org/10.1016/j.procs.2016.08.284>
- Morstatter, F., Pfeffer, J., Liu, H., & Carley, K. (2013), Is the sample good enough? Comparing data from Twitter’s streaming API with Twitter’s firehose. *Proceedings of ICWSM*, 400–408. https://doi.org/10.1007/978-3-319-05579-4_10
- Nabi, Z., Wagle, R., & Bouillet, E. (2015), The best of two worlds: Integrating IBM InfoSphere Streams with Apache YARN. In *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014* (pp. 47–51). <https://doi.org/10.1109/BigData.2014.7004443>
- Nicoll, A. (2011), Stuxnet: targeting Iran’s nuclear programme. *Strategic Comments*, 17(2), 1–3. <https://doi.org/10.1080/13567888.2011.575612>
- Penetration Testing Software | Metasploit. (2017), Retrieved November 9, 2016, from <https://www.metasploit.com/>
- Ranjan, R. (2014), Streaming Big Data Processing in Datacenter Clouds. *IEEE Cloud Computing*, 1(1), 78–83. <https://doi.org/10.1109/MCC.2014.22>

- Roesch, M. (1999), Snort: Lightweight Intrusion Detection for Networks. *LISA '99: 13th Systems Administration Conference*, 229–238. <https://doi.org/http://portal.acm.org/citation.cfm?id=1039834.1039864>
- Sadalage, P., & Fowler, M. (2012), NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. *Vasa*. <https://doi.org/0321826620>
- Sergey Bushik. (2012), A vendor-independent comparison of NoSQL databases : Cassandra, HBase, MongoDB, Riak. *Altoros*, 1–6.
- Shanahan, J. G., & Dai, L. (2015), Large Scale Distributed Data Science using Apache Spark. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, 2323–2324. <https://doi.org/10.1145/2783258.2789993>
- Sharma, S., & Tim. (2015), Classification and Comparison of NoSQL Big Data Models. *International Journal of Big Data Intelligence*, 2(3), 201–221.
- Sukhdev, D. V. (n.d.). EVALUATING CASSANDRA , MONGO DB LIKE NOSQL DATASETS USING HADOOP STREAMING, 1–4.
- Sullivan, D. (2015), *NoSQL Fore Mere Mortals*.
- Taylor, J. (2012), Oracle Real Time Decisions (RTD) Product Review.
- Terry, D., Goldberg, D., Nichols, D., & Oki, B. (1992), Continuous queries over append-only databases. *ACM SIGMOD Record*, 21(2), 321–330. <https://doi.org/10.1145/141484.130333>
- Tolerance, F. (2009), Sharding , Amazon , and the Birth of NoSQL, 39–51.
- Veronika Abramova. (2014), Experimental Evaluation of Nosql Databases. *International Journal of Database Management Systems (IJDMS) Vol.6, No.3, June 2014*, 6(3), 1–16. <https://doi.org/10.5121/ijdms.2014.6301>
- Wang, H., Can, D., Kazemzadeh, A., Bar, F., & Narayanan, S. (2012), A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, (July), 115–120. <https://doi.org/10.1145/1935826.1935854>
- White, T. (2012, Hadoop: The definitive guide 4th Edition. *Online*, 54, 258. <https://doi.org/citeulike-article-id:4882841>
- Wireshark ·Go Deep. (2017), Retrieved November 9, 2016, from <https://www.wireshark.org/>

- Yang, H., Dasdan, A., Hsiao, R.-L., & Parker, D. S. (2007), Map-reduce-merge. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07* (p. 1029). <https://doi.org/10.1145/1247480.1247602>
- Yulevich, Y., Pyasik, A., & Gorelik, L. (2012), Anomaly detection algorithms on IBM InfoSphere streams: Anomaly detection for data in motion. In *Proceedings of the 2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012* (pp. 301–308). <https://doi.org/10.1109/ISPA.2012.145>

APPENDICES

Appendix I: External Examiner Corrections

Presented Chapters	Comments from External Supervisor	Correction Area done by Candidate	Page number.
(a) Chapter 1:	Proposed to edit the second specific objective and its corresponding question	Second specific objective with its corresponding question has been edited successful	Page numbers 13
(b) Chapter two: Literature Review	-contradiction on statement in relation to years of publication at section 2.2.4 -to complete literature review with research gap	-correction has been made to the actual years of publication at section 2.2.4 -research gap added at section 2.3	Page number 19,22
(c) Chapter 3 Research Methodology	-In methodology one presents what is to be done, how and why to enable another researcher to repeat what the candidate did in realizing the different research questions. This is not coming out clearly. -Figures and	-The whole chapter was restructured and improved by numbering the table and figures using chapter-wise format. -justification has been added at section 3.2	Page number 23 -53

	<p>numbers must be numbered using chapter wise format.</p> <p>-to remove part of literature review or to use justification for a selection part.</p>		
(d) Summary, conclusion and future work	<p>-improve the summary by providing quantitative findings and what it means.</p> <p>-Conclusion should be sharper and link to research objective/ questions</p>	<p>section was improved as instructed at sections 5.1 and 5.2.</p>	<p>Chapter five Sections 5.1 and 5.2</p>
(e) Dissertation Presentation and Writing	<p>-There are a lot of grammar, punctuation and other issues some of which are highlighted in the dissertation.</p> <p>-figures and tables must be numbered using chapter wise format</p> <p>-quality of figure must be improved</p>	<p>-figures and tables for the whole dissertation has been improved, grammar and punctuation mistakes has been collected.</p>	<p>The whole Dissertation</p>

