Spring 2015

Studying the effect of multi-query functionality on a correlation-aware SQL-to-mapreduce translator in Hadoop version 2

Thivviyan Amirthalingam
Purdue University

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Thivviyan Amirthalingam

Entitled
STUDYING THE EFFECT OF MULTI-QUERY FUNCTIONALITY ON A CORRELATION-AWARE SQL-TO-MAPREDUCE TRANSLATOR IN HADOOP VERSION 2

For the degree of Master of Science

Is approved by the final examining committee:

John Springer
Chair

Victor Barlow

Eric Matson

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): John Springer

Approved by: Jeffrey Whitten 4/28/2015
Head of the Departmental Graduate Program Date

STUDYING THE EFFECT OF MULTI-QUERY FUNCTIONALITY ON A CORRELATION-AWARE SQL-TO-MAPREDUCE TRANSLATOR IN HADOOP VERSION 2

A Thesis

Submitted to the Faculty

of

Purdue University

by

Thivviyan Amirthalingam

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2015

Purdue University

West Lafayette, Indiana

ஈன்ற பொழுதின் பெரிதுவக்கும் தன்மகனைச்

சான்றோன் எனக்கேட்ட தாய்.

(குறள் 69)

மகன் தந்தைக் காற்றும் உதவி இவன்தந்தை

என்நோற்றான் கொல்லெனும் சொல்.

(குறள் 70)

For, Amma and Appa, my parents.

ACKNOWLEDGEMENTS

My sincere gratitude to my major professor, Dr. John Springer, for his support and guidance has made this thesis project possible and, instilled a lifelong learning experience.

I also thank my committee members, Dr. Eric Matson, Professor Julie Mariga and Professor Victor Barlow for their involvement and feedback throughout this project's duration.

A special note of thank you is also due to the Information Technology at Purdue (ITaP) and its Research Computing (RCAC) division for their technical support and recommendations.

Last but not the least, I would like to also express my appreciation to my family, significant other and friends for their love, support and, the occasional, sanity checks.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

Correlation-aware translator - A model that identifies the correlation between operations within a single complex query (Lee et al., 2011).

Execution performance - The execution time it takes for a particular system to complete a single or a set of specific function(s) and/or instructions (Gu et al., 2014).

Hadoop - According to Apache's official online definition of Hadoop, it is a framework for a distributed processing of large amount of data balanced across nodes utilizing simple programming models.

MapReduce - "MapReduce is a programming model and an associated implementation for processing and generating large data sets" (Dean & Ghemawat, 2004, pg.1).

SQL - According to IBM's online SQL Guide, Structured Query Language (SQL) is a set-oriented mean for the users to allow query, update and delete tables of in a relational database environment.

LIST OF ABBREVIATIONS

CPU: Central Processing Unit

DDL: Data Definition Language

DML: Data Manipulation Language

ITAP: Information Technology at Purdue

NOSQL: Not-only Structured Query Language

SQL: Structured Query Language

TPC: Transactional Performance Processing Council

UDF: User Defined Functions

XML: Extensible Mark-up Language

YSMART: Yet another SQL-to-MapReduce Translator

ABSTRACT

Amirthalingam, Thivviyan. M.S., Purdue University, May 2015. Studying the Effect of Multi-query Functionality on a Correlation-aware SQL-to-MapReduce Translator in Hadoop version 2. Major Professor: Dr. John Springer.

The advent of big data has prompted both the industry and research for numerous solutions in catering to the need for data with high volume, veracity, velocity and variety properties. The notion of ever increasing data was initially publicized in 1944 by Fremont Rider, who argued that the libraries in American Universities are doubling in size every sixteen years (Press, 2013). Then, when the digital storage era came to be, it became easier than ever to store and manage large volumes of data. The need for efficient big data systems is now further fueled by the “Internet of Things” as it opens floodgates for, never before seen, new information flow.

These phenomena have called for a simpler and more scalable environment with high fault tolerance and control over availability. With that motivation in mind, and as an alternative to relational databases, numerous Not-Only Structured Query Language (NoSQL) databases were conceived. Nonetheless, relational databases and their de facto language, Structured Query Language (SQL) are still prominent among wider user groups.

This thesis project ventures into bridging the gap between Hadoop and relational databases through allowing multi-query functionality to a SQL-to-MapReduce translator. In addition to that, this research also includes the upgrade of the translator to a newer Hadoop version to utilize newer tools and features added since its original deployment.

This study also includes the analysis of the modified translator’s behavior under different sets of conditions. A regression model was devised for each of the experiments made and presented as significant means of understanding the data collected and any future estimates.

CHAPTER 1. INTRODUCTION

MapReduce and Hadoop have become a game-changing paradigm in big data analytics mainly due to their fault tolerance and scalability. However, relational databases and SQL-like queries still play a prominent role within the user communities. Such phenomena, i.e., the popularity of relational databases, have triggered multiple investigations on effective translation model between the relational and NoSQL data models.

This research is an extension of a paper, *YSMART: Yet another SQL-to-MapReduce Translator* (2011). The aforementioned paper, written by multiple researchers at the Ohio State University with the collaboration of Facebook, discusses a new optimized method of translating SQL queries to MapReduce code by considering the correlation between the operations within a single complex query (Lee et al., 2011).

Lee et al. propose that by applying a set of rules to minimize the number of MapReduce jobs, YSMART can significantly “reduce redundant computation, I/O operations and network transfers compared to existing translators” such as Hive and Pig (Lee et al., 2011, p. 1). Although YSMART optimizes a complex SQL query plan tremendously, there are still many ways of improving it and its underlying Hadoop framework especially in the case of multiple queries.

This research dived into the possibility of introducing multi-job functionality to the translator design to allow multiple SQL queries to be translated collectively. In addition to that, it also included an upgrade for Hadoop version 2 compatibility.

1.1 Research Question

What is the behavior observed when the current implementation of a correlation-aware SQL-to-MapReduce translator is integrated with multi-query functionality on Hadoop version 2?

1.2 Scope

The popularity of Hadoop and the MapReduce framework has led to many investigations on making it better and, ultimately, faster. Current business need demands nothing less than the best solution for big data analytics. Resultantly, every single aspect of the framework is studied for optimization.

Rong Gu et al. (2011) categorized the efforts of improving the Hadoop MapReduce framework into four different classes: (a) optimization through scheduling algorithms; (b) optimization through specialized hardware and software; (c) optimization for specific MapReduce application; and (d) optimization through the configuration settings and parameters. On the other hand, the prevalent acceptance of relational databases and SQL queries has also led to multiple studies on how to incorporate a query-like language with MapReduce.

This research was aimed at introducing multi-job functionality to the current implementation of the correlation-aware SQL-to-MapReduce translator (Lee et al., 2011) and verifying its compatibility with Hadoop version 2. It will not discuss optimization methods discussed later in Chapter 2. Furthermore, to provide consistency, all the tests performed were in the same environment. This ensures that external factors (e.g., difference in CPU, memory, network capacity and disk performance) do not interfere with the results collected.

Additionally, this research focused on a set of specific benchmarked complex queries such as TPC-H for the translation process. In most cases, simple queries do not allow a correlation to be formed due to their naive nature. Complex queries, in contrast, provide more robust testing scenarios for the experiment.

1.3 Significance

As mentioned earlier, SQL queries have been the predominant form of interface between users and relational data systems. Although the NoSQL environment offers a totally new approach to data storage, retrieval and management, users still expect similar query-like languages. This has led to the advancements of Hive and Pig, the major high-level SQL-like querying languages on the market (Wang & Chan, 2014).

Despite their popularity, Hive and Pig pose a serious performance issue for certain complex queries. This is due to the one-operation-to-one-job nature of these translators (Lee et al., 2011). In response, the correlation-aware SQL-to-MapReduce translator was introduced. This method identifies the correlation within a single complex query and reduces the execution time tremendously compared to Hive and Pig.

This research has added to that framework and introduced the multi-job functionality to the underlying MapReduce framework. Such investigation would not only allow multiple queries to be processed by the correlation-aware translator but also optimize the overall process. This offers an efficient batch-query processing approach especially with a set of complex queries.

Furthermore, the multi-query functionality would also take advantage of any similar processing (e.g., scanning similar input files and/or tabulating similar key-value outputs) and allow for more opportunities for reduced computation and I/O processing (Wang & Chan, 2014).

1.5 Assumptions

This study includes the following assumptions:

- The inter-node network traffic overhead is assumed to be miniscule and ignorable.
- The network speed is assumed to be constant.
- The selected SQL queries contain the necessary complexity for the optimization application.
- The results collected from the test environment will be proportionate or almost proportionate when scaled to a high-processing environment.
- The application of the multi-query optimization is independent of the underlying hardware and software configurations
- All selected queries allow multi-query optimization through Hadoop's underlying architecture.
- The results are collected in a controlled environment and are not affected by external factors.

- The regression model formulated from the results would be the best descriptor for future estimates.

1.6 Limitations

This study includes the following limitations:

- This research studies the effect of incorporating the multi-query functionality methods into correlation aware SQL-to-MapReduce translator design by Lee et al. (2010).
- This research only considers selected SQL queries from the TPC-H benchmark series.
- This research studies the behavior of the new implementation through general inferences, hypotheses testing and linear regression models.
- All tests are executed in a single test environment for consistency.

1.7 Delimitations

This study includes the following delimitations:

- This research only studies the time complexity of a given set of query translations. It does not include any other complexity measurements.
- Optimization is identified in the multi-query read level. Any other optimization such as, including but not limited to, software/hardware optimization, execution optimization, algorithm optimization, etc. will not be included as a part of this study.

- This study does not include comparisons between the proposed model and translators such as Tenzing, MRPacker, YSMART, etc.
- This research does not include SQL queries that (a) are simplistic in nature, (b) do not allow correlation to be formed and (c) do not allow multi-query generalization method to be applied.

1.5 Chapter Summary

This chapter introduces the general idea of the proposed research. It outlines key aspects such as the research question, scope, definitions, assumptions, limitations and delimitations.

CHAPTER 2. LITERATURE REVIEW

The advent of big data has shifted the paradigm of data and its processing methods. An abundance of data collected through new and precise devices like motion detectors, heat sensors, web traffic data collectors, etc. In addition to traditional data collecting methods, the amount of data streaming in from these new devices has made it impossible to capture, store, analyze and maintain data via a relational database model.

However, relational database systems are still key players in the data management approach. The 20 year-plus old technology has placed its imprint on a majority of the user groups and it has shown to be commonly implemented for various purposes. In addition to that, Online Transaction Processing, or more commonly known as OLTP, has fueled the motivation for the relational databases in the fast-paced online world. The prevalence of relational databases eventually led to the popularity of Structured Query Language (SQL), the de facto platform for the user-data-system communication.

These phenomena have inspired the author to look into the means of effective translation models between a relational database and a NoSQL environment. The following chapter will explore previous works that were made in this area. The subsequent pages are organized as follow: (a) Hadoop/MapReduce Overview, (b) High Level Declarative Languages, (c) SQL-MapReduce Translators and (d) MapReduce Optimizers.

2.1. MapReduce Overview

Mass information gathered on the day-to-day basis is only as useful as the knowledge that can be derived from it. However, current performance of relational database solutions and their analytical tools degrade when substantial amount of data is introduced. According to Deshpande (2011), there are a number of complications when designing a large-scale data processing system. Some of them include:

1. Managing the environment's processors to adapt to the large data sets.
2. Incorporating parallelization and distribution without introducing excessive overhead costs.
3. Managing input file flows.
4. Providing fault tolerance and high availability.

To overcome these problems, Dean and Ghemawat (2008) proposed the MapReduce programming model. Designed at Google, MapReduce simplifies large data computation and storage by allowing users to specify "a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key" (Dean & Ghemawat, 2008, p.1).

The programming model allows large data parallelization and distribution over multiple clusters of commodity machines (Dean & Ghamawat, 2008). In addition to that, it also automates the load balancing and locality optimization of a system. This allows the users to be able to focus on the business logic rather than the technical implementation for a particular system.

2.2. <u>High Level Declarative Languages</u>

As mentioned before, though big data solutions, e.g., Hadoop, offer a better means of storing and computing large quantities of data, they have yet to fully influence the common user groups due to the dominant popularity of relational databases and SQL. The relational approach, which has been popular for more than two decades now, has caused the user community to avoid the possibility of working with non-structured data models.

To overcome these adaptation problems, many investigations have been made in introducing a SQL-like, high-level declarative language for the Hadoop environment. In this section, two such implementation will be discussed.

2.2.1. Hive

Hive is an implementation of a SQL-like, high-level declarative language that runs on top of a Hadoop environment. The query language used in Hive, HiveQL, converts SQL-like commands to MapReduce codes that can be run across multiple Hadoop clusters. In addition to that, Hive also allows users to create custom MapReduce jobs that can be plugged into SQL codes (Thusoo et al., 2009).

Thusoo et al. (2009) also said that HiveQL provides a wide range of SQL-like query commands. This includes the select, project, join, aggregate and union-all statements. In addition to that, the query language allows some SQL Data Definition Language (DDL), i.e., create, update and delete table statements. It also allows some SQL Data Manipulation Language (DML) that contains functionalities such as load tables, update tables and insert into tables.

Although HiveQL provides a wide range of SQL-like command, it is merely a subset of the actual SQL syntax. This is deemed to be a flaw in the implementation, especially for queries that require unique, less-commonly-used set of SQL commands. After all, Thusoo et al. (2009, 2010) have identified their research to be a work-in-progress and acknowledged a number of improvements that can be made in their first Hive implementation, some of which are listed below:

5. The limited syntax of HiveQL
6. Naïve-based optimizer with minimal rules for optimization
7. Limited connectivity with other applications and Application Programming Interface (API) that does not support Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC)
8. Better data placement methods in a columnar storage system
9. Better multi-query optimization

2.2.2. Pig and Pig Latin

Another prominent high-level declarative language for the MapReduce environment would be Pig. Introduced in mid-2008 by the researchers at Yahoo!, Pig offers a compromising solution between the rigid, procedural MapReduce programming and the somewhat flexible, declarative SQL queries. Olston et al. (2008) argued that the procedural MapReduce model allows scalability and reliability through the Hadoop architecture whereas the SQL method is widely known and user-friendly. With that motivation, Olston et al. (2008) ventured in creating a low-level programming language, Pig Latin, for the Pig environment.

During the initial proposal of Pig, Olston et al. (2008) also pointed out some of the flaws present in the MapReduce environment. This includes the rigidity of the language, incompatibility of dataflow in joins or in n-stages and its unsuitability with common functions like projection and filtering. Thus, Pig Latin was created to combine the best of the high level, declarative queries and the low-level, procedural MapReduce programming models. Olston et al. (2008) believe that their proposed language, Pig Latin, is superior to the common MapReduce programming models due to the following reasons:

1. Dataflow-oriented language: Pig is a platform where a user provides a set of instructions where each instruction resembles a high-level data transformation. This allows more control for the programmers to manipulate large amounts of data with the least amount of hassle possible.
2. Rapid start and interoperability: Pig also offers a fast ad hoc query transformation service for its users. This is made possible by allowing users to run queries straight on an input file – provided that the tuple function is present in the query. Similarly, the output of a Pig query can be formatted into multiple forms, allowing for more flexibility. In addition to that, Olston et al. (2008) proved that the Pig environment is interoperable with other environments by placing it in the Yahoo! ecosystem.
3. User Defined Functions (UDF): Possibly one of its main attractions, Pig's ad hoc query method provides means of transforming user defined function into useful MapReduce codes. This even includes complex operations like process groupings, filtering and joining.

The implementation of Pig was extended by Gates et al. (2009); in their study, they evaluated the performance and identified the challenges of implementing a Pig environment on a large-scale basis. In addition to that, they have also listed some areas of improvement for the environment – some of which are listed below:

1. The limitation on optimization due to the ad hoc querying
2. UDF incompatibility with non-Java interface
3. Improvements on the SQL user interface
4. Allowing grouping and joining pre-partition
5. Better skew handling

2.3. SQL-MapReduce Translators and Optimizers

Through the high level declarative languages, the translation of SQL queries was investigated for a more robust SQL-to-MapReduce translator. One of such efforts was by Zhang, Wang and Han (2011) in their paper, Middleware Design for Integrating Relational Database and NoSQL based on a Data Dictionary. This research, inspired by the WEB 2.0 boom and the need for big data solutions for the web, focuses on a data dictionary model that acts as a middleware layer between a relational database and a NoSQL environment. Zhang, Wang and Han (2011) identified a mean of integrating the structured nature of relational databases and the rather loosely structured NoSQL environment through a common application that keeps track of relevant information on both ends.

Zhang, Wang and Han (2011) also promoted the notion that, despite the name, NoSQL environments have a structure in their system – it is just broad and less conventional to what is seen on the traditional data models. The researchers utilized this fact to form an effective data dictionary.

The proposed database integration has three main components in its architecture: (a) Middleware client, (b) Middleware server and (c) SQL engine. As the name suggests the middleware client resides in the application layer and serves the communication between the middleware and the application itself. The middleware server, on the other hand, is proposed to be a device with distributive data transmission capabilities. This component receives the user input and then analyzes and transfers the request to the appropriate data model with the help of the data dictionary. The last component of the architecture, the SQL Engine, “converts standard SQL requests into various types of access to the NoSQL database” (Zhang, Wang and Han, 2011, p. 1470).

The notion of assigning a structure to a NoSQL environment has triggered other investigations, including the author’s proposed research.

Other than the data dictionary implementation, other means for proper middleware solutions have been investigated. In view of that, the Extensible Markup Language (XML) was looked into as a possible middleware layer between a relational and NoSQL environment. XML, according to the W3C web domain definition (2014), is a flexible text-file format that is “designed to meet the challenges of the large-scale electronic publishing.” While it was created for the purpose of transferring data between websites, it is still a good candidate to hold information and provide a communication platform between database models.

Yu, Wang and Hua (2012) addressed the problems of integrating multiple relational database “species” and the communication complexities arise in doing so. In many cases, large organizations would maintain multiple databases with multiple database management systems. The management systems, although provide similar relation-based data model, are still different than each other. This may eventually lead to excessive administrative, maintenance and support costs to an organization. On the other hand, just having one relational database management system for an entire organization might not be a good idea due to factors such cost feasibility and project requirements.

To overcome such problems, Yu, Wang and Hua (2012) utilized the XML's compatibility with the relational database management systems and designed a middleware solution that imports and exports of SQL commands in XML formats. Though this laid the foundation for much more research, the solution proposed by Yu, Wang and Hua (2012) only focused on the inter-relational database translation.

While the inter-relational database communication was introduced and developed, another stream of investigations were made for an effective middleware design for the traditional databases and NoSQL environment. Su and Swart (2012) looked into ways of translating Oracle SQL queries to MapReduce codes. Their effort establishes a translation model within the Oracle relational database management system.

Su and Swart's in-database framework was based on the parallel query processing of the Oracle engine. This allows the solution to "partition the input data, instantiate mapper and reducer tasks, and schedule the computation using the database's computing resources" (Su and Swart, 2012, pg. 782). The design incorporates MapReduce tasks in an Oracle database therefore eliminating the need for a separate Hadoop HDFS system.

Su and Swart (2012) also identified some of the issues of successfully incorporating the MapReduce framework inside an Oracle database instance. First and foremost, the Java implementation of MapReduce is very different from the one present in the relational database's architecture. Functions such as JobTracker and TaskTracker that run on a multiple node environment are not natively available in Oracle and its architecture. Hence, a proper replication of the Hadoop environment, including the Java Virtual Machine and Hadoop Distributed File Systems, is required.

Other than that, the data-type incompatibility between the Hadoop and Oracle environment also poses a roadblock to Su and Swart's proposed design. Many Oracle data-types are immutable – non-overwritten values that are discarded once they are no longer needed. Hadoop, on the other hand, allows mutable files where values are overwritten to reduce the allocation and garbage collection overhead (Su & Swart, 2012).

Last but not the least, the incompatibility of input and output file formats was also identified as a major problem by Su and Swart (2012). The MapReduce framework requires inputs and outputs types to be in key-value pairs whereas the database management system interacts with the data through file-typed data-types. This discrepancy requires the translator model proposed by Su and Swart to be able to convert data to/from object types and file types without much processing overhead.

In addition to Su and Swart's work, the designs for database-independent SQL-to-MapReduce translators were also explored. One of the pioneers in the topic, Lee et al. (2011), introduced a correlation-aware SQL-to-MapReduce translator, YSMART, where the correlation within a single complex query is optimized before parsing it into MapReduce codes.

The research by Lee et al., inspired by the unacceptably high time complexity of common translators when processing complex queries, proposes YSMART outperforms existing translators like Hive and Pig tremendously. The slow performance is due to the one-job per operation nature of high-level declarative languages (i.e., Hive and Pig) that ultimately results in more computation, I/O operations and network transfers for complex query processing (Lee et al., 2011).

As part of their research, Lee et al. looked into three types of intra-query association models: (a) input correlation, (b) transit correlations and (c) job flow correlations. Each of these is formed through a standard set of rules. Then, an optimized MapReduce job is formed and sent to the Common MapReduce Framework (CMF) for further processing.

The work by Lee et al. (2011) was further elaborated by Lin, Ye and Ma in their paper: MR Packer: A SQL to MapReduce Optimizer (2013). In this study, the researchers proposed an optimized MapReduce execution plan generator by imposing a set of standard transformational rules to reduce the total number of MapReduce jobs. Similar to YSMART, MRPacker also finds means of using a set of guidelines for transforming the query to reduce the number of MapReduce jobs. This ultimately lowers the computational resources and storage I/O processing.

But, unlike YSMART, the creators of MRPacker believe that there can be multiple means of deriving the best solution, that is, MapReduce jobs with the lowest execution cost possible. For that reason, a SQL query is transformed into multiple MapReduce query plans and the best is chosen through an enumeration process.

2.4. MapReduce Optimizers

The large scaled query and data analysis also led to many efforts on making the big data solutions to effectively handle bigger volume of data in the shortest time possible. So naturally, the MapReduce framework was also studied for optimization.

According to Gu et al. (2014), MapReduce framework can be improved through: (a) special algorithms, (b) special hardware and software configurations, (c) configuring the MapReduce tasks and (d) optimizing job configurations. On the other hand, the proposed research is on the multi-query optimization of MapReduce jobs. This ultimately allows reduced computation, I/O and network transfers by utilizing the benefits of batch processing.

One of the prominent efforts in identifying the multi-query optimization was by done by Nykiel et al. (2010) through their work: MRShare – Sharing across Multiple Queries in MapReduce. Their work identified that, in most cases, multiple queries of a small data model often perform similar work and there is an opportunity to utilize that for reduced processing. Ultimately, Nykiel et al. (2010) proposed a sharing opportunity model that optimizes batch queries by combining identical jobs together to form a single MapReduce job.

Another important aspect of Nykiel et al. (2010) proposed in their work was the introduction of a cost model for MapReduce. This model, which assumes the execution time of a MapReduce job is heavily deterministic of its I/O operations, allows an estimation of time complexity to be calculated – provided the parameters are given and accurate.

MRShare also laid foundation for many other batch-querying optimizations. Wang and Chan (2014) proposed two new solutions in addition to MRShare's sharing opportunities: generalized grouping method and materialization technique. The generalized grouping technique relaxes the MRShare's sharing opportunity framework to enable more MapReduce jobs to be merged into a single job. On the other hand, the materialization technique offers an optimizing algorithm that enables multiple jobs to share a particular query's map input and map output scan (Wang & Chan, 2014).

Additionally, Wang and Chan (2014) also utilized the cost model as proposed by Nykiel et al. (2010) to estimate the time it takes to execute MapReduce jobs. This is pertinent to the proposed research as it shows that the cost model, though it only considers the disk and network I/O, is deemed to be a good instrument of calculating the time complexity of a MapReduce implementation.

2.5. Chapter Summary

This chapter summarizes the literature on the subject of (a) Hadoop/MapReduce, (b) High Level Declarative Languages, (c) SQL-MapReduce Translators and (d) MapReduce Optimizers. The concepts and techniques specified in each literature will definitely provide a strong foundation towards the research question of incorporating multi-query optimization to the correlation-aware SQL-to-MapReduce translation model.

CHAPTER 3. METHODOLOGY

The incorporation of multi-job optimization with the correlation-aware SQL-to-MapReduce translator requires precise, yet robust experimental design. This is due to the constrictive nature of the translator that uses a set of rules to form correlation within a single complex query. On the other hand, the multi-query functionality also requires a certain set of rules to be met before allowing any implicit optimization process.

Though the current implementation of SQL-like query translators, such as Hive and Pig, provide stable and reliable big data analytic solutions, they pose a great inefficiency for multiple complex query processing. This is due to the translators' one-operation-to-one-job translation mode where the intra-query and inter-query correlations are ignored. Consequently, this results in unacceptable time complexity especially when dealing with large number of complex queries.

The proposed research looked into the compatibility of the correlation-aware translator, as proposed by Lee et al. (2010) with the multi-query functionality on the latest Hadoop version. Furthermore, regression models of such combination (i.e., correlation-aware translation and multi-query functionality) were also studied. The optimization study, for the sake of this research, was mainly based on the time it takes to process a set of predetermined queries under a set of conditions, and this chapter will dive into the methodological details of this research.

While studying the time complexity optimization, it is also pertinent to understand how the external factors, or parameters, affect the proposed solution. As described by Wang and Chan (2014), the effectiveness of a MapReduce model can be tested by controlling four main parameters: the (a) cluster size, (b) number of queries, (3) data size and (4) split size. Because the time it takes for MapReduce jobs vary according to each of the aforementioned factors, some of these parameters, while others being constant, were studied in further detail in this research.

3.1. Software and Hardware Specifications

Both the translator and the multi-query functionality require a set of software configurations to be in place. The preliminary study has led the author to determine the following specification requirements:

- Linux (32 or 64 bit) operating system
- Java and GNU Compiler Collection (GCC) are set up
- Python configured

Additionally, Hortonwork’s Ambari sandbox was also investigated as an alternative, test environment for this research project. While the using a Linux machine provided more control to users through powerful commands, the sandbox option was also advantageous as it was made available with preset configurations and preinstalled Hive/Pig plugins necessary for the research. Other than that, the translator requires Java, GCC and Python to be properly installed and configured.

In addition to the software specifications, the research framework also required a cluster of nodes with master-worker architecture to replicate a true Hadoop environment.

As such requirement, the author has identified Purdue University's Hathi to be a viable option.

3.1.1. Hathi

Purdue University, through Information Technology at Purdue (ITaP) Research Computing (RCAC), provides a shared Hadoop cluster that is made available to partners in Purdue's Community Cluster Program. This shared resource, named Hathi, went into production on September 2014 and has been catering numerous research projects involving Hadoop since then. Hathi consists of 6 Dell nodes with two 8-core Intel E5 – 2650v2 CPUs, 64 GB of memory per node and 48 TB of local storage per node. Furthermore, according to the ITaP RCAC's user guide, all nodes have 40 Gigabit Ethernet connection and the hardware is in a 5-year warranty. The hardware specifications of the cluster are shown in Table 3.1.

Table 3.1 *Hardware Specifications of the Hathi cluster*

Specification	Value
Machine Type	Dell r720xd Servers
Processor Model	Intel E5 – 2650v2
Number of Nodes	6
Cores per Node	16
Memory per Node	64 GB
Memory in Total	384 GB
HDFS Storage per Node	48 TB
HDFS Storage in Total	288 TB

Additionally, Hathi runs on Red Hat Enterprise Linux Operating System, version 6 and use PivotalHD Hadoop Distribution for resource and job management. All updates and patches occur as security needs dictate.

The latest Hadoop distribution installed on Hathi is version 2.2.0.

3.2. Queries and Datasets

As indicated in the previous sections, the translator requires a set of predetermined queries. This is to ensure that the queries meet the complexity requirements for correlations to be formed within them. Therefore, as proposed by Lee et al. (2011), the author has elected to use the TPC-H benchmarked queries, in particular, Q6.

As for the datasets, a set of synthetic data will be used to populate tables according to the selected queries. The logical database model was made available in the Transaction Processing Performance Council (TPC) documentation. There were multiple synthetic data generators available in the market and the author has elected to use the Transactional Performance Processing Council's TPC Database Generator due to its compatibility and cost feasibility.

3.3. Independent Variables

As proposed by Wang and Chan (2014), an effective test of the optimized model involves varying four different types of parameters: the (a) data size, (b) cluster size, (c) number of queries and (d) split size. This research has tested the modified translator solution, i.e. with multi-query functionality in Hadoop version 2, and the current SQL-like query translator against varying degree of data sizes and number of queries. This ensures a robust testing environment to observe the changes in the time complexity when the parameters are altered. The following sub-sections describe the four main factors in further detail.

3.3.1. Data Size

The data volume in any big data environment is considered to be its signature and defining aspect. The sheer amount of data captured, stored, analyzed, transferred and presented in a NoSQL system makes it impossible to be managed by a traditional data processing system. The definition of “size” itself is considered to be an evolving aspect as devices with larger storage capacities are constantly introduced.

For that reason, the research explored the effect of large data sets against the correlation-aware SQL-to-MapReduce translator with the multi-query optimization. According to Afrati and Ullman (2010), a larger data set requires more reducer tasks as the number of reducer tasks should be proportional to data size. For that reason, the author has elected to use 60GB, 120GB and 180GB synthetic data sets.

3.3.2. Number of Queries

Another factor that was explored is the effect of the total number of queries against the different types of translators. In a normal scenario, the time complexity for a MapReduce job should increase almost proportionally with the number of queries. This is, in reality, not always true due to factors like CPU availability and network traffic. Nevertheless, this research has investigated the opportunities for reduced time complexity compared to other translators.

3.3.3. Cluster Size

Cluster size also plays an integral role in determining the time it takes to complete a set of MapReduce jobs. Naturally, the higher the number of clusters, the more distributive processing capabilities can be achieved even with large number of queries.

It is imperative that the proposed solution would be equally efficient in a large cluster environment. This ensures the scalability feature that is deemed to be essential in the big data realm. To replicate that, the author has elected to run set of queries against 6 nodes.

3.3.4. Split Size

The Hadoop File System, the default file system available on any Hadoop implementation, automatically splits any input file into smaller chunks. This is to enable the file system to process these data chunks in parallel and, ultimately utilize the multi-core architecture of high-performing computers. In addition to that, the split enables the Mapper to process the data more efficiently.

Split size also plays an important role in optimizing MapReduce jobs. Smaller split sizes, though provide better utilization of storage and disk blocks, also requires intensive I/O write and reads. This may lead to bottleneck issues. On the other hand, larger split sizes would result in higher, underutilized storage space. Considering this dilemma, it is only wise to set a default split size based on the data that is being processed. To show the proposed solution’s compatibility with Hadoop’s default split size, the author has elected to configure the split size to be at 512 MB.

3.4. Dependent Variable

The main dependent variable of this research was the time complexity, i.e., the time it takes for the system to fully translate multiple complex queries and execute appropriate MapReduce jobs.

3.5. Hypothesis

For each of the tests, as can be seen from Chapter 4, a hypothesis test for slope was conducted. In a linear regression test, a statistical method to identify relationship between two variables, the population is described by the slope and the intercept constant. Due its irrelevance, the intercepts are often not tested for significance. On the other hand, the slope of the regression line, usually denoted as β_x, is deemed to be the best descriptor for the relationship between the dependent and independent variables.

Accordingly, the following hypotheses can be formulated for each regression model generated in this research:

$$H_0: \beta_x = 0$$

$$H\alpha: \beta_x > 0$$

3.6. Procedure of Testing

This research has emulated a multi-query functionality to the correlation-aware SQL-to-MapReduce translator as proposed by Lee et. al (2010). To provide consistency, similar master-worker architecture was set up. In addition to that, the following Hadoop configuration was made constant throughout the research:

- I/O buffer cache size
- Default replication factor
- Number of concurrent mappers
- Number of concurrent reducers

Once the system is in place, the correlation aware SQL-to-MapReduce translator was installed and configured to allow multi-query optimization. A set of data sets were mounted on the system volume and approximately 100 trials will be run for each of the parameters discussed above. The number of trials will be reduced if the query results (a) display a similar pattern and (b) does not contain much discrepancy.

In addition to that, the hypothesis for slope was also tested using a t-test to determine whether the slope of the regression model is significantly different from zero. The t-test serves as the inference tool to assess the evidence provided by the data in favor or against the null hypothesis about the population. In the case of this research, the author has elected to analyze a regression model for each of the tests conducted. Moreover, in regards to the rejection of null-hypothesis, the author has considered a one-sided t-test and compared the p-value with a default significance level, i.e., alpha value, of 0.05.

3.7. Population and Sample

The population consists of TPC-H benchmarked queries. The queries, designed and administered by the Transactional Processing Performance Council (TPC), allow ad hoc queries that act as a decision support benchmark. These queries and their logical database model are elected due to its broad, industry-wide relevance. It also requires less maintenance and configurations thus, easing the implementation process. According to the TPC Benchmark H Standard Specification (2014), the TPC-H queries support systems that (a) examine large volumes of data, (b) execute complex queries and (c) answers critical business questions. The author has adopted these set of queries due to these advantages.

As mentioned earlier, the sample queries were a flattened versions of query Q6 of the TPC-H benchmark query set.

3.8. Measurement of Success

Since this research focused on the effect of multi-job processing on an existing translator, the measurement of success was mainly based on the time complexity of query translations and MapReduce job executions. For that, the author has formulated regression models that best describe collected data and any future estimates. Many researches in the same area have chosen a similar approach.

A measure of success, in the case of this research, involves generating a linear regression model that (a) rejects the null hypothesis for slope and (b) describes 90% or higher of the variations found in the results through its adjusted R-Squared value.

3.9. Threats and Weaknesses

The author acknowledges the following threats and weaknesses that may affect the research:

- Incompatibility between the cluster environment and the proposed solution
- Incompatibility between the cluster environment and existing SQL-like query translators

- Limitations on the current correlation-aware SQL-to-MapReduce translator
- General outliers in the collected data
- Unidentified external factors that may affect the time complexity

3.10. Chapter Summary

This chapter covers the key variables in the experiment along with the hypotheses that was tested in this project. It also describes the hardware specifications, software specifications, sample, population, test data and a general description on how the experiments were conducted.

CHAPTER 4. DATA COLLECTION AND ANALYSIS

The analysis for multi query functionality in a correlation aware SQL-to-MapReduce translator required a series of tests, designed to examine its compatibility, scalability and usability. Per se, the author has conducted multiple experiments, with varying degrees of manipulative variables to determine a regression model that best describes the data collected and future estimates.

A regression analysis is a statistical method used to investigate any relationship between two factors, if present. Though a simple fit plot between the dependent and independent variables would illustrate the general relationship between them, further analyses such as hypothesis testing, R-squared calculation and linear model generation, as presented in this chapter, will provide additional support to any discoveries.

This chapter presents its findings in three main categories of experiment: the new implementation's compatibility with (a) Hadoop 2.x.x, (b) large volume of data and (c) multiple query translations. In each experiment, a range of 60 to 100 data points were collected. As mentioned in Chapter 3, the number of trials in each experiments were highly dependent on the pattern observed during data collection. Data points with low variance had less number of runs, compared to groups with high range observed in their results.

Another major consideration for the number of runs was the total number of jobs generated as part of the query translation. In perspective, a translation that generates 10 jobs had less number of runs compared to a translation with only a single job. This was due to the time constraint on this project.

The results, once attained, were described through general inferences, hypothesis testing and regression analyses. More details can be found in the subsequent pages of this chapter.

4.1. Experiment 1: Compatibility with Hadoop version 2

Apache, the governing body of Hadoop, has been distributing and maintaining different versions of Hadoop since 2007. At the time of this thesis write-up, the earliest supported version is Hadoop 0.14.x, which was released on September 2007, and, the latest version is Hadoop 2.6.0, which was made available to the public on November 2014. While each releases improves its previous predecessors, three main upgrades were made to Hadoop at the time of this document's inception, the latest being Hadoop Version 2.

Any of the 2.x.x Hadoop versions provide an unparalleled advantage over 1.x and 0.x versions with better distributive resource management through YARN. While YSMART was originally conceived for Hadoop 0.x and 1.x, since 2008, many improvements were made to the Hadoop environment since then. This section will illustrate the compatibility of the modified translator with Hadoop version 2 releases, in particular with Hadoop 2.2.0.

4.1.1 General Inferences

For this test, a simple SELECT statement query was translated and ran against a single table of varying sizes, (a) 2.5 GB, (b) 5 GB, (c) 10 GB and (d) 15 GB. Each SQL statement translation generated one MapReduce job and 30 runs for each file size were conducted. Consequently outliers were identified through an IQR outlier test and removed from the dataset. The initial findings can found in Table 4.1.

Table 4.1 *Average Time Taken for Experiment 1 Job Execution*

File Data Size (GB)	Average Time Taken (mm:ss)
2.5	01:17
5	02:07
10	04:16
15	07:59

The box plot in Figure 4.1 describes the range and distribution of the dataset.

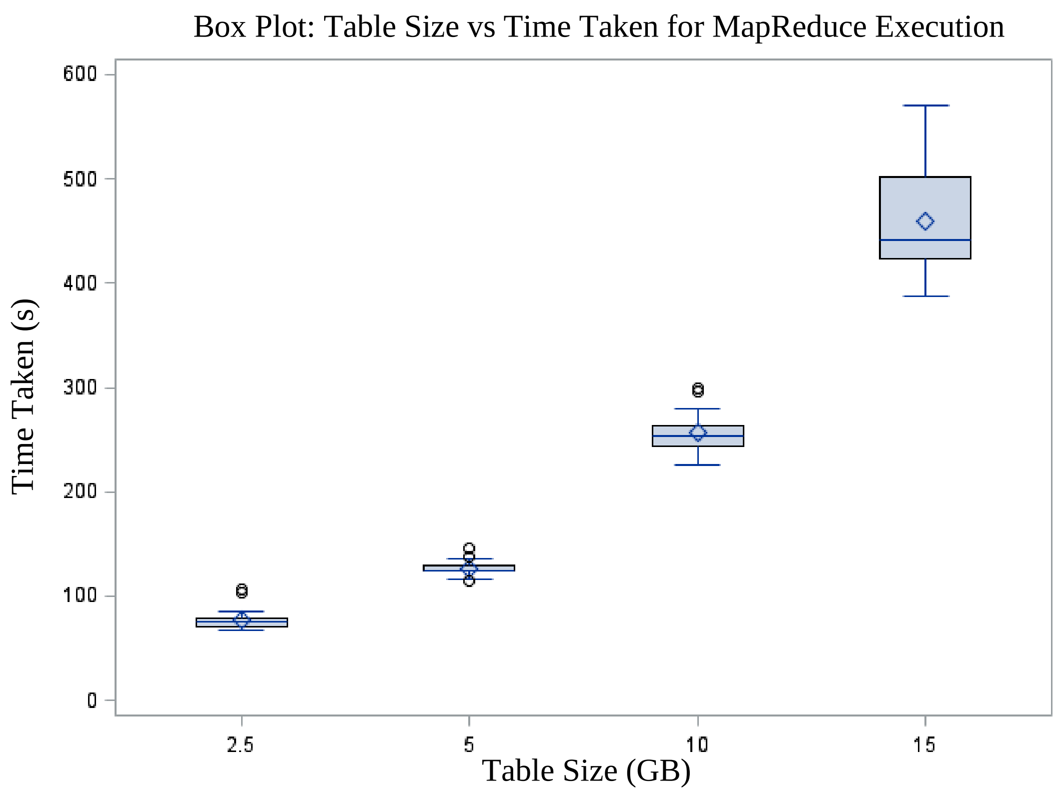

Figure 4.1 Execution time vs. table size for Experiment 1

As can be seen from Table 4.1 and Figure 4.1, there was a steady increase in MapReduce execution time as the table size was increased. This is apparent from the sheer amount of reads and writes that need to be in place. Each sample file contained a considerable amount of data, ranging from 40869957 lines in the 2.5 GB sized table to 120000000 lines in the 15 GB sized table, thus explaining the increase in execution times.

Another factor that needs to be noted is the range of data points in each group. As perceived from the boxplot in Figure 4.1, the range of results recorded increased as the file size was increased. This was as predicted, as large datasets provide more room for variance for any query, especially in a SELECT * SQL query. These results indicate that the new implementation of the correlation-aware SQL-to-MapReduce translator behaves as expected in a Hadoop 2.x.x environment.

4.1.2 Hypothesis Testing

To provide a more sound analysis, the author has elected to describe the findings in a linear regression model. Therefore, the following hypotheses, where β_1 is the slope of the model, were devised:

$$H_0: \beta_1 = 0$$

$$H\alpha: \beta_1 > 0$$

The data points were added and computed through SAS and the results for a regression analysis of variance is shown in Table 4.2:

Table 4.2 *Major hypothesis testing statistics for Experiment 1*

Number of Observations	110 (counts)
Model Mean Square	2378018 (seconds)
F Value	2126.41
P Value	< 0.0001

The P Value fell below the alpha-value of 0.05, thus allowed the rejection of the null hypothesis. It can be concluded that there was a linear relationship between file size and time taken for the map reduce execution. Furthermore, a positive linear line can be inferred from the boxplot in Figure 4.1. A more detailed analysis on the regression model is described in Section 4.1.3

4.1.3 Regression Analysis

Upon running statistical analysis for a regression model the following fit plot, as presented in Figure 4.2, was deduced.

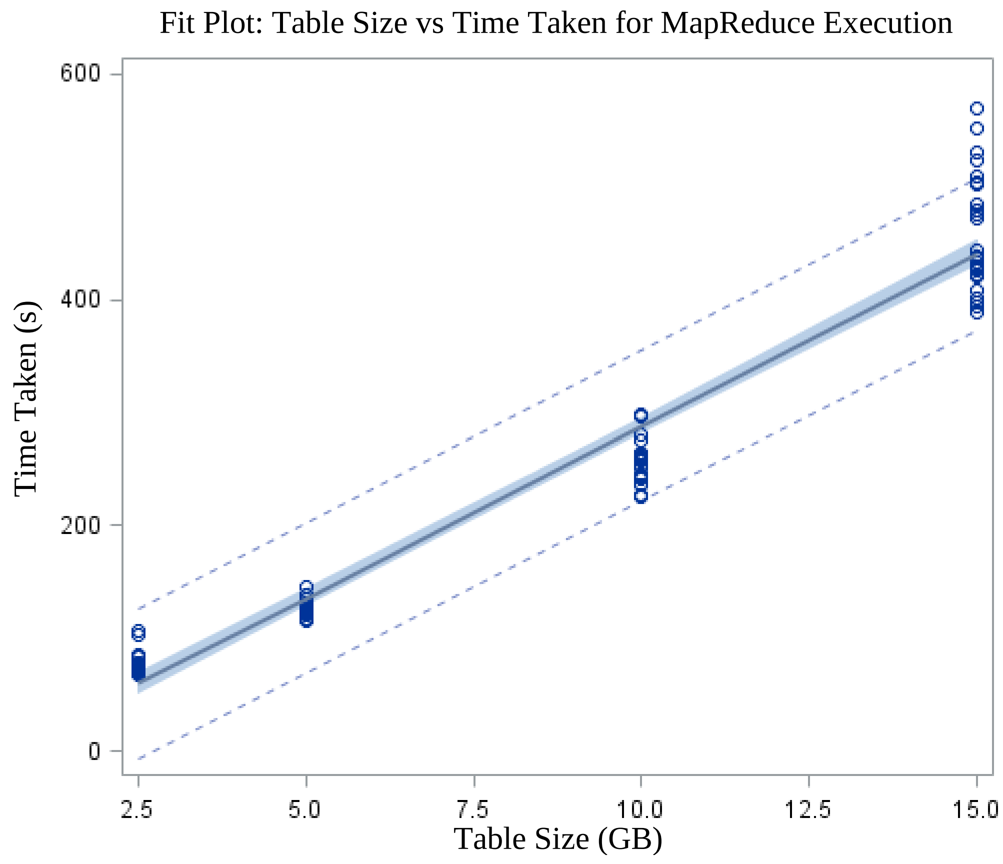

Figure 4.2 Table size vs. time taken fit plot for Experiment 1

The following are the major observations derived from Figure 4.2.

- The regression model has indicated that almost all data points fit comfortably within the 95% confidence interval range, with the exception of few observations on the 15 GB sample data group. A couple of readings can be attributed to this irregularity:

    - Hathi, as mentioned before, is a shared environment and may prone to some changes due to factors like peak usage, conflicting jobs and parallel processes. Because the translations on the 15 GB sample data were made across multiple times throughout the day by multiple persons, some discrepancies are observed.
    - Some runs failed due to the high volume of job requests. In some cases, jobs were killed by the Application Master due to the lack of space in the assigned disk. Many of such observations were removed as outliers, but the author has decided to keep some, due to the validity shown through the IQR outlier test.
    - The distributive nature of Hadoop may have initiate an implicit optimization, thus resulting on higher variance in the larger file sized table.
- A positive relationship can be found between the data size and time taken to execute the MapReduce job.
- The data points for runs on 2.5 GB sample data seemed to be slightly higher than expected, thus prompting the need for more investigation. During the inception of the sample data, small variances in actual file sizes were presumed. While the 5 GB, 10 GB and 15 GB sample data had differences of about 0.005 GB, the 2.5 GB sample file had the highest variance among all groups, which was 0.15GB higher than expected. This discrepancy can be attributed to this observation.

After considering these observations, the author has elected to process the model, as presented in Figure 4.2, as valid and derived the following linear equation:

$$\mu_{\text{time taken}} = -16.16612 + 30.45342x + \varepsilon_i$$

In words, this formula translates to a line with a y-intercept of -16.16612 and slope of 30.45342, where x is the table size. Furthermore, the statistical analysis indicated that this line has an R-Squared value of 0.9512, that is, 95.12% of the total variation in time taken is explained by this least squares regression model on file size. Additionally, R-Square also translates this model into a strong positive linear relationship between file size and execution time.

4.2 Experiment 2: Complex query translation on large data sets

Experiment 2 of this thesis involved the testing of the modified translator's ability to generate MapReduce codes for complex query translation on multiple degrees of sample size. For this purpose, Q6 (Query 6) from the TPC-H series were selected and multiple runs were made against 60 GB, 120 GB and 180 GB of data. All sample data was generated from the TPC-H database generator tool provided by the Transaction Performance Processing Council.

4.2.1 General Inferences

In each group, the translated MapReduce jobs for the standard Q6 were run 20 times respectively. After collecting the time taken for the jobs to completely execute, any outliers were removed by the IQR outlier test. The average of the results, ex post facto of the outlier removal is described in Table 4.3.

Table 4.3 *Average Time Taken for Experiment 2 Job Execution*

Sample Data Size (GB)	Average Time Taken (mm:ss)
60	00:49
120	02:40
180	03:55

As can be inferred from Table 4.3 above, the mean of the results collected showed a steady increase as the sample data size increases, almost proportionally. In order to understand the ranges within each group, a box plot was constructed as in Figure 4.3.

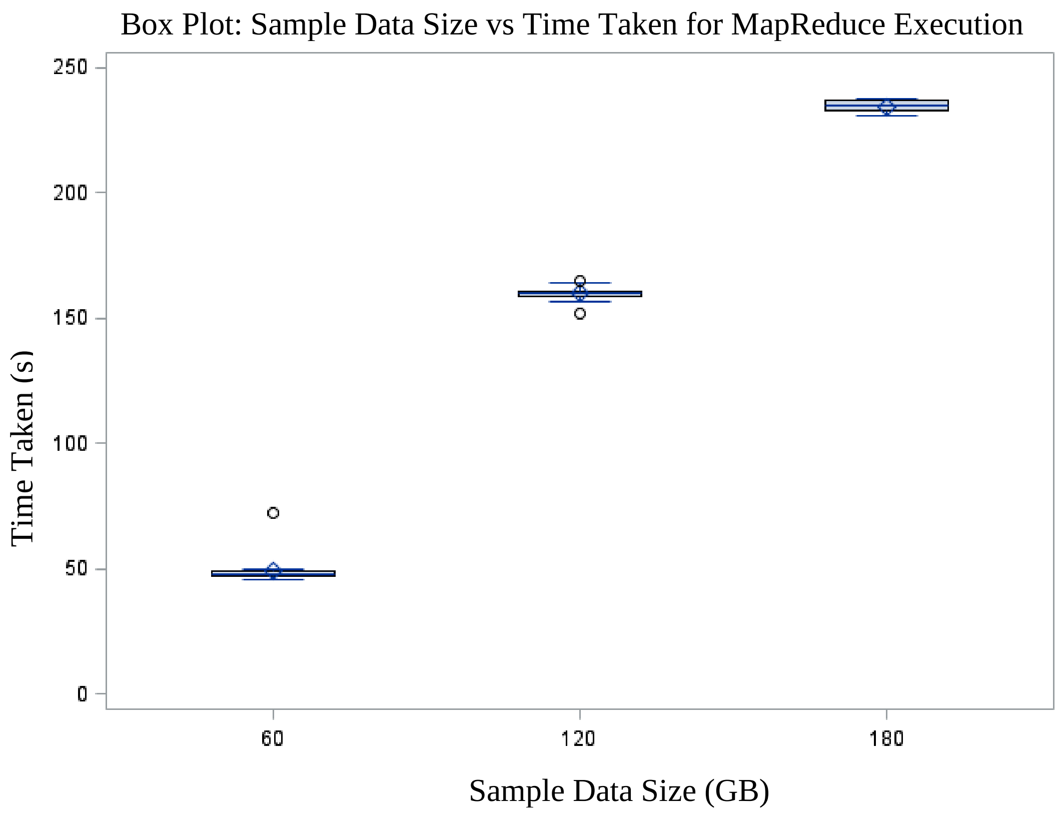

Figure 4.3 Execution time vs. sample data size for Experiment 2

The following observations can be made from the box plot above:

- There was a steady increase in the MapReduce execution time as the sample source data size increases.
- There was a strong positive linear relationship between sample data size and execution time.
- The ranges, both IQR and maximum/minimum range, were small across all groups. This shows that there were little variation present in the result collected, even with larger sample sizes.

4.2.2 Hypothesis Testing

Similar to the previous experiment, a hypothesis testing was set up to provide a statistical significance to any slope present in the linear relationship. For that, the following hypotheses, where β_1 is the slope of the regression model, were formulated:

$$H_0: \beta_1 = 0$$

$$H\alpha: \beta_1 > 0$$

The results were analyzed through SAS and the significant values are presented in Table 4.4:

Table 4.4 *Major hypothesis testing statistics for Experiment 2*

Number of Observations	57 (count)
Model Mean Square	335969
F Value	3907.72
P Value	< 0.0001

The P-Value was considerably less than the alpha value, thus allowing the rejection of the null hypothesis. There was a significant slope between execution time and sample data size. In addition to that, from observing the box plot in Figure 4.3 and trends in Table 4.3, it was established that the linear relationship is positive.

4.2.3 Regression Analysis

In order to further evaluate the results, a regression analysis was made through SAS and the following fit plot was constructed.

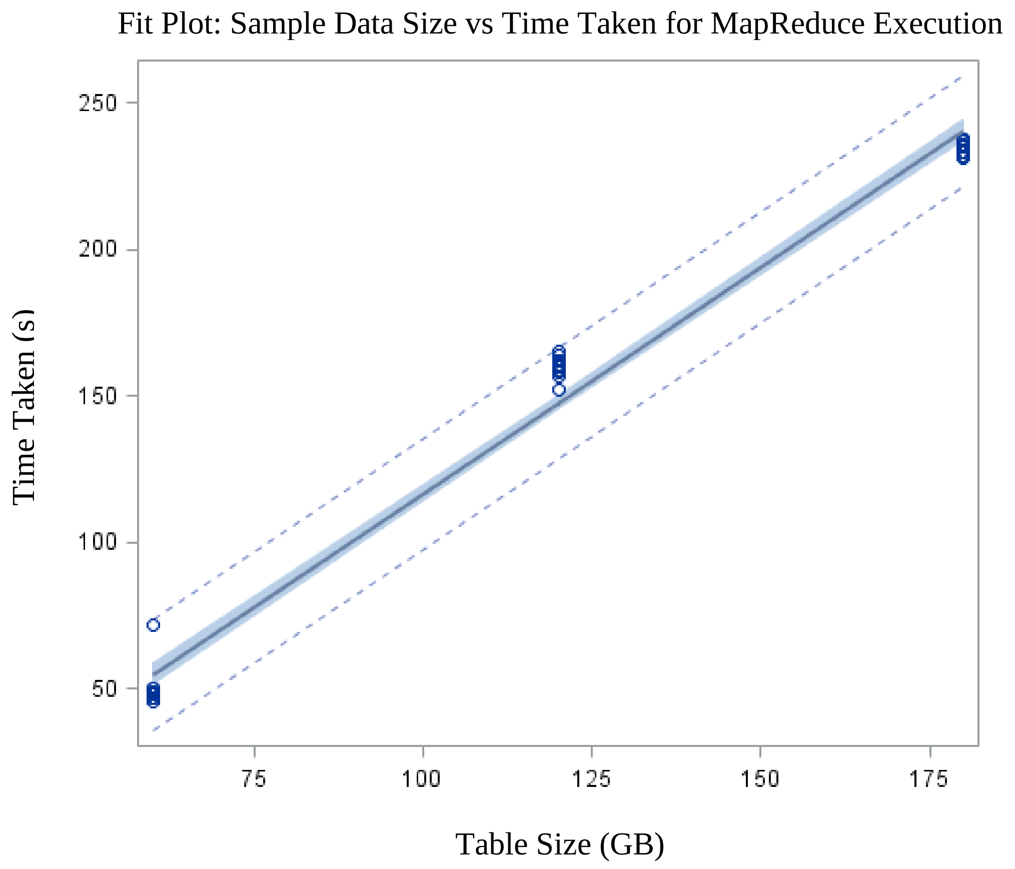

Figure 4.4 Table size vs. time taken fit plot for Experiment 2

The following observations were made based on Figure 4.4:

- All data points collected fits within the 95% interval range of the linear regression model. This was due to the low variance between the maximum and minimum points within each group.
- The means for 60 GB and 180 GB sample size groups are below the regression line, whereas the mean for 120 GB sample size lied above the model.

After considering the trends, fit plot and analysis of variance, the following equation was deemed to be the best predictor for the regression model:

$$\mu_{\text{time taken}} = -38.07786 + 1.54726x + \varepsilon_i$$

This equation translates to a line with a y-intercept of -38.07786 and, more importantly, a slope of 1.54726 seconds increase per gigabyte increase in the sample data. Furthermore, the analysis of variance concludes that this model has an adjusted R-squared model of 0.9859, meaning 98.59% of the total variation of the execution time is explained by the sample size used in the model.

4.3 Experiment 3: Multi-query Functionality

Another major aspect of this thesis project was to introduce multi-query functionality to the correlation-aware SQL-to-MapReduce translator. As such, the author has elected to translate variations of Q6 (Query 6), as presented in Appendix A, from the TPC-H series by manipulating the WHERE clause of the query. This decision was made for the following reasons:

- Query 6 contains necessary complexity yet can be accomplished through only one job. Queries like Q17, Q18 and Q21 generates jobs ranging from 7 to 10, thus increasing the execution time drastically.
- Multiple jobs, even ones that are generated as part of a single-query translation, take different execution time and add unwanted complexity to the result analysis.
- Using different queries may possibly create drastic variance in the time taken for the job execution.
- The goal of this experiment was to test the multi-query translation functionality rather than the diversity of the queries translated. It is assumed that the new implementation of the translator would work in similar fashion to other queries, provided that they meet the general requirement for YSMART.

4.3.1 General Inferences

Multiple versions of Q6 were devised, with consistency in mind, and translated into MapReduce jobs using the modified translator. Twenty (20) runs were made for single query translation, while 16 runs were made for 5, 10 and 15 queries translation respectively. Any outliers were removed before analyzing the data. Figure 4.5 below shows a box plot of the initial findings:

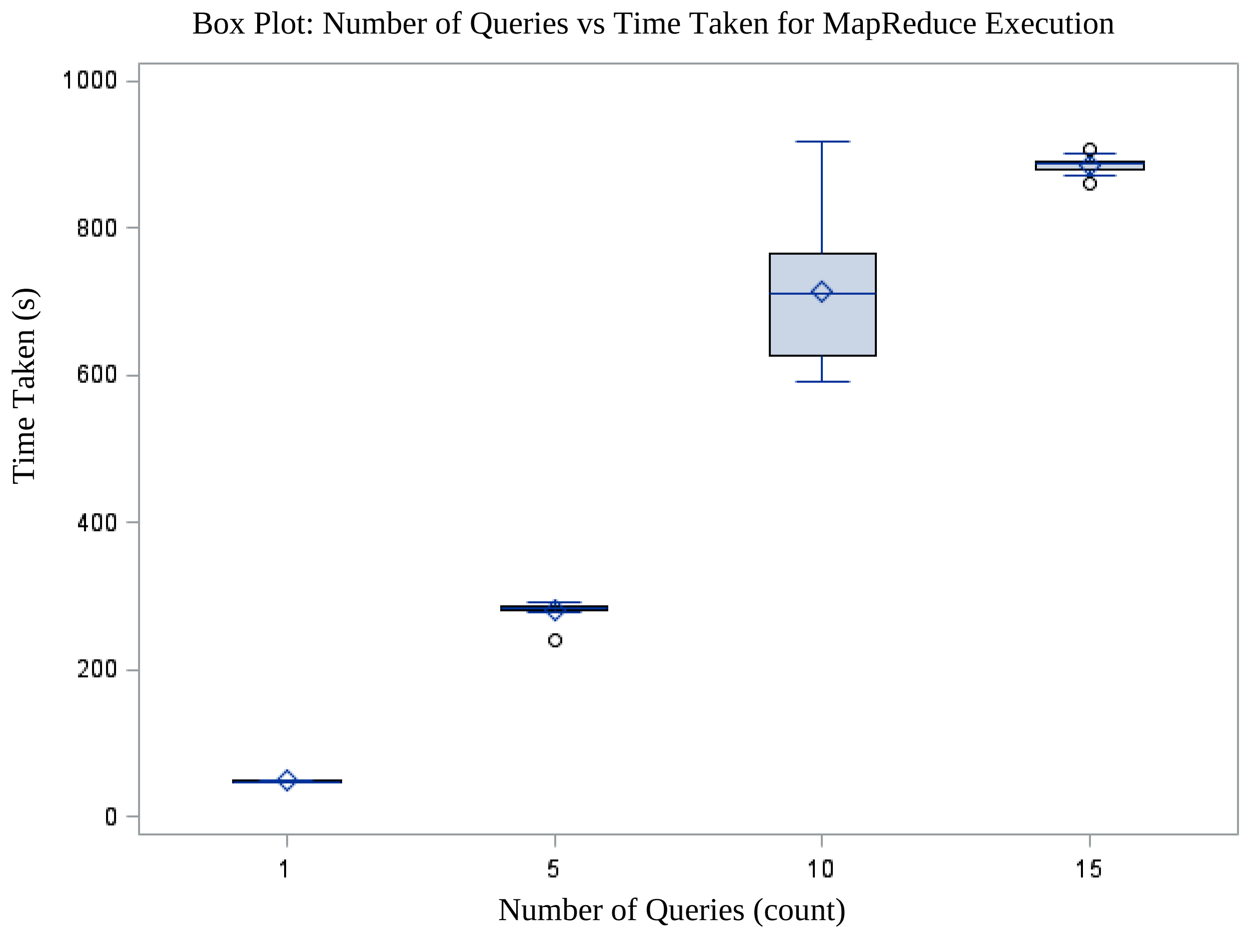

Figure 4.5 Execution time vs. number of queries for Experiment 3

As illustrated in Figure 4.5, the results for this test has returned a strong positive linear relationship. The total time taken for job(s) execution increased as the number of queries were increased, as expected.

One observation that needs to be noted was the high variance in the time taken for the 10 queries translation group. Upon investigation, the author has found that the data collection for that particular group was done by three researchers, almost at the same time. As mentioned before, as Hathi is a shared environment, this can be attributed to multiple concurrent job executions. A portion of the log illustrating this notion can be found in Appendix B.

All other groups, that is, 1 query, 5 queries and 15 queries have shown very little discrepancies within each respective group due to separate data collections.

4.3.2 Hypothesis Testing

The hypotheses for this test, where β_1 is the slope of the regression model, was as follow:

$$H_0: \beta_1 = 0$$

$$H\alpha: \beta_1 > 0$$

The data points were plotted in a fit plot through SAS. Table 4.5 describes major findings:

Table 4.5 *Major hypothesis testing statistics for Experiment 3*

Number of Observations	69 (count)
Model Mean Square	7476471 (seconds)
F Value	1358.20
P Value	< 0.0001

Due to the high F Value, which was computed from the model's mean square, a low P Value can be derived. This allowed the rejection of the null hypothesis; there was a significant slope in the linear regression model. In addition to that, a more detailed regression analysis was made and summarized in the following section.

4.3.3 Regression Analysis

A regression analysis was done through SAS for all 69 data points. As per earlier tests, any outliers were removed using the IQR outlier test. The following fit plot can be construed from such analysis:

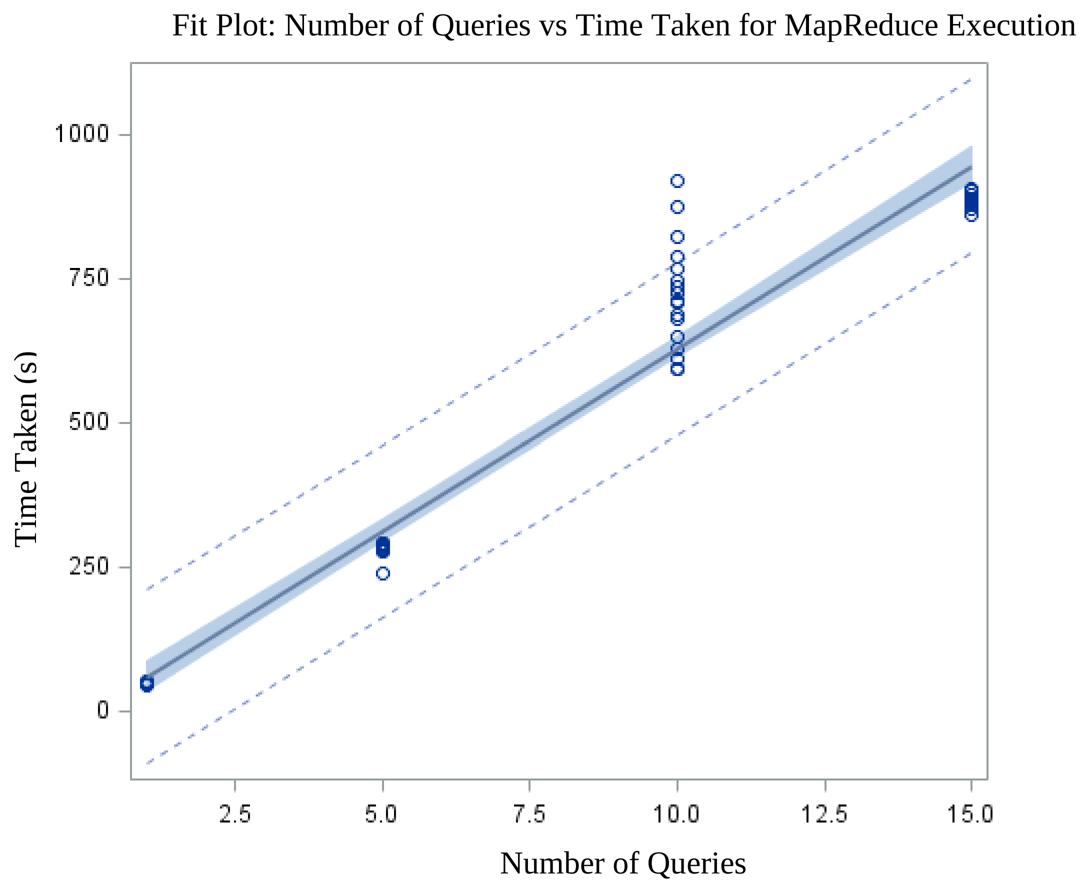

Figure 4.6 Number of queries vs. time taken fit plot for Experiment 3

The following observation can be seen from the fit plot in Figure 4.6:

- Data points showed a strong positive linear relationship between number of queries translated and time taken for the MapReduce code execution.

- Data points for 1 query, 5 queries and 15 queries fit comfortably in the 95% confidence interval range.
- Only 4 data points in the 10 queries translation group were recorded to be above the upper boundary of the interval.

After considering these observations, the following equation was formulated to be the best predictor for this linear model:

$$\mu_{\text{time taken}} = -3.81811 + 63.31410x + \varepsilon_i$$

This equation translates to a line with a slope of 63.31410 seconds per query increase. The y-intercept, on the other hand, is calculated to be -3.81811 seconds and can be, due to practicality, capped off at 0 second.

In addition to this, the statistical analysis also indicates that this regression model yielded an adjusted R-squared value of 0.9523, that is, 95.23% of the model explained all the variability of the recorded data around its mean of the model.

4.4 Chapter Summary

This chapter presents the textual, tabular and graphical representation of the results collected from various tests. It also accounts for any trends observed of the data points. In addition to that, this chapter terms the general inferences, hypothesis testing and statistical regression analyses made for each test.

CHAPTER 5. CONCLUSIONS, DISCUSSIONS AND FUTURE DIRECTIONS

This chapter summarizes the findings of this thesis project. It further describes the general discussions, and future recommendation on improving the modified correlation-aware SQL-to-MapReduce translator.

5.1 Conclusions

The author has implemented a modified version of an existing SQL-to-MapReduce model, as part of this research, to improve its multi-query functionality and compatibility with newer Hadoop versions. This study mainly focused on the modified translator's ability to process multiple queries and its compatibility with the latest Hadoop version since its conception, version 2.

In the first experiment, the author has tested the modified translator's compatibility with Hadoop version 2.2.0 through series of tests with varying degree of table sizes. Data was collected and analyzed for tables sized 2.5 GB, 5 GB, 10 GB and 15 GB. The results of this experiment has shown that there was a steady increase in the execution time as the table size was increased as well as a notable increase in the range of the data points within each group. In addition to that, a hypothesis for regression slope was tested and allowed the rejection of the null hypothesis.

Consequently, the modified implementation’s ability to translate a comparatively complex query on larger data sets was tested. For that, Q6 of the TPC-H series was translated and translated for 60GB, 120 GB and 180 GB of sample data. Similar to Experiment 1, the results have indicated that there was a steady linear relationship between execution time and sample data size. In addition to that, the null hypothesis for the slope hypotheses test was rejected due to the small P Value returned.

Finally, the multi query functionality was tested with varying versions of Q6 of the TPC-H benchmarked series. The total execution time was collected and outliers were removed through the standard IQR outlier test. Analyzing the results also indicated that the total execution time followed a predictive linear relationship with the number of queries. Any irregular observations were noted and explained.

An analysis of variance for regression model was accompanied for each of the aforementioned experiments. Table 5.1 shows the summary of the findings.

Table 5.1 *Regression models and respective r-squared values for experiments 1, 2 and 3.*

Experiment	Dependent Variable	Regression Model	R-Squared
1	Table size	$\mu_{\text{time taken}} = -16.16612 + 30.45342x + \varepsilon_i$	0.9512
2	Sample data size	$\mu_{\text{time taken}} = -38.07786 + 1.54726x + \varepsilon_i$	0.9859
3	Number of Queries	$\mu_{\text{time taken}} = -3.81811 + 63.31410x + \varepsilon_i$	0.9523

As presented in Table 4.3, all experiments yielded a strong positive linear relationship between respective dependent variable and execution time, with high R-squared value. This indicates that these models explain a large portions of variations in each experiments. Furthermore, this also solidifies these formulae’s validity to predict the execution time of the modified translation model for other scenarios.

5.2 Discussions

This research was conducted to be a small-scale sampling of introducing multi query functionality to an existing correlation-aware SQL-to-MapReduce translator. During the process, an upgrade was made to the translator to operate in a Hadoop version 2, particularly in Hadoop 2.2.0, environment.

In addition to that, all queries translated in this thesis project were from the TPC-H benchmarked series. Selected queries were specific to the modified translator’s need as well as to the experiment requirements. Although the author has conducted experiments that best describe the modified translator’s behavior under different circumstances, any changes to the internal and external environment, including the usage of other SQL queries, may potentially yield different observations.

Furthermore, the author has avoided the usage of complex queries like Q18 and Q21 of the TPC-H benchmarked series due to the total number of MapReduce jobs generated as part of their translations. A good substitute, that is Q6, was elected for both Experiment 1 and Experiment 2.

The initial plan of this thesis was to, additionally, study the behavior of the modified translator on varying degree of nodes. This was meant to test the implementation’s horizontal scalability but, due to the time and infrastructure limitations, the study of this aforementioned factor could not be included in this research.

5.3 Future Directions

The modified translator does provide an advantage when it comes to translating multiple queries at the same time. Compared to Hive and Pig, this modified implementation of YSMART allows users to translate SQL queries and execute MapReduce jobs that are generated as part of their translation. In this work, explicit optimization of the multi-job processing was not a focus, and could potentially be an improvement in future works.

In addition to that, the author has elected to translate only queries from the TPC-H benchmarked series. While these queries allow for a comprehensive testing mechanism, it does not cover the entirety of SQL syntaxes. Similar to Hive and, its query language, HiveQL, the translation tool only supports a subset of the SQL syntax. Future works can be made on expanding the syntax for more unique SQL functions and clauses, thus forming a more complete tool.

This works has also limited the usage of queries from the TPC-H benchmarked series. Complex queries such as Q17, Q18 and Q21 were not used due to time and infrastructure limitations. The author acknowledges that YSMART, the underlying correlation-aware SQL-to-MapReduce translator, was initially tested using these queries and future efforts can be done on understanding the modified translator's behavior in processing these above-mentioned queries.

A reduced scope of this project was the testing of the translator's horizontal scalability by varying the number of nodes in the test environment. Due to the infrastructure constraints, this could not be done thus, opening another opportunity for future works. Furthermore, performance comparisons with similar, yet prominent solutions such as Hive and Pig could potentially further promote the translator’s advantages.

All sample data in this research was generated through the Transactional Performance Processing Council’s database generator tool. The data generated in a random fashion and has no meaning to it. Testing the tool with a more insightful data might pave way for new discoveries in the future.

5.4 Chapter Summary

This chapter summarized the major findings of this thesis project as per the research question presented in Chapter 1. Subsequently, a discussion on the major observations and potential fallbacks were discussed in Section 5.2. Finally, the author has identified a number of prospective future works that might bring improvements on to this research.

LIST OF REFERENCES

LIST OF REFERENCES

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

Deshpande, N. N. (2011). Improving the performance of the spectral deconvolution stage of the proteomic discovery process (Order No. 1501874). Available from Dissertations & Theses @ CIC Institutions; ProQuest Dissertations & Theses A&I. (905561906). Retrieved from http://search.proquest.com/docview/905561906?accountid=13360

Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., & Srivastava, U. (2009). Building a high-level dataflow system on top of Map-Reduce: The Pig experience. *Proceedings of the VLDB Endowment*,*2*(2), 1414-1425.

Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C., & Huang, Y. (2014). SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters. *Journal of Parallel and Distributed Computing*,*74*(3), 2166-2179.

Jihong, Z. Q. L. J. C. (2004). Heterogeneous data exchange based on XML and its implementation in Java. *Computer Applications and Software*, *11*, 022.

Lee, R., Luo, T., Huai, Y., Wang, F., He, Y., & Zhang, X. (2011, June). YSMART: Yet another SQL-to-MapReduce translator. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on* (pp. 25-36). IEEE.

Lin, L., Lychagina, V., Liu, W., Kwon, Y., Mittal, S., & Wong, M. (2011). Tenzing a SQL implementation on the MapReduce framework.

Lin, X., Ye, Y., & Ma, S. (2013, October). MRPacker: An SQL to MapReduce optimizer. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management* (pp. 1157-1160). ACM.

Nykiel, T., Potamias, M., Mishra, C., Kollios, G., & Koudas, N. (2010). MRShare: Sharing across multiple queries in MapReduce. *Proceedings of the VLDB Endowment*, *3*(1-2), 494-505.

Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008, June). Pig Latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1099-1110). ACM.

Press, G. (2013, December 21). A very short history of Big Data. Retrieved April 1, 2015, from http://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/

Su, X., & Swart, G. (2012, May). Oracle in-database Hadoop: When MapReduce meets RDBMS. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 779-790). ACM.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., & Murthy, R. (2009). Hive: A warehousing solution over a Map-Reduce framework. *Proceedings of the VLDB Endowment*, *2*(2), 1626-1629.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., & Murthy, R. (2010, March). Hive-a petabyte scale data warehouse using Hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (pp. 996-1005). IEEE.

Wang, G., & Chan, C. Y. (2013). Multi-query optimization in MapReduce framework. *Proceedings of the VLDB Endowment*, *7*(3).

Zhang, H., Wang, Y., & Han, J. (2011, December). Middleware design for integrating relational database and NOSQL based on data dictionary. In *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on* (pp. 1469-1472). IEEE.

APPENDICES

Appendix A Sample Queries

SELECT Query (Used for Experiment 1)

```
select
     *
from
     lineitem;
```

QUERY 1 (Used in Experiment 1 and 2)

```
select
     l_returnflag,
     l_linestatus,
     sum(l_quantity) as sum_qty,
     sum(l_extendedprice) as sum_base_price,
     sum(l_extendedprice * (1 - l_discount)) as
sum_disc_price,
     sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
as sum_charge,
     avg(l_quantity) as avg_qty,
     avg(l_extendedprice) as avg_price,
     avg(l_discount) as avg_disc,
     count(*) as count_order
from
     lineitem
where
     l_shipdate <= '1998-09-04'
group by
     l_returnflag,
     l_linestatus
order by
     l_returnflag,
     l_linestatus;
```

QUERY 6 (Used in Experiment 2 and 3)

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1995-01-01'
        and l_shipdate < '1996-01-01'
        and l_discount > 0.07
        and l_discount < 0.09
        and l_quantity < 24;
```

Variations of QUERY 6 (Used in Experiment 3)

Variation 1:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1995-01-01'
        and l_shipdate < '1996-01-01'
        and l_discount > 0.07
        and l_discount < 0.09
        and l_quantity < 24;
```

Variation 2:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 3:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1997-01-01'
        and l_shipdate < '1998-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 4:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.05
        and l_discount < 0.15
        and l_quantity < 100;
```

Variation 5:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 6:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1997-01-01'
        and l_shipdate < '1998-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 7:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1992-01-01'
        and l_shipdate < '1994-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 8:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1993-01-01'
        and l_shipdate < '1999-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 120;
```

Variation 9:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.10
        and l_discount < 0.15
        and l_quantity < 50;
```

Variation 10:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1990-01-01'
        and l_shipdate < '1993-01-01'
        and l_discount > 0.10
        and l_discount < 0.15
        and l_quantity < 150;
```

Variation 11:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1995-01-01'
        and l_shipdate < '1996-01-01'
        and l_discount > 0.07
        and l_discount < 0.09
        and l_quantity < 24;
```

Variation 12:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 13:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1997-01-01'
        and l_shipdate < '1998-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 50;
```

Variation 14:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.05
        and l_discount < 0.15
        and l_quantity < 100;
```

Variation 15:

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1996-01-01'
        and l_shipdate < '1997-01-01'
        and l_discount > 0.07
        and l_discount < 0.10
        and l_quantity < 5
```

Appendix B Partial Log for Experiment 3 (Translation of 10 Queries)

Table B.1 *Partial Log for Experiment 3*

Run Number	Start Job Number	Start Time	End Time
1	job_1422989359880_0918	15/03/15 23:50:11	15/03/16 00:01:40
2	job_1422989359880_0918	15/03/15 23:50:11	15/03/16 00:01:40
3	job_1422989359880_0936	15/03/16 00:13:32	15/03/16 00:27:19
4	job_1422989359880_0954	15/03/16 00:34:53	15/03/16 00:48:18
5	job_1422989359880_0976	15/03/16 01:02:27	15/03/16 01:15:35
6	job_1422989359880_0994	15/03/16 01:37:15	15/03/16 01:51:31
7	job_1422989359880_1019	15/03/16 02:09:45	15/03/16 02:23:26
8	job_1422989359880_1045	15/03/16 02:42:24	15/03/16 02:56:16
9	job_1422989359880_1066	15/03/16 03:07:28	15/03/16 03:24:11
10	job_1422989359880_0918	15/03/15 23:49:52	15/03/16 00:00:12
11	job_1422989359880_0928	15/03/16 00:04:40	15/03/16 00:41:57
12	job_1422989359880_0943	15/03/16 00:20:40	15/03/16 00:45:47
13	job_1422989359880_0968	15/03/16 00:50:53	15/03/16 01:06:44
14	job_1422989359880_0988	15/03/16 01:31:18	15/03/16 01:50:12
15	job_1422989359880_1008	15/03/16 01:50:55	15/03/16 02:04:04
16	job_1422989359880_1022	15/03/16 02:12:43	15/03/16 02:28:19
17	job_1422989359880_1038	15/03/16 02:32:22	15/03/16 02:47:53
18	job_1422989359880_1065	15/03/16 03:07:17	15/03/16 03:27:38
19	job_1422989359880_1085	15/03/16 03:39:54	15/03/16 04:07:28