Hadoop Performance Modeling and Job Optimization for Big Data Analytics

A Thesis submitted for the Degree of
Doctor of Philosophy

By

Mukhtaj Khan

Department of Electronic and Computer Engineering
College of Engineering, Design and Physical Sciences
Brunel University London, UK

March 2015

Abstract

Big data has received a momentum from both academia and industry. The MapReduce model has emerged into a major computing model in support of big data analytics. Hadoop, which is an open source implementation of the MapReduce model, has been widely taken up by the community. Cloud service providers such as Amazon EC2 cloud have now supported Hadoop user applications. However, a key challenge is that the cloud service providers do not a have resource provisioning mechanism to satisfy user jobs with deadline requirements. Currently, it is solely the user responsibility to estimate the require amount of resources for their job running in a public cloud. This thesis presents a Hadoop performance model that accurately estimates the execution duration of a job and further provisions the required amount of resources for a job to be completed within a deadline. The proposed model employs Locally Weighted Linear Regression (LWLR) model to estimate execution time of a job and Lagrange Multiplier technique for resource provisioning to satisfy user job with a given deadline. The performance of the propose model is extensively evaluated in both in-house Hadoop cluster and Amazon EC2 Cloud. Experimental results show that the proposed model is highly accurate in job execution estimation and jobs are completed within the required deadlines following on the resource provisioning scheme of the proposed model.

In addition, the Hadoop framework has over 190 configuration parameters and some of them have significant effects on the performance of a Hadoop job. Manually setting the optimum values for these parameters is a challenging task and also a time consuming process. This thesis presents optimization works that enhances the performance of Hadoop by automatically tuning its parameter values. It employs Gene Expression Programming (GEP) technique to build an objective function that represents the performance of a job and the correlation among the configuration parameters. For the purpose of optimization, Particle Swarm Optimization (PSO) is employed to find automatically an optimal or a near optimal configuration settings. The performance of the proposed work is intensively evaluated on a Hadoop cluster and the experimental results show that the proposed work enhances the performance of Hadoop significantly compared with the default settings.

Acknowledgements

Firstly, I would like to express my gratitude to Prof. Maozhen Li who supervised my research and guided me with great dedication throughout the years. His view of research has proved precious to my research and supporting work. His personality, together with his high standards for research, makes him an ideal advisor. I would also like to thank to Prof. Gareth A. Taylor for his valuable help and guidance during my research.

I would like to thank my parents and all my family members. They helped me to pass through the toughest time of my life. Without their continuous encouragement and help, I will never reach the place where I am stand.

I am appreciative to the Abdul Wali Khan University Mardan, Pakistan for supporting my research.

In addition, I would like to thank Mr. Izaz, Mr. Suleman, Mr. Jahanzeb and Mr. Bilal for their sincere support and take care of me in my study period.

Declaration of Authorship

The work detailed in this thesis has not been previously submitted for a degree in this University or at any other and unless otherwise referenced it is the authors own work.

List of Publications

The following papers have been accepted / to be submitted for publication as a direct or indirect result of the research discussed in this thesis.

Journal Papers:

Yang Liu, Maozhen Li, **Mukhtaj Khan** and Man Qi , "A MapReduce based Distributed LSI for Scalable Information Retrieval ", Journal of *Computing and Informatics*, vol. 33, no. 2, pp. 2014.

M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, "Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data," IEEE Transactions on *Smart Grid*, vol. 6, no. 1, pp. 360–368, Jan. 2015.

M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop Performance Modeling for Job Estimation and Resource Provisioning," *Parallel Distrib. Syst. IEEE Trans.*, vol. PP, no. 99, p. 1, 2015 (in press, DOI: 10.1109/TPDS.2015.2405552).

M.Khan, Z.Huang, M.Li and G.A. Taylor, "Optimizing Hadoop Parameter Settings with Gene Expression Programming Guided PSO" submitted to *Parallel Distrib. Syst. IEEE Trans.*

Conference Papers:

M. Khan, M. Li, P. Ashton, G. Taylor, and J. Liu, "Big data analytics on PMU measurements," in Proceedings of the *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD),* , 2014, pp. 715–719.

M. Khan, Y. Liu and M. Li, "Data locality in Hadoop cluster systems," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on*, 2014, pp. 720-724.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are living in the era of Big Data. Today a vast amount of data is generating everywhere due to advances in the Internet and communication technologies and the interests of people using smartphones, social media, Internet of Things, sensor devices, online services and many more. Similarly, in improvements in data applications and wide distribution of software, several government and commercial organizations such as financial institutions, healthcare organization, education and research department, energy sectors, retail sectors, life sciences and environmental departments are all producing a large amount of data every day. For examples, International Data Corporation (IDC) reported that 2.8 ZB (zettabytes) data of universe were stored in the year of 2012 and this will reach up to 40 ZB by 2020 [1]. Similarly Facebook processes around 500 TB (terabytes) data per day [2] and Twitter generates 8 TB data every day [3]. The huge datasets not only include structured form of data but more than 75% of the dataset includes raw, semi-structured and unstructured form of data [4]. This massive amount of data with different formats can be considered as *Big Data*.

The derivation of *Big Data* is vague and there are a lot of definitions on *Big Data*. For examples, Matt Aslett defined *Big Data* as "*Big Data is now almost universally understood to refer to the realization of greater business intelligence by storing, processing, and analyzing data that was previously ignored due to limitation of traditional data management technologies*" [5]. Recently, the term of *Big Data* has received a remarkable momentum from governments, industry and research communities. In [6], *Big Data* is defined as a term that encompasses the use of techniques to capture, process, analyse and visualize potentially large datasets in a reasonable timeframe not accessible to standard IT technologies. The term *Big Data* is basically characterized with 3 Vs [4]:

- Volume, the sheer amount of data generated (i.e. from terabytes to zettabytes),
- Velocity, the rate the data is being generated (i.e. from batch data to streaming data), and
- Variety, the heterogeneity of data sources (i.e. from structured data to unstructured data).

There are several factors that are involved in producing *Big Data*. One factor is the Internet and communication technology as it has been advanced to enable people and devices to be increasingly interconnected not only some time but all the time. Small integrated circuits are now so economical that people are using in almost every object to make them intelligent which is another reason of generating of mountains of data. The continuous reduction in the prices of storage devices is also a factor for Big Data.

Many organizations have realized the real-value benefits of Big Data and today they do have access to Big Data but they are facing significant challenges in processing and analyzing the wealth amount of data timely and effectively. More importantly, how to extract important information and knowledge from *Big Data* due to the sheer volume of the data in different forms (i.e. structured, semi-structured and unstructured) is a extremely challenging task. For many decades, the organizations have successfully applied relational database management systems (DBMS) for data storage and analysis. However, managing *Big Data* with its associative characteristics such as volume, velocity and variety is a challenging task for traditional DBMS because DBMS are hard to scale with ever increasing data and only support structured data format. However, opportunity is available with the right technology platform, to store and analyze the Big Data timely and effectively. The recent studies show that the right technology platform could be the use of a massive parallel and distributed computing platform. This platform can be found by implementing Hadoop MapReduce framework on cloud computing environment.

Cloud computing is a concept that involves sharing of computer resources over the Internet among multiple users in order to maximized effectiveness and utilization of the resources. The resources are not only shared among the users but it can be dynamically allocated and de-allocated on a demand basis. Normally, the users acquire the virtual computation resources from cloud service providers following a pay-as-you-go policy and run their applications on the allocated resources [7]–[9]. The MapReduce computing model has become a representative enabling technology in support of data intensive Cloud computing applications [10].

The Hadoop framework [11] is an open-source implementation of the MapReduce paradigm that is originally proposed by Google [12]. It offers an effective distributed computing environment that is capable of storing and processing a huge amount of unstructured data. Hadoop has received a wide acceptance from the community due to its opens source nature and extensively used for data intensive applications [13]–[19]. It offers remarkable features such as scalability, fault-tolerance and automatic code parallelization using commodity computers. Furthermore, cloud service providers such as Amazon has designed Elastic MapReduce (EMR) that enables users to execute their Hadoop applications across its Elastic Cloud Computing (EC2) nodes [20].

1.1 Motivations

There were three major motivations that drove the PhD research.

- Sustainability in power systems is so vital that an enormous effort must be made to avert power system breakdown scenarios. The blackout in North East America (August, 14 2003) and previous critical events all over the world are driving the industry to develop more automatic, adaptive and efficient computational tools for power system stability and monitoring analysis. It is becoming highly impossible for traditional supervisory control and data acquisition (SCADA) systems to predict or avert eventualities in a timely manner which may lead to power system catastrophes [21], [22]. One solution to these challenges is the development of the Wide Area Monitoring System (WAMS). A WAMS consists of a network of synchronized Phasor Measurement Units (PMUs) [21], [23] which provide a high sampling rate up to 60 samples per second that can be used to enhance the reliability, stability and security of the power systems. For this reason the PMUs are being rapidly deployed in the power systems globally. Tang et al. [24] pointed out that hundreds of PMUs have been deployed in the U.S. power grid and worldwide in the past few years. The large scale and rapid deployment of PMUs in power grids has led to Big Data issues. A PMU sampling at 60 samples per second generates about 300MB per day. A reasonable size of a power grid network with a few hundred PMUs would generate a big data at TB scale per day. The UK National Grid expects the measurements of the PMUs to be stored for a minimum period of one year which will pose a huge challenge for computation, analysis and storage. In addition, the power system community is expecting a scalable, resilient and fault-tolerant computing platform that can effectively store and timely process massive volumes of PMU data.

- As the Hadoop framework supports public Cloud computing such as Amazon EC2. This feature enables the organization to utilize the Cloud services as a pay-as-you-go manner. To use the EC2 Cloud, users have to configure the required amount of resources (virtual nodes) for their applications. However, the EC2 Cloud in its current form does not support Hadoop jobs with deadline requirements. It is purely the user's responsibility to estimate their job execution time and the amount of require resources to complete their jobs within deadline. Hence, Hadoop performance modeling has become a necessity in estimating the job completion time and provisioning the right amount of resources for a user jobs with deadline requirements.
- Hadoop MapReduce has become the most widely adopted computing framework for big data analytics. However, the performance of a Hadoop job is highly affected by configuration parameter settings. The recent research shows that the configuration parameter settings play a key role in the performance of a Hadoop, i.e. a small change on one of the parameter settings can have a huge impact on the performance of a Hadoop job. Hadoop has more than 190 configuration parameters that manage the execution flow of a Hadoop job. Most of Hadoop users even do not know about these configuration parameters and if they do not

supply any values to the configuration parameters, the system will automatically assign default values. However, on default configuration settings a Hadoop job does not effectively utilize the underlying resources and as a result Hadoop might not produce an optimal or a near optimal performance. Furthermore, it is highly challenging to find a mathematical model or a fitness function that can correlate the inter-dependencies among the Hadoop parameters. Moreover, the large set of parameters and the complex inter-connections among the configuration parameters further increase the complexity of manually tuning these parameter settings. Hence, an efficient, effective and automatic approach to parameter tuning is highly needed.

1.2 Methodology

This research first evaluates the performance of Hadoop in parallelization of detrended fluctuation analysis for fast event detection on massive PMU data [28]. It investigates in-depth the execution principles of Hadoop and mathematically models the three core execution phases of Hadoop (the map phase, the reduce phase and the shuffle phase) and based on that it applies locally weighted linear regression to estimate the execution time of a Hadoop job [29]. It employs Lagrange Multipliers technique for resource provisioning to satisfy jobs with deadline requirements. Finally, it employs particle swarm optimization technique to enhance the performance of Hadoop by automatically optimizing its parameter settings.

An experimental Hadoop cluster with two Intel servers was set up to evaluate the proposed works presented in this thesis. The specifications and configurations of the two server machines are presented in Chapter 4. The Oracle Virtual Box was installed on the two server machines and 8 Virtual Machines (VMs) were configured on each server machine. Each VM was assigned with 4 CPU cores, 8GB RAM and 150GB hard disk storage. The Ubuntu 12.04 TLS operating system was installed on every VM. The Hadoop-1.2.1 version was configured on VMs and Starfish [30] software was used to collect jobs profile information whenever is required.

To further evaluate the works presented in this study, another Hadoop cluster was setup on Amazon EC2 Cloud using 20 *m1.large* instances. Each instance was configured with 2vCPUs, 420GB hard disk and 7.5GB physical memory. The same Hadoop version, operating system and Starfish were installed on each instance.

1.3 Major Contributions to Knowledge

The major contributions of the thesis can be summarized as follows:

- The thesis presents a scalable, resilient and fault-tolerant computing framework for massive PMU data storage and analysis. The proposed framework is based on the Hadoop framework and the open-source OpenPDC software [27]. A small Java based application is developed to automatically stream the PMU data into the Hadoop Distributed File System (HDFS). An event detection algorithm based on Detrended Fluctuation Analysis (DFA) is parallelized in the Hadoop MapReduce cluster to process large volumes of PMU data. The parallel DFA algorithm is evaluated from the aspect of speedup, scalability and accuracy in comparison with standard DFA. The speedup of parallel DFA in computation is initially analysed through Amdahl's Law, a revision to the law is then proposed, suggesting enhancement to its capability to analyse the performance gain in computation when parallelizing data intensive application in a cluster computing environment.
- It presents an improved HP model for Hadoop job execution estimation and resource provisioning. The improved HP model mathematically models all the three core phases of a Hadoop job including the overlapping and non-overlapping stages of the job. Furthermore, the model employs Locally Weighted Linear Regression (LWLR) technique to estimate the execution time of a Hadoop job with a varied number of reduce tasks. Based on job execution estimation, the improved HP model employs Lagrange Multipliers technique to provision the amount of resources for a Hadoop job to complete within a given

deadline. The performance of the purpose model is extensively evaluated on both the in-house Hadoop cluster and the Amazon EC2 cloud.

- The thesis optimizes the performance of Hadoop by automatically tuning its parameter configuration settings. This work employs Gene Expression Programming (GEP) to build an objective function based on a training dataset. The objective function represents the correlations and inter-dependencies of the parameters. The fitness function is used to work as an objective function for Hadoop performance optimization. For this purpose of optimization Particle Swarm Optimization technique is employed to search for a set of optimum values of the configuration parameters within a search space. Unlike other works which divide the search space into sub-spaces, the proposed work considers the entire search space in the optimization process in order to maintain the inter-dependencies among the configuration parameters.
- The proposed works presented in this thesis have been intensively evaluated on both an in-house Hadoop cluster and the Amazon EC2 Cloud. Evaluation results are presented and analyzed in depth.

1.4 Thesis Organization

The remainder of the thesis is organized as follows:

Chapter 2 provides a general background on both the MapReduce programming model and the Hadoop MapReduce computing framework. It also introduces the Hadoop framework optimization approaches from the aspects of job scheduling, data locality and configuration parameter settings.

Chapter 3 introduces the design and implementation of a Parallel Detrended Fluctuation Analysis (PDFA) algorithm. The performance of the PDFA is evaluated from the aspects of speedup, accuracy and salability in comparison with the sequential DFA approach. The speedup of the PDFA is analyzed following Amdahl’s Law. This Chapter also presents a revision to the Amdahl’s Law to enhance its capability in analyzing the performance gain in computation when computation is parallelized in cluster environments.

Chapter 4 presents the design, implementation and evaluation of a Hadoop job performance model that accurately estimates a job execution time and further provisions the require amount of resource for a job to be completed within a deadline. The proposed model mathematically models three core phases of a job execution and employs Locally Weighted Linear Regression (LWLR) model to estimates the job completion time. Furthermore, the proposed model employs Lagrange Multipliers for the resource provisions to satisfy jobs with deadline requirements.

Chapter 5 first presents empirical evidence that configuration parameters do have significant effects on the performance of Hadoop. It then presents the design and implementation of a Hadoop job optimization model that improve the performance of a Hadoop by automatically tuning the configuration parameter settings. The model employs Gene Expression Programming (GEP) technique to develop a fitness function based on historical job profile information. Particle Swarm Optimization is employed to search for the optimal or near optimal values for these configuration parameters. The performance of the proposed work is compared with the default settings, Rule-of-Thumb settings and Starfish recommendations for Hadoop jobs.

Chapter 6 concludes the thesis and discusses some limitations of the research. In addition, a number of future works are pointed out for further improvements and extensions.

References

[1] “How much data is out there,” *webopedia.com*. [Online]. Available: http://www.webopedia.com/quick_ref/just-how-much-data-is-out-there.html. [Accessed: 26-Feb-2015].

[2] “Facebook data grows by over 500 TB daily,” *slashgear.com*. [Online]. Available: http://www.slashgear.com/facebook-data-grows-by-over-500-tb-daily-23243691/. [Accessed: 26-Feb-2015].

[3] "Twitter by the Numbers," *slideshare.net*. [Online]. Available: http://www.slideshare.net/raffikrikorian/twitter-by-the-numbers. [Accessed: 26-Feb-2015].

[4] P. zikopoulos, C., C. Eaton, D. DeRoos, T. Deutsch, and G. Lapis, *Understanding Big Data-Analytics for Enterprise Class Hadoop and Streaming Data*. New York, Chicago,San Francisco, USA: The McGraw-Hill Companies, 2012.

[5] T. Lubos, "Big Data Hadoop NoSQL DB -Introduction," *academia.edu*, 2013. [Online]. Available: https://www.academia.edu/5107042/Big_Data_Hadoop_NoSQL_DB_-_Introduction. [Accessed: 26-Feb-2015].

[6] "Big data, a new world of opportunities," *Networked European Software and Services Initiative (NESSI)*, 2012. [Online]. Available: http://www.nessi-europe.com/Files/Private/NESSI_WhitePaper_BigData.pdf. [Accessed: 25-Feb-2015].

[7] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing," in *Cloud Computing*, N. Antonopoulos and L. Gillam, Eds. Springer London, 2010, pp. 195–217.

[8] L. Wang, M. Kunze, J. Tao, and G. von Laszewski, "Towards Building a Cloud for Scientific Applications," *Adv. Eng. Softw.*, vol. 42, no. 9, pp. 714–722, Sep. 2011.

[9] L. Wang, G. von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu, "Cloud Computing: a Perspective Study," *New Gener. Comput.*, vol. 28, no. 2, pp. 137–146, 2010.

[10] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-Hadoop: MapReduce Across Distributed Data Centers for Data-intensive Computing," *Futur. Gener. Comput. Syst.*, vol. 29, no. 3, pp. 739–750, Mar. 2013.

[11] "Apache Hadoop," *Apache*. [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Feb-2015].

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004, p. 10.

[13] Y. Lee, W. Kang, and H. Son, "An Internet traffic analysis method with MapReduce," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010, pp. 357–361.

[14] A. Kimball, S. Michels-Slettvet, and C. Bisciglia, “Cluster computing for web-scale data processing,” *ACM SIGCSE Bull.*, pp. 116–120, 2008.

[15] B. White, T. Yeh, J. Lin, and L. Davis, “Web-Scale Computer Vision using MapReduce for Multimedia Data Mining,” *University of Maryland*, 2010. [Online]. Available: http://www.umiacs.umd.edu/~lsd/papers/brandyn-kdd-cloud.pdf. [Accessed: 28-Feb-2015].

[16] S. K and V. MS, “Mining of Web Server Logs in a Distributed Cluster Using Big Data Technologies,” *Int. J. Comput. Sci. Appl.*, vol. 5, no. 1, pp. 137–142, 2014.

[17] U. Kang, C. E. Tsourakakis, and C. Faloutsos, “PEGASUS: Mining Peta-scale Graphs,” *Knowl. Inf. Syst.*, vol. 27, no. 2, pp. 303–325, May 2011.

[18] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1426–1437, Aug. 2009.

[19] M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, “Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data,” *Smart Grid, IEEE Trans.*, vol. 6, no. 1, pp. 360–368, Jan. 2015.

[20] “Amazon Elastic MapReduce,” *Amazon*. [Online]. Available: http://aws.amazon.com/elasticmapreduce/. [Accessed: 28-Feb-2015].

[21] M. Zima, M. Larsson, P. Korba, C. Rehtanz, and G. Andersson, “Design Aspects for Wide-Area Monitoring and Control Systems,” *Proc. IEEE*, vol. 93, no. 5, pp. 980–996, 2005.

[22] P. M. Ashton, G. A. Taylor, M. R. Irving, A. M. Carter, and M. E. Bradley, “Prospective Wide Area Monitoring of the Great Britain Transmission System using Phasor Measurement Units,” in *Power and Energy Society General Meeting, 2012 IEEE*, 2012, pp. 1–8.

[23] M. Rihan, M. Ahmad, and M. Salim Beg, “Phasor measurement units in the Indian smart grid,” in *Innovative Smart Grid Technologies - India (ISGT India), 2011 IEEE PES*, 2011, pp. 261–267.

[24] Y. Tang and G. N. Stenbakken, “Traceability of calibration for Phasor Measurement Unit,” in *Power and Energy Society General Meeting, 2012 IEEE*, 2012, pp. 1–5.

[25] P. Trachian, “Machine learning and windowed subsecond event detection on PMU data via Hadoop and the openPDC,” in *Power and Energy Society General Meeting, 2010 IEEE*, 2010, pp. 1–5.

[26] M. Edwards, A. Rambani, Y. Zhu, and M. Musavi, “Design of Hadoop-based Framework for Analytics of Large Synchrophasor Datasets,” *Procedia Comput. Sci.,* vol. 12, no. 0, pp. 254–258, 2012.

[27] “OpenPDC,” *CodePlex*. [Online]. Available: http://openpdc.codeplex.com/. [Accessed: 28-Feb-2015].

[28] M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, “Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data,” *Smart Grid, IEEE Trans.*, vol. 6, no. 1, pp. 360–368, Jan. 2015.

[29] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, “Hadoop Performance Modeling for Job Estimation and Resource Provisioning,” *Parallel Distrib. Syst. IEEE Trans.*, vol. PP, no. 99, p. 1, 2015.

[30] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, “Starfish: A Self-tuning System for Big Data Analytics,” in In CIDR, 2011, pp. 261–272.

Chapter 2

Background

The MapReduce programming model has become a major computing platform that supports parallel and distributed processing for data-intensive applications such as network traffic analysis, machine learning, web data processing, and scientific simulation. Hadoop is the most prevalent open-source implementation of the MapReduce programming model and it has been taken up by an increasingly wide user community for big data analytics. This chapter first provides an overview of the MapReduce programming model and its implementation systems. It then introduces the Hadoop MapReduce framework and its optimization techniques. This Chapter also describes some recent developments of Hadoop including Hadoop 2 and the eco-system of Hadoop.

2.1 MapReduce Programming Model

The MapReduce programming model originally proposed by Google in 2004, has become a major programming model for parallel and distributed process of a large scale dataset in computer cluster environments [1]. In the MapReduce programming model, the computation is specified in the form of a map function and a reduce function. The map function process a block of dataset as a (key, value) pair and produces map output in the form of a list of (key, value) pairs. The intermediate values are grouped together based on the same key e.g. k_2 and then pass to the reduce function. The reduce function takes the intermediate key k_2 along with its associated values and processes them to produce a new list of values as final output. The map function and the reduce function are executed independently on allocated resources which shows high parallelism.

$$
\begin{aligned}
&map\_function(k_1, v_1) \rightarrow list(k_2, v_2) \\
&reduce\_function(k_2, list(v_2)) \rightarrow list(v_3)
\end{aligned}
$$

The logical dataflow diagram of the MapReduce programming model is shown in Figure 2.1 which presents a weather dataset with various air temperatures in different years. The program needs to find the highest air temperature in each year. The year is represented as the key and the air temperature is represented as the value in the dataset [2].

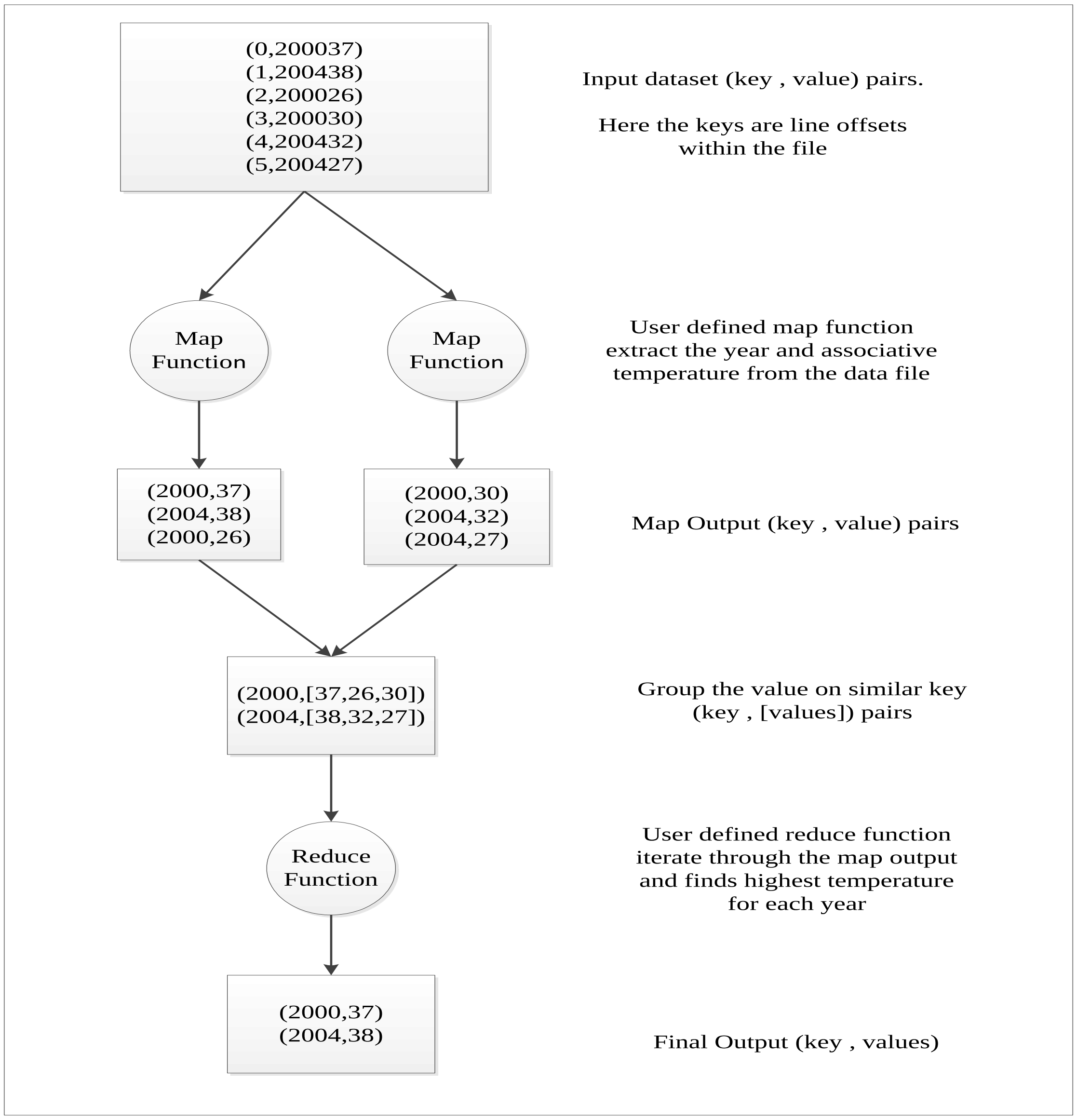
(0,200037)
(1,200438)
(2,200026)
(3,200030)
(4,200432)
(5,200427)
Input dataset (key , value) pairs.
Here the keys are line offsets within the file
Map Function
Map Function
User defined map function extract the year and associative temperature from the data file
(2000,37)
(2004,38)
(2000,26)
(2000,30)
(2004,32)
(2004,27)
Map Output (key , value) pairs
(2000,[37,26,30])
(2004,[38,32,27])
Group the value on similar key (key , [values]) pairs
Reduce Function
User defined reduce function iterate through the map output and finds highest temperature for each year
(2000,37)
(2004,38)
Final Output (key , values)

Figure 2.1: A logical data flow of the MapReduce programming model.

The MapReduce programming model was initially developed for Web base data processing but now it has been applied in other domains of data intensive applications such as machine learning, scientific simulation, healthcare data analysis, and Web data mining. The remarkable features of the MapReduce programming model include fault-tolerance, simplicity and scalability. MapReduce is a highly scalable computing model to enable thousands of inexpensive commodity computers to be used as an effective computing platform for distributed and parallel computing. The model provides facilities to automatically detect and handle node failures scenario without any effect on the computation completion. It is a simple programming model because it allows the application developers to provide only a sequential implementation of application logic expressed in functional-style (i.e. map function and reduce function) and the runtime system deals with low-level parallelization details, i.e. partitioning input dataset, scheduling program execution across multiple nodes of a computer cluster, handling node failures and managing inter-nodes communications. Hence, the MapReduce programming model reduces difficulties of parallel programming, so that programmers can easily achieve the low level parallelism on cluster nodes for complex tasks.

The MapReduce programming model has a number of implementations such as Hadoop [3], Dryad [4], Phoenix [5], Mars [6] and Sector/Sphere [7]. The Dryad [4] is a general purpose distributed execution system proposed by Microsoft. The Dryad engine is based on dataflow graph where computations are expressed as vertices. The vertices can be executed on a set of computers and inter-computer communication can be achieved through channels that connect the vertices. The Phoenix [5] was proposed by Stanford University, a programming API and runtime system based on Google MapReduce programming model. It can automatically create threads and dynamically schedule the threads on multiple processors. The processor failure is automatically handled in the Phoenix system for fault tolerance. It is mainly designed for multi-core and multiprocessor systems [8]. He et al. proposed Mars [6], a MapReduce implementation on Graphic Processor Unites (GPUs). The Mars APIs automatically parallelize the map function and the reduce function on GPUs threads. The Mars can perform better than CPU-based MapReduce implementations because GPUs can provide massive parallelism. Gu et al. proposed Sector/Sphere [7] for graphic processing applications. In the Sector/Sphere system, sector is a distributed file system across commodity machines and used for data storage. It also provides scalability and fault-tolerance facilities. The data parallel computation is achieved through sphere i.e. the sphere can be used to process data stored in the sector in parallel. Among the aforementioned implementation systems of the MapReduce programming model, the Hadoop framework is the most widely used MapReduce platform due to its open source nature. The details of the Hadoop framework are given in the next section.

2.2 Hadoop MapReduce Framework

Hadoop [3] is an opens source implementation of the MapReduce programming model and has become the foremost computing platform for big data analytics. It was originally developed by Doug Cutting[1] and Mike Cafarella[2] in 2005. Cutting was working that time at Yahoo. Since then, Hadoop has become a core project of Apache™. The Apache™ Hadoop is a framework written in Java that distributes and parallelizes computation on massive datasets across a cluster of computers using simple programming model (MapReduce programming model). It has become the most prevalent framework for big data analytics and it is being used by many organization such as Yahoo, Facebook, YouTube, Twitter, Google, LinkedIn [9] to process and analyze their massive amounts of data . Today popular big data analytics service providers such as IBM, Oracle, Microsoft, Dell, Cloudera and Hortonworks either have been offering Hadoop-related products (such as Infosphere BigInsights and Exadata) or providing support to users on Hadoop MapReduce (Cloudera, HortonWorks) [10]–[14].

The Hadoop MapReduce framework is highly scalable and it can be scaled up from a single machine to tens of thousands machines, each of which offering local computation and data storage. The size of the Hadoop cluster can shrink or expand dynamically based on workload. The Hadoop MapReduce framework is developed with fundamental hypothesis that machine failure is common in cluster computing and it should be handled

[1] http://en.wikipedia.org/wiki/Doug_Cutting

[2] http://web.eecs.umich.edu/~michjc/bio.html

automatically in software level by the framework. Therefore, fault-tolerance and automatically machine failure handling techniques are included in the framework.

2.2.1 Hadoop Architecture

The Hadoop MapReduce framework mainly includes: MapReduce and Hadoop Distributed File System (HDFS). The architecture of the Hadoop MapReduce framework is shown in Figure 2.2.

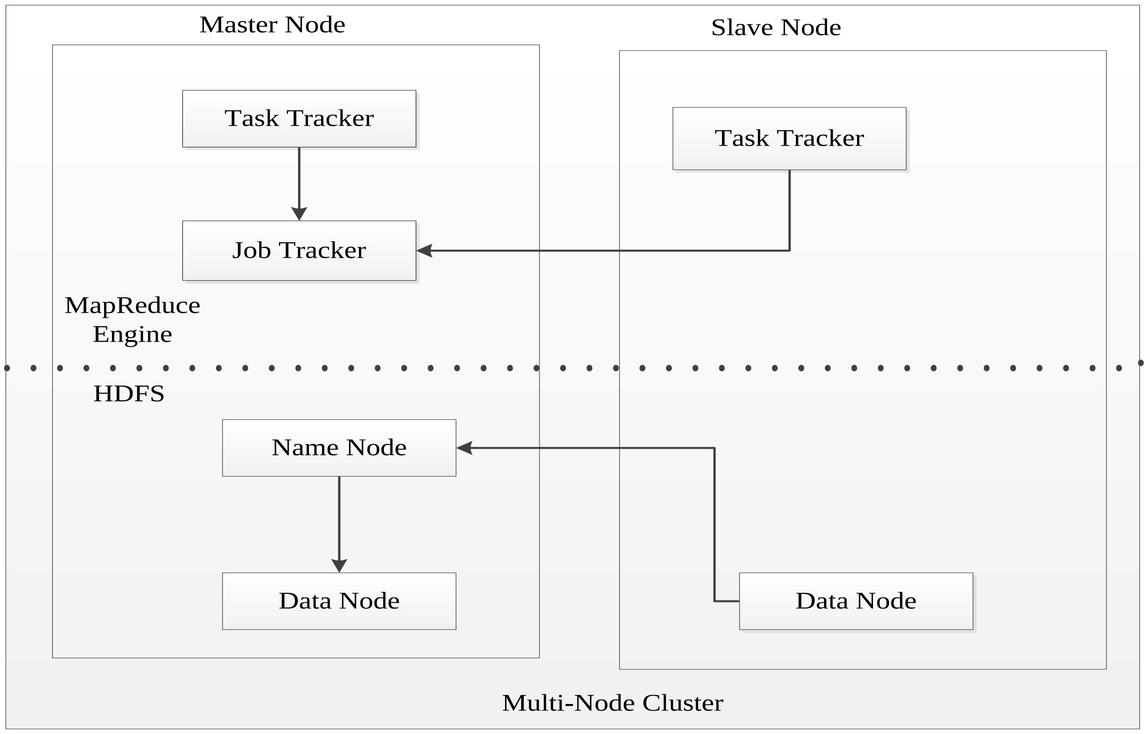

Figure 2.2. Architecture of the Hadoop MapReduce

The MapReduce divides the computation into map tasks and reduce tasks and executes them parallel on a cluster of nodes. It consist of a job tracker and a number of task trackers services. The job tracker is running on the master node (Name Node) and it is responsible for controlling the overall operation of the MapReduce framework. It manages the task tracker, assigns tasks to a task tracker, monitoring the progress of running tasks and dealing with node failure situations. The task trackers run on the slave nodes (worker nodes / Data Nodes) which actually execute all the map tasks and the reduce tasks i.e. all the map tasks and the reduce tasks run on the worker nodes. The task tracker periodically coordinates with the job tracker through a heartbeat message to updates on the progress of the running tasks. If the job tracker is not receiving the heartbeat message from a particular task tracker for a certain period of time, then the job tracker declares that the particular task trackers is a dead worker and automatically assigns the running tasks to the next available worker node.

2.2.1.1 MapReduce

MapReduce is a processing engine of the Hadoop MapReduce framework. A Hadoop MapReduce job is a unit of works that executes across multiple computing nodes and it completes in multiple phases i.e. map phase and reduce phase. The details of these phases are available in Chapter 4. In the map phase, each map task processes a block of input dataset that is generally stored in a distributed file system. The input dataset is typically divided into blocks of pre-defined size (i.e. 64MB or 128MB), and distributed over cluster nodes. The map tasks read the data blocks and applies the user-defined map function. The map output (intermediate files) produced by the map function is collected in physical memory. It is written periodically from the physical memory into a local disk of a processing node that executes the map task. The locations of the map output are passed to the master node that is responsible to forward to a node that executes the reduce task. In the reduce phase, the reduce tasks read the map output from remote locations, sort it by the map output key, so that all the values of the same key are grouped together. After that, the reduce tasks iterate through sorted map output files and passes unique intermediate key and the associated values to the reduce function. The output of the reduce function is written into a distributed file system as a final output. The number of final output files are depended on the number of the reduce tasks initiated. The Hadoop job execution flow is shown in Figure 2.3.

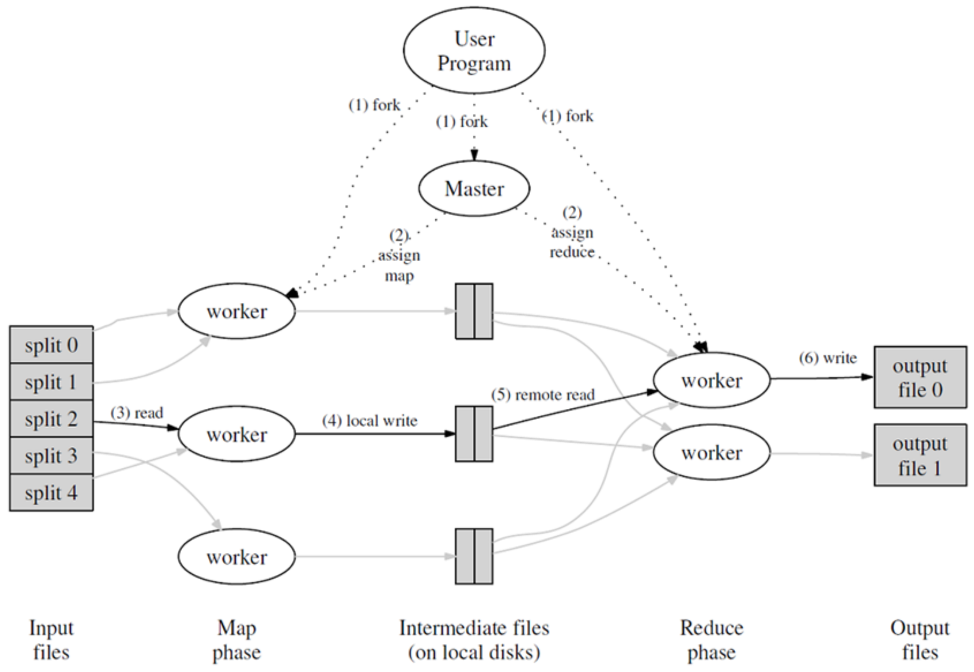

Figure 2.3: The Hadoop MapReduce job execution flow[3]

The MapReduce programming model executes the map function and the reduce function on multiple computing nodes in order to achieve parallelism. The number of map tasks and the reduce tasks that can run simultaneously on a cluster nodes is depended on the number of map slots and reduce slots configured on the worker nodes. A slot is a unit of resources (CPU, physical memory) that can be assigned to a task. The number of slots (i.e. map slots and reduce slots) can be configured through a configuration file. For examples, a cluster has 10 worker nodes, and if 2 map slots and 2 reduce slots are configured on each worker node. Then total 20 map tasks and 20 reduce tasks will be executed parallel. If the number of map slots and the number of reduce slots are less than the number of map tasks and the number of reduce tasks then a job will be completed in multiple waves. The job waves are more explained in Chapter 4.

[3] Simplified Data Processing on Large Clusters, http://dl.acm.org/citation.cfm?id=1327492

2.2.1.2 Hadoop Distributed File System

When a dataset is continuously growing, it might not be possible for a single computer to store and process the continuously growing dataset. It becomes compulsory to partition the dataset and distributes across a network of computers. File systems that manage storage across a network of computers are called distributed file systems. Hadoop has its own distributed file system called HDFS (Hadoop Distributed File System) which is designed for storing massive amounts of data across a cluster of nodes [15][2] . The HDFS is designed based on Google File System (GFS) [16], [17] with the idea that the most effective way of data writing and reading as write-once and read-many times. It is a highly fault-tolerant system and provides high throughput access to application data. The HDFS system is mainly designed for batch processing rather than interactive system [18]. When a dataset is copied to the HDFS, the HDFS divides the dataset into blocks of an equal size, makes multiple replicas of each block and distributes throughout a cluster of nodes (Data Node) as an independent units. The replicas of the block are stored on different nodes and racks. The default size of a data block is 64MB and the number of replicas is 3, however, a user can change the size of block and number of replica in HDFS configuration file. The multiple replicas support fault-tolerance and availability of a data. For examples, if a node is crashed, the data blocks stored on it will be available on other nodes.

The HDFS system consists of a single Name Node service and several Data Node services as shown in Figure 2.4. The Name Node service is running on the master node and is responsible for managing the file system namespace and standardizes access to files. The Data Node services are running on slave nodes and are responsible for storing the data files, creating and deleting the blocks as well the replicas upon instructions received from the Name Node. A detail about the HDFS system is available in Chapter 3.

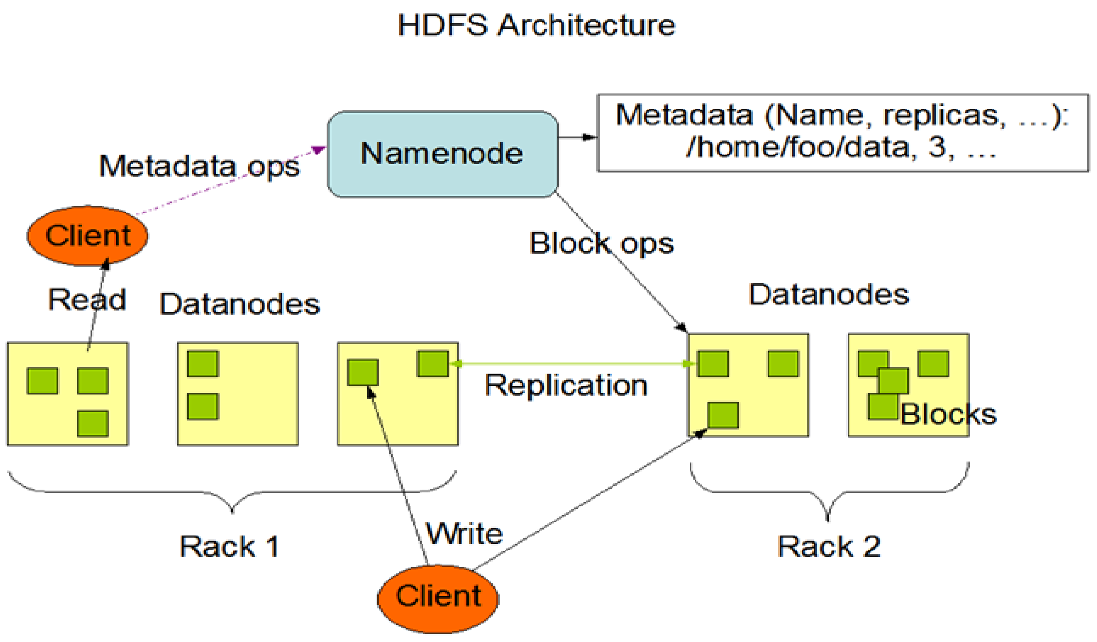

Figure 2.4: The HDFS architecture[4]

2.2.2 Hadoop MapReduce Framework Optimization Approaches

The performance of the Hadoop MapReduce framework can be improved from different aspects such as:

- Job scheduling / Task scheduling
- Data locality algorithms
- Configuration parameters tunings

2.2.2.1 Job Scheduling or Task Scheduling

Job scheduling or task scheduling in the Hadoop MapReduce is employed to efficiently manage workload among the processing nodes and effectively share the resources of a Hadoop cluster between different jobs and users. With the help of scheduling techniques, more than one user can execute multiple jobs in parallel and as a result the cluster resources would be utilized effectively. On the other hand, without a sophisticated

[4] http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

scheduling mechanism, the performance of the Hadoop cluster can be affected due to imbalance workload distribution and unfair resources sharing. Moreover, the Hadoop framework supports both homogeneous and heterogamous environments. Distribution of the workload among the processing nodes in the homogeneous environment is relative simple and easy, however, in the heterogeneous environment, the workload is distributed based on the processing capacity of the nodes.

To effectively share the resources between the jobs and balance the workload among the processing nodes, Hadoop framework has come with default schedulers such as First-Come-First-Serve (FCFS) [19], [20], Fair Scheduler [21] and Capacity Scheduler [22] . The FCFS allocates all resources to a job on receipt, i.e. a job submitted first will get all the resources and a job submitted later will wait until the first job is completed. The Fair Scheduler divides the jobs into pools and fairly allocates the resource shares to each pool so that all jobs acquire an equal share of the cluster resources over time. It also solves the problem of FCFS where short execution jobs have to wait for a long execution job to complete. The Capacity Scheduler is similar to Fair Scheduler; however, the Capacity Scheduler is used in a large cluster and shares resources among multiple organization and organizes the jobs into queues. Apart from the default Hadoop scheduling, researchers have proposed job/task scheduling techniques to enhance the performance of the Hadoop framework. Sandholm et al. [23] proposed a dynamic priority mechanism that allows scheduler to allocate resources dynamically to multiple users based on their priorities and demands. It assigns the resources on a proportional basis in the form of map slots and reduce slots. It also gives incentives to users to optimize and customize their allocated resources in order to fulfill their job requirements. Kc et al. proposed [24] a constrain-based scheduler that considers the users deadlines as input and schedules only those jobs that meet the specified deadlines. It employed cost model that estimates the job execution time and it schedules the job if the job completes within a deadline. Nguyen [25] proposed a hybrid scheduling algorithm based on a dynamic priority to minimize the response time of a variable length of concurrent running jobs. The dynamic priority is calculated based on three factors, i.e. the waiting time of a job, the length of execution time of the job, and unscheduled tasks of the job. The algorithm relaxes the ordering of tasks assignment in order to achieve data locality. The aforementioned schedulers try to improve the performance of a Hadoop framework with the assumption that the processing nodes of the cluster are homogeneous and thus they do not consider the heterogeneity of the underlying resources.

For a heterogeneous environment of the Hadoop cluster, several researchers have proposed scheduling algorithms in order to improve the performance of the Hadoop cluster. Zaharia et al. proposed LATE [26] and Chen et al. proposed SAMR [27] to improve the performance of the Hadoop in heterogeneous environment. Both approaches deals with straggler tasks (a task which has slow progress) and executes speculative tasks (backup task) of the straggler tasks on another machine in order to minimize the overall execution time of a job because the straggler tasks significantly increase the execution of the job. Both the approaches find strangler task by estimating the execution completion time of all the running tasks. The LATE algorithm estimates the execution time of the tasks in a static way while SAMR algorithm uses dynamic approach to estimate the execution time of the running tasks. Furthermore, SAMR classified the cluster nodes into fast nodes and slow nodes and initiates the backup task only on the fast nodes. Sun et al. further improved the SAMR and proposed ESAMR [28] which takes the types of the jobs into consideration because different types of the jobs have different task execution time in the map phase and reduce phase. The ESAMR employs k-mean to classify the jobs into different classes. Rasooli et al. [29] proposed a hybrid scheduling approach for Hadoop heterogeneous environments. The approach is based on three scheduling algorithm such as FCFS, Fair Scheduler and COSHH [30] scheduler. The selection of the scheduler is made based on the current utilization of the resources and the total number of waiting tasks. If the system is under-utilized (i.e. the number of available slots is greater than the number of submitted tasks) then the hybrid approach will pick up FCFS scheduler. If the system utilization is balanced then the system will use Fair Scheduler otherwise the system will employ COSHH scheduler. Kumar et al. [31] proposed a Context Aware Scheduler for Hadoop (CASH) to improve the performance of the framework. The CASH schedules the tasks based on job characteristics (i.e. CPU-intensive or I/O intensive) and processing nodes characteristics (i.e. Computation good or I/O good). It classifies the jobs and the processing nodes based on their characteristics. The tasks are scheduled on the processing nodes that fulfill the requirements. Further readings on Hadoop scheduling are available in [32].

2.2.2.2 Data Locality Algorithms

Hadoop has become a major computing platform for intensive-data applications. To efficiently process a large amount of data, Hadoop should provide an efficient data locality scheduling mechanism for enhancing the performance of the Hadoop system in a shared cluster environment. One of the Hadoop principles is that moving computation is cheaper then moving data when dealing with large amounts of datasets. This principle indicates that it is often better to move the computation close to where the data is located rather than to move the data to the computation node where the application is running. This is especially true when the size of data is very large because migration of computation reduces the network congestion and improves the overall performance of the system. When a computation task is moved closer to data it consumes, this is called data locality. Today a cluster can have thousands of shared nodes which transmit massive data that impose network load and create congestion, so an efficient scheduler must avoid unnecessarily data transmission. Scheduler considers the data locality as it is a determining factor for the MapReduce performance mechanisms because network bandwidth is scarce resource for these systems. In fact, a high locality of data enhances the throughput of the system [61].

For each node, all map tasks are classified into three levels of locality according to the distance between the input data and computation nodes. The most efficient locality is the first level locality where the processing map task is launched on the node holding the task input data called the node level locality. When a task cannot achieve the first level locality then scheduler executes the task on the node where the computation node and data node located in the same rack called rack level locality (second level). If the task still fails to achieve the second level locality then a scheduler launches the task on a node in a different rack which is called off-the-rack level locality (third level). If the data locality is not achieved, data transferring and I/O cost can seriously affect the performance because of the shared network bandwidth.

As the data locality is a determining factor in Hadoop. There are several factors that affect date locality such as the size of a Hadoop cluster, the number of data replications (replicas) and job execution stage. In a large cluster with a small number of jobs, the probability of the data locality is low. For example, if a job has 5 map tasks and is submitted to a cluster with 100 nodes, it is unlikely to get a high locality rate. Since each task has 3 copies of the input data which are distributed on 3 different nodes, therefore, at most 15 out of the 100 nodes have input data for the job. That is, the probability of the data locality for the job is 15%. If number of nodes is decreased to 50 then the data locality of the job will be increased to 30%.

Similarly, the number of replicas and job execution stage also affect the data locality. Increasing the number of replicas of input data improves the data locality but it consumes extra storage. At the job initialization stage, the probability of a job data locality is high because there are a large number of unmapped tasks and the required input data of these unmapped tasks are available on large number of nodes. While at the job end stag, the probability of a job data locality is low because a small number of unmapped tasks are left and the required input data of these tasks are available on small number of nodes.

The Hadoop default scheduler schedules jobs using FCFS and already considers data locality [33]. When the master node receives a heartbeat from a slave node which indicates that a free map slot is available, the job tracker on the master node first tries to find the map task in the head-of-line job whose input data is stored on that node. If it is found then a node level locality is achieved and task will be launched on that node. When node level locality is impossible then the job tracker tries to seek a rack level locality. If it is still fail then task is arbitrary picked up and launched on as off-the-rack node. This simple scheduling algorithm favors data locality but has deficiencies. For example, this algorithm strictly follows the FCFS policy where tasks are scheduled one by one and each task is scheduled without considering its impact on other tasks.

Let us consider a Hadoop cluster of three nodes (N1, N2 and N3) as shown in Figure 2.5. Each node has at least one free map slot. Let us assume that there are three tasks (t1, t2 and t3). Each task input data has multiple copies which are stored over multiple nodes for reliability purpose. Task t1 input data is stored on nodes N1, N2 and N3 (DT1), task t2 input data is stored on nodes N1 and N2 (DT2) and task t3 input data is stored on node N1 (DT3) as shown in Figure 2.6.

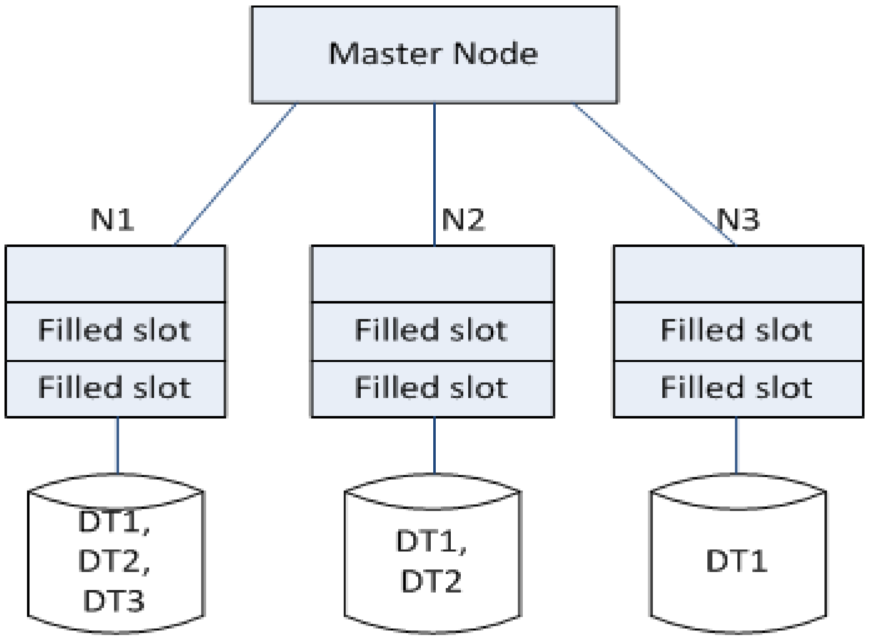

Figure 2.5: A Hadoop cluster with 3 nodes.

The Hadoop scheduler assigns task t1 to node N1 and achieves the node level locality, task t2 is assigned to node N2 and it also achieves the node level locality. There is now only node C that has the idle slot and only one unscheduled task t3 and this task must be assigned to node C, as shown in the Figure 2.6. To summarize, both tasks t1 and t2 achieve data locality while task t3 loses data locality. The reason is that the Hadoop scheduler processes the tasks one by one rather than considers the tasks on all the available idle slots. All the tasks can achieve the data locality if the scheduler processes all the tasks on all available idle slots at once as shown in Figure 2.7.

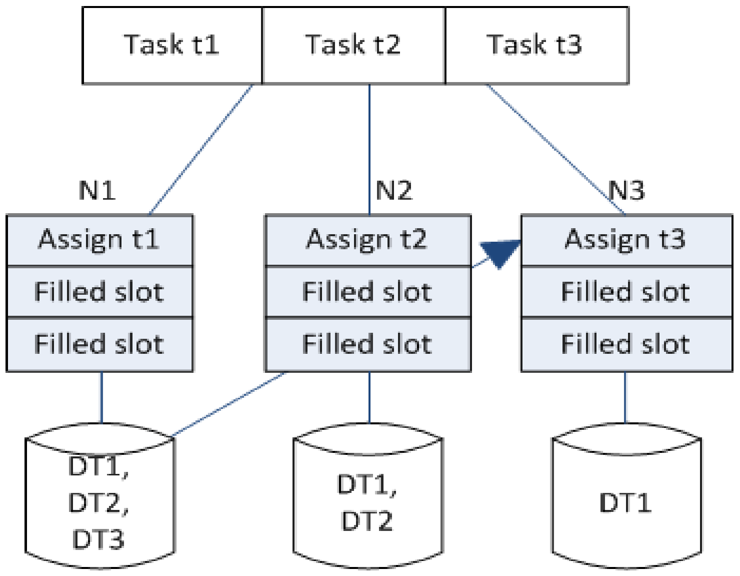

Figure 2.6: Task 3 is assigned without data locality.

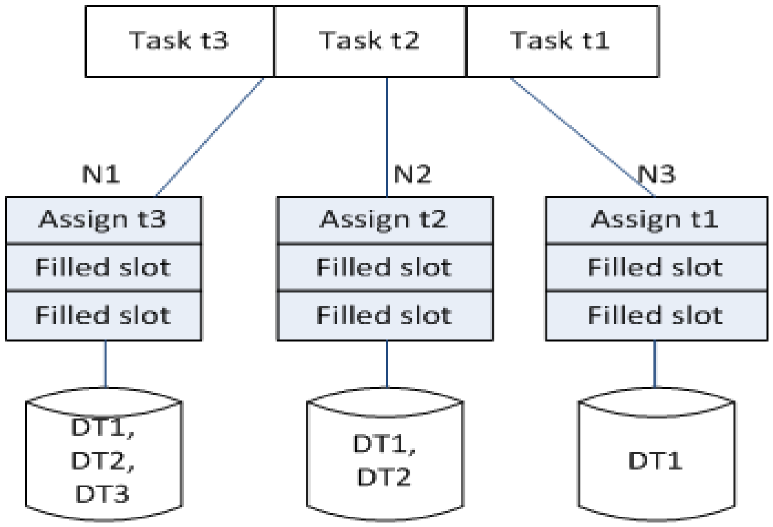

Figure 2.7: All the 3 tasks are assigned with data locality.

The default Hadoop scheduler provides some mechanisms to improve data locality but have some inherent deficiencies. To improve the performance of the Hadoop system, researchers have proposed numerous data locality aware scheduling algorithms.

Abad et al. designed DARE (Adaptive Data Replication) algorithm [34] to improve data locality by dynamically replicating the popular data on different nodes. They have proposed (i) a greedy approach and (ii) a probabilistic approach. In the current implementation, when it is impossible for a map task to gain local data, the MapReduce framework fetches data from a remote node (data located at a different node) for processing and discards when map task is completed. The greedy approach takes the advantage of remotely fetched data, makes subset of that data and inserted into HDFS at the node that fetched it. However, the greedy approach cost huge disk storage due to replicating all the fetched data. To address this issue, the DARE using eviction mechanism using LRU (Least Recently Used) policy that delete least recently used data blocks to make space for the new replica. Unlike the greedy approach, the probabilistic approach does not replicate remotely fetched data immediately but replicate only popular data. In this approach, an individual node runs algorithm independently to generate replica of most popular data. The knowledge of dynamically replicated data is transmitted to the Name Node, so that this information will be made available to the scheduler and other users of file system to achieve better data locality. This approach also applies the aging eviction mechanism, to quickly evict the files with a decreasing popularity.

Zaharia et al. [33] developed an algorithm called delay scheduling to enhance the data locality rate in a Hadoop environment. The delay scheduler is applied into Fair Scheduler in Hadoop. Fair Scheduler has changed from allocating equal share (time slot) to each job to allocating equal share to each user. Each user has its own pool in a shared cluster and a minimum share (a minimum number of slots) is assigned to each user. If a user cannot use their time slots, other users can use these slots instead. If a user cannot get the minimum share, preemption occurs, which reallocates the resources among the users. There are two approaches of preemption (i) killing the running jobs or (ii) wait for running jobs to complete. Killing a running job immediately scarifies the time it had been running while the waiting approach does not have such an issue but scarifies the fairness. The delay scheduling algorithm uses the waiting approach to achieve data locality and it defines as “when a job cannot launch a local-map task then it wait for small amount time, letting the other jobs launch the task instead”. The delay scheduling relaxes the strict job order for task assignment and delays jobs execution if the job has no map task local to the available slave node. The maximum delay time D is specified. If a job map task has been skipped for a longer time (i.e. longer than the D time unit), it is allowed to launch a rack-level task. If it is skipped for further longer times then it is allowed to launch an off-rack level task. These skip times are called delay times and are an important factor in this algorithm. The values of the delay time are set either by default which is 1.5 times to slave node heartbeat or based on a rate at which the slots free up which is less than average task length.

He et al [35] developed a matchmaking scheduling algorithm to enhance data locality in a MapReduce cluster. The main idea behind this algorithm is to give every node a fair chance to grab a local task before assigning a non-local task. Like the delay scheduling, the matchmaking algorithm also relaxes the strict job order when assigning a map task to a node. That is, if a node fails to find a local job in the queue; the algorithm will continue to search the succeeding jobs. To give a fair chance to every node to get a local map task, when a node cannot find a local map task for the first heartbeat, no non-local task will be assigned to the node i.e. the node gets no task for this heartbeat interval. If a node still fails to find a local map task for the second heartbeat interval, the matchmaking technique will assign a non-local task to the node to avoid wasting computation resources. This algorithm assigns a locality marker value to every node to mark its status. If none of jobs in the queue has a local map task to a node, depending on the status of this node (locality marker value), the matchmaking algorithm will decide whether or not to assign the node to a non-local task. When a new job is added, all the slave node locality marker values will be cleared because the new job may comprise a local map task for some slave nodes.

Sangwon et al. [36] proposed two innovative techniques, i.e. Prefetching and Pre-shuffling that can enhance the overall performance of a MapReduce cluster. The prefetching technique enhances data locality while the pre-shuffling reduces the shuffling of intermediate result data produced by a map function. The prefetching is a bi-directional technique where on one side the complex computation is performed and on the other side to be required data is prefetched and assigned to the corresponding task. This technique prefetched the required data block of map tasks close to the computation node or to the local rack in pipeline manner. The prefetcher module also monitors the synchronization status between the computation and prefetches as both activities are performed simultaneously. The pre-shuffling technique tries to predict the target reducer where the intermediate result data are partitioned before the execution of mapper, in order to reduce the network overhead.

Zhang at el. [37] designed Next-K-Node scheduling (NKS) algorithm to improve data locality of map task in homogeneous environment and has been implemented in Hadoop 0.20.2. The algorithm first preferentially schedules the tasks which satisfies the node level locality. If no such a map task is available then the NKS method calculates the probabilities of each task and schedules the one with the highest probability. The NKS method generates the low probabilities for the tasks of whose input data is stored on the next k nodes, so that it can reserve these tasks for these nodes. In this method the main factor is the next k node which is predicted node to issue requests for the next task. In this method the next k node is determined based on the progress report of the running task. In Hadoop, task trackers periodically report the progress of the running tasks to the job tracker. To calculate the progress of the running task, the size of the processed data is divided by the size of the whole input data. In homogeneous environment all the nodes are identical in term of processing and disk capacities and therefore process the task at the same speed. So the task with highest progress will be completed first and the node running this task will issue a request for the next task earlier than other nodes. Therefore, the NKS method predicts the next k node on the basis of progress of the running tasks. However, in the case of different input data sizes of the map tasks, the NKS method cannot predict the next k node correctly on the basis of tasks progress because these tasks will be completed at different times. In this case the NKS method takes an imaginary task of whose input data size is equal to the map task. To correctly predict the next k node, the progress of the imaginary task is mapped with original task progress.

2.2.2.3 Configuration Parameters Tunings

Hadoop has extraordinary features such as scalability, resilience and automatic code parallelization. Despite that, Hadoop is a large and complex framework including a number of components that interact with each other across multiple machines. The performance of a Hadoop job is sensitive to each component of the Hadoop framework, underlying hardware, network infrastructure and Hadoop configuration parameter settings. It is becoming difficult for Hadoop users to setup an optimized Hadoop cluster due to the large number of configuration parameters. The current version of the Hadoop framework has more than 190 configuration parameters and some of them have a significant effect on the performance of a Hadoop job. Recent research shows that a small change in one of the configuration parameter values can have a huge impact on the performance of a Hadoop job when the job runs on the same amounts of resources and process the same size of an input dataset [38]. In addition, there are complex inter-dependencies among the configuration parameters, i.e. changing the value of one configuration parameter can have a huge impact on the other configuration parameters [39]. This thesis provides empirical evidence in Chapter 5 to demonstrate that how the performance of a Hadoop is affected by changing the values of the configuration parameters.

The performance of the Hadoop framework is sensitive to the configuration parameters. Therefore, numerous performance models and guidelines have been proposed in literature. The guidelines proposed in [2], [40], [41] consider only the processing capacity (i.e. CPUs and physical memory) of nodes for recommending optimum values for the configuration parameters . The models presented in [42]–[44] have targeted specific jobs (i.e. query based jobs and short jobs). Enhancing the performance of the Hadoop system based on resource provisioning is presented in [45]–[47]. There are some sophisticated performance models presented in [48]–[51] that automatically recommend optimum configuration parameters based on historical job profile information.

This thesis also presents an optimization work (details are presented in Chapter 5) that recommends optimum configuration parameter settings in order to improve the performance of a Hadoop job. The optimization work first employs Gene Expression Programming (GEP) to construct an objective function. It then employs Particle Swarm Optimization (PSO) technique to search an optimum values for the configuration settings. The complex inter-relation among the configuration parameters are considered during the optimization process.

2.3 Hadoop Ecosystem

The Hadoop paradigm is a major computing platform for large data storage and analysis, however, it is not effective for all problems that comprising huge datasets. The Hadoop main components i.e. MapReduce and HDFS are mainly designed to process unstructured datasets, though, the performance of the Hadoop is affected when it processes old-fashion structured datasets. This is because the Hadoop paradigm is not originally designed to processes structured datasets. In addition, Hadoop system is unable to process the datasets that are stored outside the HDFS. To overcome these issues, several Hadoop ecosystems have been developed over the past few years. A brief introduction of some of them is given below:

Apache Pig: Pig is a scripting language (data flow language) that is developed by Yahoo for analyzed large amount of datasets in parallel through a language called Pig Latin [52]. The Pig compiler automatically converts the Pig script into series of MapReduce programs so that it can be executed on a Hadoop cluster. Pig script can run on a single virtual machine using JVM or it can be executed on cluster of nodes. In Pig script, commands such as filtering, grouping and joining can be expressed in the form of user-defined functions. Pig is basically developed for batch processing. It is not appropriate for all types of data processing applications. If a user query only touch a small portion of data in a huge dataset, the Pig will not perform better because it will scan the whole dataset or at least a huge portion of the dataset (because the Pig does not support random access). The Pig commands (i.e. filtering, grouping, describe and joining) are showing the impression that the Pig is similar to the SQL, however there is significant difference between Pig and the SQL as presented in [2]

Apache Hive: Hive is a query language for data warehousing on top of Hadoop. It is designed by Facebook to execute SQL like statements on a large volume of datasets generated by Facebook every day and stored in HDFS. Hive interacts with dataset via HiveQL, a Hive query language based on SQL. Hive can be fitted between Pig and traditional RDBMS because like the RDBMS, Hive uses relation (table) with a schema to store the dataset and similar to the Pig, Hive use distributed storage (HDFS) to store the tables. Users who are familiar with map/reduce programming can express the logic into map functions and reduce functions and plug into Hive, if it is difficult for them to express the logic in HiveQL [2], [53].

Apache HBase: HBase is a scalable distributed column-oriented table inspired from Google BigTable [54] and designed on top of HDFS. As the Hadoop MapReduce does not support random access to data, Hadoop applications can access massive datasets in real-time with random read/write access via HBase. It is not a RDBMS and does not support SQL but it has the ability to address the problems that the RDBMS cannot. For examples, it can store a large dataset and distributes the table on a cluster of nodes. HBase is basically used to store a large number of web pages (billion) as a WebTable and MapReduce programs are executed against the WebTable to retrieve information. The WebTable is accessed randomly and in real-time as users click on a websites. HBase automatically divides the table horizontally into multiple regions and distributes it on regional server machines. Each region consists of a subsection of a table. Initially, there is a one region (table), however, when the size of the table grows and reaches to a configurable threshold, the system automatically partitions the table in row-wise into two equal regions. In case of a very large table, HBase can have a cluster of servers and these servers can be managed through ZooKeeper services. ZooKeeper is discussed below. More readings on HBase system are available in [2][55].

Apache ZooKeeper: ZooKeeper is a centralized distributed coordination service for distributed applications. It is originally developed by Yahoo and later it has become part of the Hadoop ecosystem. The services provide by ZooKeeper include configuration management, synchronization, naming and group membership. HBase, Flume and HDFS HA (high availability) all depend on ZooKeeper [2], [56].

Apache Sqoop: Hadoop processes a vast dataset when it is stored in HDFS. If the dataset is stored outside HDFS e.g. in a relational database, then Hadoop program needs to employ external APIs. Sqoop is an open-source tool that provides facilities to users to efficiently fetch a huge dataset from a relational database into Hadoop for onward processing. In addition, Sqoop can transfer data from a relational database system into the HBase system. It currently works with the relational databases including MySQL, SQL server, Oracle, DB2 and Postgre SQL [2], [57].

Apache Flume: Apache Flume is a highly reliable and distributed service which can be used to automatically collect and aggregate a huge streaming data from different sources and transfer into HDFS. Initially it was developed to collect streaming data from web log but now it can be used to collect datasets from different sources and transfer into HDFS. The Flume architecture mainly includes source, sink (which delivers the data to HDFS), channel (a conduit which connects the source and sink) and agent (JVM that runs Flume services) [58].

2.4 Hadoop 2

Currently, there are two branches of Hadoop releases, i.e. Hadoop 1 and Hadoop 2. The current stable version of the Hadoop1, i.e. Hadoop-1.2.1-1 was released in November 2014 and the current stable version of the Hadoop 2, i.e. Hadoop-2.6.0 was released in November 2014 [59].

Hadoop1 is the most popular Hadoop framework for batch processing and shows the potential value of big data distributed processing. However, the Hadoop 1 is not attractive for interactive applications, machine learning applications and memory intensive applications. To support the above mentioned applications, Apache Hadoop developers have modified the major modules of the HDFS and the MapReduce, and presented Hadoop 2. The major advancements made in Hadoop 2 over Hadoop 1 includes (a) the HDFS federation and (b) the resource manager (YARN) and (c) HDFS HA (High Availability) [32] [60].

The Hadoop 1 supports only single Name Node that manages the whole cluster namespaces. Using HDFS federation feature, Hadoop 2 can support multiple Name Nodes in a single cluster, i.e. the entire cluster namespaces can be managed with multiple Name Nodes. The second inclusion in Hadoop 2 is YARN (Hadoop NextGen) which is a resource manager and works like Hadoop operating system. It is designed to separate resource management from data processing. Prior to Hadoop 2, the resource management and data processing was managed by MapReduce. Now, in Hadoop 2, MapReduce only handling the data processing and the resource management is managed by YARN. Another addition in Hadoop 2 is the HDFS HA that supports high availability of HDFS. Hadoop 1 suffers from a single point of failure as it supports only a single Name Node and the failures of it can make the HDFS cluster inaccessible. To overcome the single point of failure, the HDFS HA feature provides an option of redundant Name Nodes which can be configured in an active/passive mode.

2.5 Summary

This chapter presented the background of the MapReduce programming model and Hadoop MapReduce framework. This chapter also extensively reviewed a number of Hadoop job optimization approaches which are related to job scheduling, data locality and configuration parameter settings.

Reference

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004, p. 10.

[2] T. White, *Hadoop:The Definitive Guide*, 3rd ed. Yahoo press, 2012.

[3] “Apache Hadoop,” *Apache*. [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Feb-2015].

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007.

[5] K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, “Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources,” in *SIGPLAN Not.*, 2003, vol. 38, no. 10, pp. 216–229.

[6] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars: a MapReduce framework on graphics processors,” in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT ’08*, 2008, p. 260.

[7] Y. Gu and R. Grossman, L, “Sector and Sphere: The Design and of High Performance Data Cloud,” *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 376, no. 1897, pp. 2429–2445, 2009.

[8] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for Multi-core and Multiprocessor Systems,” in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, 2007, pp. 13–24.

[9] “PoweredBy - Hadoop Wiki,” *wikipedia.org*. [Online]. Available: http://wiki.apache.org/hadoop/PoweredBy.

[10] “Hadoop Wiki,” *wikipedia.org*. [Online]. Available: http://wiki.apache.org/hadoop/. [Accessed: 20-Feb-2015].

[11] “Hadoop for the Enterprise,” *IBM-Hadoop*. [Online]. Available: http://www-01.ibm.com/software/data/infosphere/hadoop/enterprise.html. [Accessed: 20-Feb-2015].

[12] “Managing Big Data by Using Hadoop and Oracle Exadata,” *Oracle*. [Online]. Available:http://download.oracle.com/otndocs/products/bdc/hadoop-loader/pdf/oow2011_managing_bd_w_hadoop_exadata.pdf. [Accessed: 20-Feb-2015].

[13] “Cloudera Hadoop Distribution,” *Cloudera*. [Online]. Available: http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html. [Accessed: 20-Feb-2015].

[14] “Hortonworks,” *wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/Hortonworks. [Accessed: 20-Feb-2015].

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1–10.

[16] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google File System,” in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003, pp. 29–43.

[17] “Google File System,” *wikipedia.org*, 2015. [Online]. Available: http://en.wikipedia.org/wiki/Google_File_System. [Accessed: 01-Feb-2015].

[18] “Hadoop Distributed File System,” *Apache Hadoop*, 2015. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Accessed: 19-Feb-2015].

[19] “Scheduling in Hadoop,” *IBM- Developer work*. [Online]. Available: http://www.ibm.com/developerworks/library/os-hadoop-scheduling/. [Accessed: 20-Feb-2015].

[20] B. T. Rao and L. S. S. Reddy, “Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments,” *Int. J. Comput. Appl.*, vol. 34, no. 9, p. 5, Jul. 2011.

[21] “Fair Scheduling,” *Apache Hadoop*. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html. [Accessed: 20-Feb-2015].

[22] “Capacity Scheduler,” *Apache Hadoop*. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html. [Accessed: 22-Feb-2015].

[23] T. Sandholm and K. Lai, “Dynamic proportional share scheduling in Hadoop,” in *JSSPP’10 Proceedings of the 15th international conference on Job scheduling strategies for parallel processing*, 2010, pp. 110–131.

[24] K. Kc and K. Anyanwu, “Scheduling Hadoop Jobs to Meet Deadlines,” in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 388–392.

[25] P. Nguyen, T. Simon, M. Halem, D. Chapman, and Q. Le, “A Hybrid Scheduling Algorithm for Data Intensive Workloads in a MapReduce Environment,” in *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, 2012, pp. 161–167.

[26] Z. Guo and G. Fox, “Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization,” in *CCGRID*, 2012, pp. 714–716.

[27] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, “SAMR: A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment,” in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, 2010, pp. 2736–2743.

[28] X. Sun, C. He, and Y. Lu, “ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm,” in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, 2012, pp. 148–155.

[29] A. Rasooli and D. G. Down, “A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems,” in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 1284–1291.

[30] A. Rasooli and D. G. Down, “An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems,” in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, 2011, pp. 30–44.

[31] K. A. Kumar, V. K. Konishetty, K. Voruganti, and G. V. P. Rao, “CASH: Context Aware Scheduler for Hadoop,” in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 2012, pp. 52–61.

[32] I. Polato, R. Re, A. Goldman, and F. Kon, “A comprehensive view of Hadoop research. A systematic literature review,” *J. Netw. Comput. Appl.*, vol. 46, no. 0, pp. 1–25, 2014.

[33] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in *Proceedings of the 5th European Conference on Computer Systems*, 2010, pp. 265–278.

[34] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE: Adaptive Data Replication for Efficient Cluster Scheduling," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, 2011, pp. 159–168.

[35] C. He, Y. Lu, and D. Swanson, "Matchmaking: A New MapReduce Scheduling Technique," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 40–47.

[36] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009, pp. 1–8.

[37] X. Zhang, Z. Zhong, S. Feng, B. Tu, and J. Fan, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments," in *Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, 2011, pp. 120–126.

[38] S. Babu, "Towards automatic optimization of MapReduce programs," *Proc. 1st ACM Symp. Cloud Comput. - SoCC '10*, p. 137, Jun. 2010.

[39] C. Li, H. Zhuang, K. Lu, M. Sun, J. Zhou, D. Dai, and X. Zhou, "An Adaptive Auto-configuration Tool for Hadoop," in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, 2014, pp. 69–72.

[40] "Hadoop Performance Tuning." [Online]. Available: https://hadoop-toolkit.googlecode.com/files/White paper-HadoopPerformanceTuning.pdf. [Accessed: 23-Feb-2015].

[41] "7 tips for Improving MapReduce Performance," 2009. [Online]. Available: http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/. [Accessed: 20-Feb-2015].

[42] J. Yan, X. Yang, R. Gu, C. Yuan, and Y. Huang, "Performance Optimization for Short MapReduce Job Execution in Hadoop," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*, 2012, pp. 688–694.

[43] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, “SHadoop: Improving MapReduce Performance by Optimizing Job Execution Mechanism in Hadoop Clusters,” *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2166–2179, Mar. 2014.

[44] K. Elmeleegy, “Piranha: Optimizing Short Jobs in Hadoop,” *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 985–996, Aug. 2013.

[45] B. Palanisamy, A. Singh, and B. Langston, “Cura: A Cost-Optimized Model for MapReduce in a Cloud,” in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 2013, pp. 1275–1286.

[46] J. Virajith, B. Hitesh, C. Paolo, K. Thomas, and R. Antony, “Bazaar: Enabling Predictable Performance in Datacenters,” 2012.

[47] K. Kambatla, A. Pathak, and H. Pucha, “Towards Optimizing Hadoop Provisioning in the Cloud,” in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, 2009.

[48] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, “Starfish: A Self-tuning System for Big Data Analytics,” in *In CIDR*, 2011, pp. 261–272.

[49] G. Liao, K. Datta, and T. L. Willke, “Gunther: Search-based Auto-tuning of Mapreduce,” in *Proceedings of the 19th International Conference on Parallel Processing*, 2013, pp. 406–419.

[50] D. Wu and A. S. Gokhale, “A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration,” in *20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013*, 2013, pp. 89–98.

[51] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir, “Panacea: Towards Holistic Optimization of MapReduce Applications,” in *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, 2012, pp. 33–43.

[52] “Apache Pig,” *Apache Hadoop*. [Online]. Available: http://pig.apache.org/. [Accessed: 24-Feb-2015].

[53] “Apache Hive,” *wikipedia*. [Online]. Available: http://en.wikipedia.org/wiki/Apache_Hive. [Accessed: 24-Feb-2015].

[54] “Google BigTable,” *wikipedia.org*. [Online]. Available: http://en.wikipedia.org/wiki/BigTable. [Accessed: 24-Feb-2015].

[55] “Apache HBase,” *Apache HBase*. [Online]. Available: http://hbase.apache.org/. [Accessed: 24-Feb-2015].

[56] “ZooKeeper,” *Apache Hadoop*. [Online]. Available: http://zookeeper.apache.org/doc/trunk/. [Accessed: 24-Feb-2015].

[57] “Sqoop,” *Apache Software Foundation*. [Online]. Available: http://sqoop.apache.org/. [Accessed: 25-Feb-2015].

[58] “Apache Flume,” *Apache Flume*. [Online]. Available: http://flume.apache.org/. [Accessed: 25-Feb-2015].

[59] “Hadoop releases,” *Apache Hadoop*. [Online]. Available: http://hadoop.apache.org/releases.html. [Accessed: 25-Feb-2015].

[60] D. Sullivan, “Getting Start with Hadoop 2.0,” *Apache Hadoop*, 2014. [Online]. Available: http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html. [Accessed: 25-Feb-2015].

[61] M. Khan, Y. Liu and M. Li, “Data locality in Hadoop cluster systems,” in *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on*, 2014, pp. 720-724

Chapter 3

Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data

Phasor measurement units (PMUs) are being rapidly deployed in power grids due to their high sampling rates and synchronized measurements. The devices high data reporting rates present major computational challenges in the requirement to process potentially massive volumes of data, in addition to new issues surrounding data storage. Fast algorithms capable of processing massive volumes of data are now required in the field of power systems. This chapter presents a novel parallel detrended fluctuation analysis (PDFA) approach for fast event detection on massive volumes of PMU data, taking advantage of a cluster computing platform. The PDFA algorithm is evaluated using data from installed PMUs on the transmission system of Great Britain from the aspects of speedup, scalability, and accuracy. The speedup of the PDFA in computation is initially analyzed through Amdahl’s Law. A revision to the law is then proposed, suggesting enhancements to its capability to analyze the performance gain in computation when parallelizing data intensive applications in a cluster computing environment.

3.1 Introduction

Security in power systems is so vital that major efforts must be taken in order to avert potential power system blackout scenarios. The blackout in North East America on the 14th August 2003 and other critical grid events all over the world are driving the industry to develop more automatic, adaptive and efficient computational tools for power system monitoring and stability analysis. It is becoming highly impractical for traditional supervisory control and data acquisition (SCADA) systems to predict or avert eventualities in a timely manner that may lead to power system catastrophes [1]–[3].

One solution to these challenges is presented in the ongoing development of wide area monitoring systems (WAMS). WAMS comprise a network of synchronized phasor measurement units (PMUs) [1], [4], which provide data at sampling rates typically equivalent to one cycle of the power systems fundamental frequency (50 Hz on the Great Britain (GB) system). This data, if efficiently managed and processed, can be used to enhance the reliability, stability and security of power systems. For these reasons PMUs are being deployed in power systems globally, resulting in rapidly growing volumes of data, posing network operators with new challenges in terms of data storage and timely analysis of the potentially massive datasets.

As a result of the growing complexities in power systems from the increased integration of renewable generation sources and the networks ongoing expansions, it is now vital that data surrounding power system events, such as generation losses, are accurately captured. These events provide the only reliable source of information on the true power system dynamics, providing greater understanding of system inertia, something that is of growing concern on the power system of GB. Timely analysis of these events is critical to understanding the necessary generation response and reserve requirements for a secure network [5]. They also permit the analysis of any trends in the behavior of the power system under different operating conditions and provide means to validate or improve offline system modeling tools.

A number of research works have been proposed for the detection of system events with PMU data. The work described in [6] details an approach based on finite impulse response (FIR) filtering that is concerned with detecting transient power system events, as a means of determining steady-state information from PMUs to improve situational awareness. Whereas, the work presented in [7] uses a generator clustering approach to determine the source of an event based on detecting the largest initial rotor swing. Other works have dealt with screening volumes of data for significant events, applying algorithms based on Fourier transforms and Yule Walker methods [8], [9].

In this chapter the design and implementation of a parallel detrended fluctuation analysis (PDFA) algorithm, for fast event detection on massive volumes of PMU data, is presented. The approach is implemented in the MapReduce programming model [10], which has become a major software technology in the support of data intensive applications, making use of a cluster of inexpensive commodity computers. The work develops some of the authors' previous studies [11], on the use of detrended fluctuation analysis (DFA) for the detection of power system events on small datasets, more specifically for the detection of instantaneous generation losses, as a requirement for power system inertia estimation [5]. In contrast with previous works, the methodology presented in this chapter is focused with determining the exact instant a specific event starts, so that the event can be isolated for additional analysis. Flagging the presence of an event is intended, in the online sense, to act as a trigger for the running of steady-state estimators [11].

The PDFA is tested and demonstrated in two stages, the first providing details of a laboratory based online setup, using a PMU installed at the domestic supply and the openPDC platform [12] with a localized Data Historian (DH) to collect and store 50 Hz resolution data. The second, details the application to the WAMS installed on the transmission system of GB, whereby an offline data mining approach is demonstrated. The performance of the PDFA is compared with the original sequential DFA in terms of efficiency and accuracy, using PMU data from the GB WAMS. The speedup of the PDFA in computation is analyzed with Amdahl's Law, and based on this analysis, a revision to Amdahl's Law is then proposed. The revision aims to enhance the capability of analyzing the performance gain in computation when parallelizing data intensive applications in cluster computing environments.

3.2 Over View of HPC and Big Data Analytics

With the advent of the smart grid the power system is becoming increasingly complex and computationally intensive. The power systems community faces the challenge of finding suitable methods to solve growing computational issues, for instance, processing massive volumes of PMU data. Such methods can be found in the field of high performance computing (HPC) through parallel processing.

The message passing interface (MPI) is a parallel programming model used to parallelize computation across multiple processors or computers. The MPI model has been used to distribute computation tasks over grid computing nodes [13] and in [14] it was deployed in the HPC environment to parallelize a contingency analysis algorithm. However, the MPI model still requires improvement in areas such as parallel I/O, scalability, fault-tolerance and topology awareness [15]. It is worth noting that the MPI forum added the advanced feature of dynamic process management to MPI version 2.0, with the intention to dynamically add or remove the processes when running MPI jobs. However, the existing fault-tolerance capabilities are not the property of the MPI but of the program that couples within the MPI implementation [16]. The latest version of MPI (3.0) does not currently have fault-tolerance capabilities, rather it is proposed for future versions [17], [18].

An alternative approach can be found in cluster computing. In [19] a High-Performance Hybrid Computing approach was applied to reduce the execution time of massive contingency analysis algorithms. In [19] the algorithm was parallelized using a XMT multithread C/C++ compiler on Gray XMT (multithread HPC computing platform) and conventional cluster computers. In addition, the work in [20] proposed a large scale smart grid stability monitoring application using a conventional cluster of computers to speed up the analysis of PMU measurements. These two separate approaches can increase the speed of program execution by adding more processing nodes however, they rely on centralized management, which can be vulnerable to node failure.

Gao and Chen [21] used the parallel computing toolbox within MATLABs Distributed Computer Server (MDCS) to parallelize their contingency analysis algorithm on multiple processors, whilst in [22] a parallel processing method for two monitoring techniques in Prony analysis and an extended complex Kalman filter on multicore systems is explored. Similarly in [23] a genetic algorithm was parallelized. However, these approaches are not resilient and fault-tolerant. The aforementioned approaches can significantly reduce the execution time of large complex computation however, applying these approaches in power system applications is not simply a case of adding more processing units, they require careful design of programs and middleware to make the applications compatible with underlying hardware and software. Furthermore, these approaches (cluster and MPI based) can be scaled by adding more processing nodes. However, they lack the ability to respond to node failures. For example, if any processing node fails as a result of a hardware or software problem, they do not have any remedy to migrate the running tasks to another available node.

Alternatively the work in [22] and [24] proposes the cloud computing platform for smart grid data storage and real-time analysis. They parallelize the processing in cloud computing environments to achieve faster computation. To reduce the risk of data accessibility during node failures, data is replicated on multiple machines however, in the instance of node failures no solution is provided to gracefully assign the running computation to another node.

A solution to these issues can be found in the Hadoop MapReduce framework, proposed in a number of areas [25]–[28], offering a reliable, fault-tolerant, scalable and resilient framework for storing and processing massive datasets. In [25] a machine learning technique is applied whilst in [26] simple statistic calculations (maximum, minimum, and average) are used to process PMU datasets. However, both of these works leave out the implementation details and provide no evaluation of their methodology or results. The work in [27] and [28] uses the Hadoop distributed file system (HDFS) for storing data and Pig scripting language for simple statistical calculations. The main focus of both works is to compare the performance of the Hadoop distributed processing with the Multicore system.

3.3 Wide Area Monitoring GB System

The WAMS running on the GB National Grid is in the early stages of its deployment. Around 40 PMUs have been installed on the transmission system of England and Wales through a series of upgrades to digital fault recorders (DFRs) and the installation of four dedicated PMUs, the majority of which are configured to report back to a central phasor data concentrator (PDC) at the national control center.

The primary role of the system is to monitor for any oscillatory behavior between the generators in Scotland and those of England and Wales. An interarea mode had been previously identified at around 0.5 Hz involving all of the GB system and remains a cause for concern across a major system constraint boundary; in the two 120 km 400 kV double circuits that connect the Scottish Network with the North of England. Alarms are sent from this system in real-time to the energy management system (EMS), to alert the network operators when the system is believed to be approaching instability. This constraint is considered to hinder the transfer of future renewable generation in Scotland to the main demand centers in England and Wales.

The PDC is configured to store the 50 Hz PMU data at maximum resolution for a rolling one year period, after this time the data is to be archived off at a reduced resolution of 10 Hz for upto 10 years. With the amount of PMUs set to increase on the GB system, as additional DFRs are upgraded and new dedicated PMUs are installed [2], this represents a growing challenge in terms of data storage. In addition it is now of vital importance to capture data surrounding system events as they provide the only reliable source of information on the response of the power system, these events need to be captured at full resolution to assist in inertia estimation methods [5] and continuing validation of the offline network model. Due to the growing volumes of data, importance is therefore placed on timely analysis through fast algorithms and identification of such events.

In addition PMUs have also been deployed at the domestic supply at four U.K. Universities, Brunel, Birmingham, Manchester and Strathclyde. Synchrophasor data, in voltage (magnitude and phase), frequency and rate of change of frequency (RoCoF), is measured locally at 50 Hz and sent via the Internet to a server in Ljubljana, Slovenia hosted by ELPROS. This system provides good geographical visibility of the GB transmission system with PMUs well distributed across the network, providing good visibility with regard to the impact of any system events through the Anglo-Scottish connection. In addition, a laboratory setup exists at Brunel University where a PMU is configured to communicate data locally to a PDC. The server is running the openPDC software [12], designed by the Tennessee Valley Authority (TVA) and administered by the Grid Protection Alliance (GPA). The openPDC is used to collect, manage and process real-time synchrophasor measured values. This system is an example of a low cost, easy installation alternative to the larger scale WAMS solutions.

3.4 Design of PDFA

The PDFA proposed in this chapter works by detrending a dataset of PMU frequency measurements on a sample-by-sample sliding window. The window is configured to be 50 samples long, this is to detect for changes over an one second period (at 50 Hz), looking for a specific loss shape in frequency, following an instantaneous loss in generation. The loss shape typically lasts for one second, before primary response services take over and arrest the drop in frequency [5]. A root mean square (RMS) value is then taken of the fluctuation, F for every window, as shown in Eq. (3.1), this value is then compared with a threshold value, predetermined through a number of previous baseline studies, F = 0.2x10−3 to detect for the presence of an event.

$$F(n) = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left[e(k)\right]^{2}} \qquad (3.1)$$

where *n* is the size of the window (50 samples), *k* is the sample number and *e(k)* is the detrended signal.

Previous works on detrending power system data [29] have focused on removing trends or denoising power system data for the purposes of processing transient oscillations, other work [30] and the original implementation of DFA [31] have focused on the detection of long-range correlations in data series. This is all separate from the work described in this chapter. The purpose of detrending the data for this application is to highlight the specific changes in the PMUs measured values as a result of captured transients on the network; the process has the affect of filtering the normal variations in the signal that are predominantly a feature of the high resolution measurements, placing the focus on extreme changes over relatively short time spans.

The PDFA approach is the development of the DFA method to operate efficiently on massive volumes of PMU data, using MapReduce cluster computing. It is very important to note that the inherent sample by sample sliding window approach of the presented DFA method is highly disposed to HPC and Big Data Analytics.

3.4.1 MapReduce Programming Model

MapReduce is a parallel and distributed programming model originally developed by Google for processing massive amounts of data in a cluster computing environment [10], [32]. Due to its remarkable features such as fault-tolerance, simplicity and scalability, MapReduce has become a major software technology in support of data intensive applications [33]. MapReduce is a highly scalable model; thousands of commodity computers can be used as an effective platform for parallel and distributed computing.

As shown in Figure 3.1, the MapReduce model divides computational tasks into Map and Reduce stages. In the Map stage, the computation is divided into several Map tasks to be executed in parallel on cluster computing nodes or virtual machines (VMs). Each Map task (a user-define Map function) processes a block of the input dataset and produces an intermediate result (IR) in the form of key/value pairs, which are then saved in local storage. In the Reduce phase, each Reduce task (a user-define Reduce function) collects the IR and combines the values together corresponding to a single key to produce the final result. It should be noted that the Map and Reduce functions are executed independently.

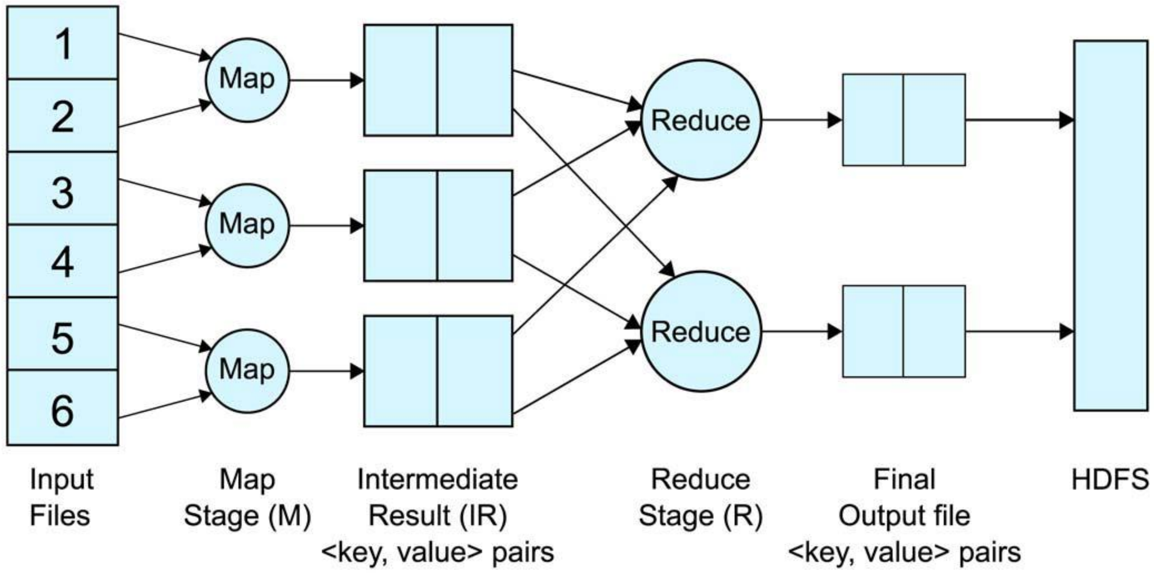
1
2
3
4
5
6
Map
Map
Map
Reduce
Reduce
Input Files
Map Stage (M)
Intermediate Result (IR) <key, value> pairs
Reduce Stage (R)
Final Output file <key, value> pairs
HDFS

Figure 3.1: MapReduce model

3.4.2 MapReduce Implementation with Hadoop

The MapReduce programming model has been implemented in a number of systems such as Mars [34], Phoenix [35], Dryad [36], and Hadoop [37]. Hadoop is the most popular implementation of MapReduce and has been widely employed by the community due to its open source nature. Hadoop was originally developed by Yahoo to process huge amounts of data (over 300 TB) across a cluster of low-cost commodity computers [38]. It is worth noting that Hadoop not only works in cluster computing environments, but also in cloud computing systems such as the Amazon EC2 Cloud [39].

The architecture of the Hadoop framework, as shown in Figure 3.2, comprises its own file system, HDFS [40]. HDFS is designed to store massive amounts of data (terabytes or petabytes) over a large number of computer clusters and provides fast, scalable access to data. HDFS follows a client server architecture, where there is a Name Node acting as the server and multiple Data Nodes that act as clients. The HDFS has high availability (HA) features by providing the option to configure two Name Nodes in the same cluster in the form of active Name Node or passive Name Node (Standby Name Node). This feature is used to reduce the risk of single points of failure. The passive Name Node deals with fast failover in case the active Name Node crashes as a result of software or hardware malfunction [41].

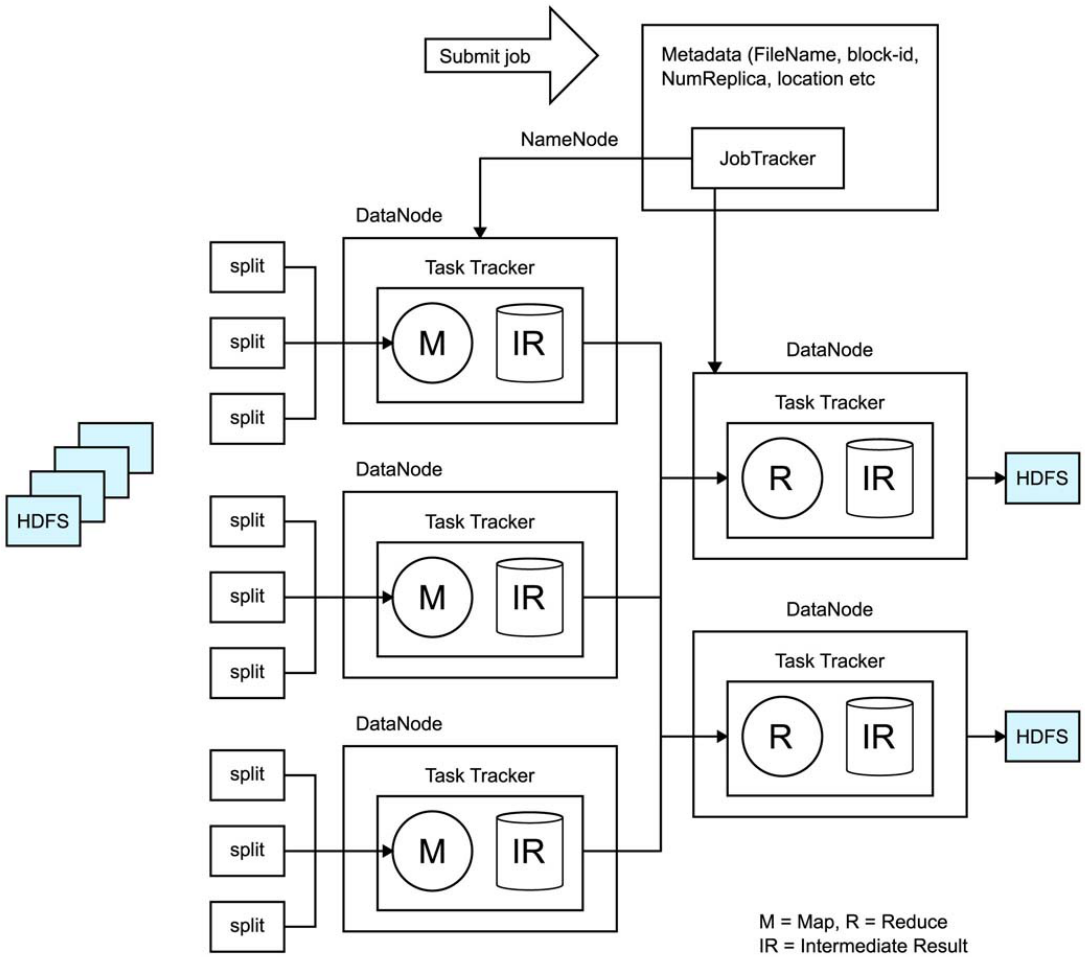

Figure 3.2: Hadoop framework.

HDFS automatically splits input files into equal size blocks (64 MB or 128 MB by default) that are distributed across the Data Nodes. Each data block has multiple replicas (3 by default), which are stored on different data nodes. If the cluster network topology has more than one rack then the block replicas will be stored on different rack machines. The purpose of data replication and distribution on different machines is to maximize reliability and availability of data.

The Name Node manages the namespace of the file system and regulates the client's access to files. It does not store data itself, but rather maintains metadata files that contain information such as file name, block id, number of replicas, mapping between blocks and Data Nodes on which the blocks are stored and the location of each block replica. The Data Nodes manage the storage directly attached to each Data Node and execute Map and Reduce tasks.

The Job Tracker runs on the Name Node and is responsible for dividing user jobs into multiple tasks, scheduling the tasks on the Data Nodes, monitoring the tasks and reassigning the tasks in the instance of a failure. The Task Tracker runs on Data Nodes, receiving the Map and Reduce tasks from the Job Tracker and periodically contacts with the Job Tracker to report the task completion progress and requests for new tasks.

3.4.3 PDFA Implementation

The original DFA was implemented in MATLAB specifically for the offline application of event detection, focusing on small datasets and the determination of the t = t0 moment or exact start time of a specific event.

The PDFA, as described in this chapter, is intended for the analysis of massive volumes of PMU data. It was implemented in the Hadoop MapReduce framework using the Python programming language due to its flexibility and open source. The algorithm was implemented, as depicted in Figure 3.3, through the following two staged data collection approaches.

1) Online Data Collection: The laboratory based setup at Brunel University comprises a domestic supply connected PMU measuring positive sequence voltage values, frequency and RoCoF. This data is sent through a local area network (LAN) to an openPDC historian. The openPDC software is configured in such a way that when the historian data size reaches 100 MB, a new data storage file is created in .d format with a corresponding time-stamp.

A data agent has been created in the Java programming language using a number of Hadoop core libraries and the Java directory watch service package. The application code is encapsulated in the while loop statement to execute continuously, monitoring the historian folder to detect for the presence of new .d files. Once the new file is created in the historian folder, the data agent application automatically moves it to the Hadoop cluster HDFS storage.

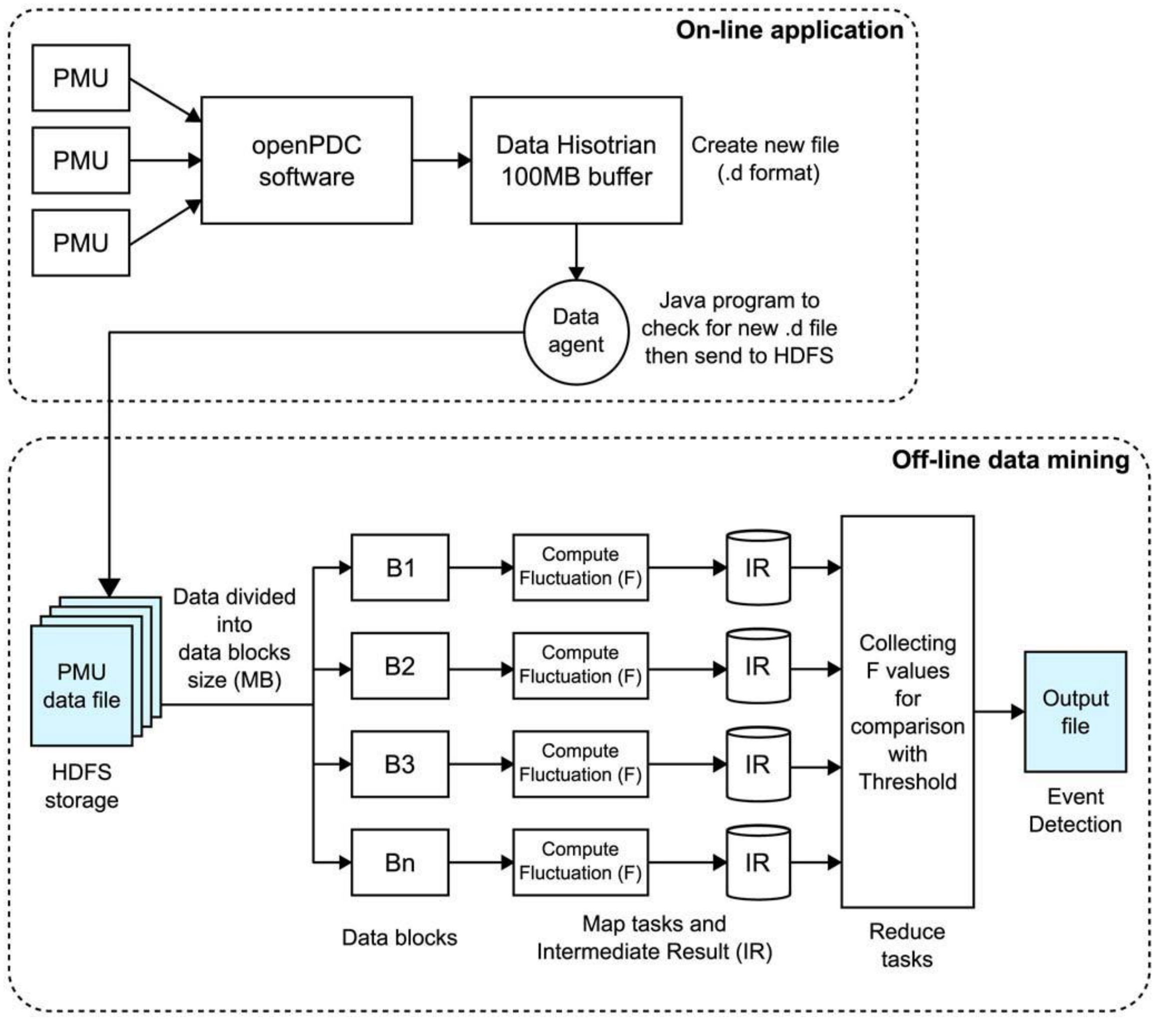
On-line application
PMU
PMU
PMU
openPDC software
Data Hisotrian 100MB buffer
Create new file (.d format)
Data agent
Java program to check for new .d file then send to HDFS
Off-line data mining
PMU data file
HDFS storage
Data divided into data blocks size (MB)
B1
B2
B3
Bn
Data blocks
Compute Fluctuation (F)
IR
Map tasks and Intermediate Result (IR)
Collecting F values for comparison with Threshold
Reduce tasks
Output file
Event Detection

Figure 3.3: Architecture of PDFA implementation.

2) Offline Analysis—Big Data: Having proven the online data collection side of the system, the following analysis can either be performed as a complement to this process or alternatively it can work in a Data Mining sense where massive datasets are provided directly to the HDFS storage. It should be noted that the HDFS storage system is not capable of working with the .d file format provided during the online data collection process. At present this file is manually converted to .csv format offline using the historian playback module within the openPDC software. This process will be automated at a later stage as part of further work, to allow the entire process to be carried out online in near real-time.

The Hadoop MapReduce supports a number of programming languages such as Java, Python, and C++. Java is the native language of Hadoop and so programs written in Java can be directly executed. Programs written in any other language require application program interfaces (APIs) to execute. For example, programs written in C++ are executed through the Pipes API and programs written in Python will execute through the Streaming API [42].

PDFA has been written in Python in the form of Map and Reduce functions, as Python is open source and unlike Java contains a large amount of the required mathematical functionality. The PDFA is then executed through the Streaming API in the Hadoop MapReduce environment.

When a dataset is moved onto a Hadoop cluster, the HDFS automatically divides it up into blocks B, shown in Figure 3.3. The block size is specified in the cluster configuration file (hdfs-site. xml), for instance, if a historian dataset is 16 MB and the block size value has been set to 2 MB, then the total number of blocks for that dataset will be 8 (16/2 = 8). The total number of Map tasks is equal to the total number of blocks.

When the PDFA program is submitted to the Hadoop framework, the framework automatically divides the PDFA program into a number of Map and Reduce tasks. A block of the PMU dataset is assigned to each Map task and the number of Map tasks executed in parallel to process the dataset depends upon the number of Map slots specified in the cluster configuration file (mapred-site.xml). For the PDFA, one slot was configured on each VM, as a result eight Map slots were configured in the cluster and so eight Map tasks were executed in parallel to process the historian dataset. The number of Map slots configured in a VM depends on the processing capacity (physical memory and number of CPU cores) of the VM.

Each Map task processes the assigned data block on a sliding window of 50 samples (as per the DFA algorithm) and calculates the fluctuation value F. The F values are buffered in memory of size 100 MB, which can also be set in the configuration file. When the content of the buffer memory reaches a threshold value of 80% (80 MB) a background thread is started to spill the contents of the memory buffer to a local disk as an intermediate result (IR). The number of IR files is equal to the number of Reduce tasks.

After completion of the Map phase, the PDFA Reduce tasks are initiated and collect the calculated F values. The number of Reduce tasks is also configurable by the user in the configuration file. The number of Reduce tasks to be executed in parallel depends on the number of Reduce slots configured in the configuration file. For the PDFA, eight Reduce tasks and eight Reduce slots were configured, so as to fully utilize all the available Reduce slots. Each Reduce task compares every value of F with the threshold value F = 0.2x10−3, any value greater than this threshold is flagged as an event for further analysis.

Most of the conventional cluster-based approaches have issues of reliability and fault-tolerance. The PDFA is implemented in a Hadoop based cluster computing environment, as it offers built-in remarkable features such as high availability, fault-tolerance and scalability. The framework supports multiple replicas of the data blocks and distributes them on different computers/VMs to overcome any fail situations and delays. The cluster can easily be scaled by adding more processing nodes to increase the speedup of computation. During the job execution, if any processing nodes crash due to software or hardware failures, the Job Tracker will automatically detect it and assign the running tasks to another available node.

3.5 Evaluation and Experiment Results

We have compared the performance of the PDFA with that of the sequential DFA from the aspects of both efficiency in computation and accuracy. The performance was evaluated using 6000 samples of frequency data (2 min at 50 Hz), provided by National Grid. The data contained a known system event, in the loss of a generator exporting approximately 1000 MW. In order to create a Big Data scenario, this dataset was replicated a number of times to provide a relatively large dataset with over 32 million samples.

3.5.1 Experiment Setup

The experiments were carried out using a high performance Intel Server machine comprising four Intel Nehalem-EX processors running at 2.27 GHz each with 128 GB of physical memory. Each processor has ten CPU cores with hyper thread technology enabled in each core. The specific details of the hardware and software implementation are displayed in Table 3.I. The analysis of the sequential DFA was carried out on just one of the VMs, whereas the PDFA was run on upto 8 VMs.

Table 3.1: Experimental configuration of Hadoop cluster.

Hardware	CPU	40 Cores
	Processor	2.27GHz
	Storage	2TB and 320GB
	Connectivity	100Mbps Ethernet LAN
Software	Operating System	Ubuntu 12.04 TLS
	Python	Version 3.3
	JDK	Version 1.6
	Hadoop	CDH 4.5
	Oracle Virtual Box	Version 4.2.8
	OpenPDC	Version 1.5

3.5.2 Results

A number of experiments were carried out to evaluate the efficiency and accuracy of the PDFA method. From Figure 3.4, it can be seen that the PDFA outperforms the sequential DFA in computation significantly using 8 VMs. The execution time of the sequential DFA increases with an increasing number of data samples, while the execution time of the PDFA remains relatively constant.

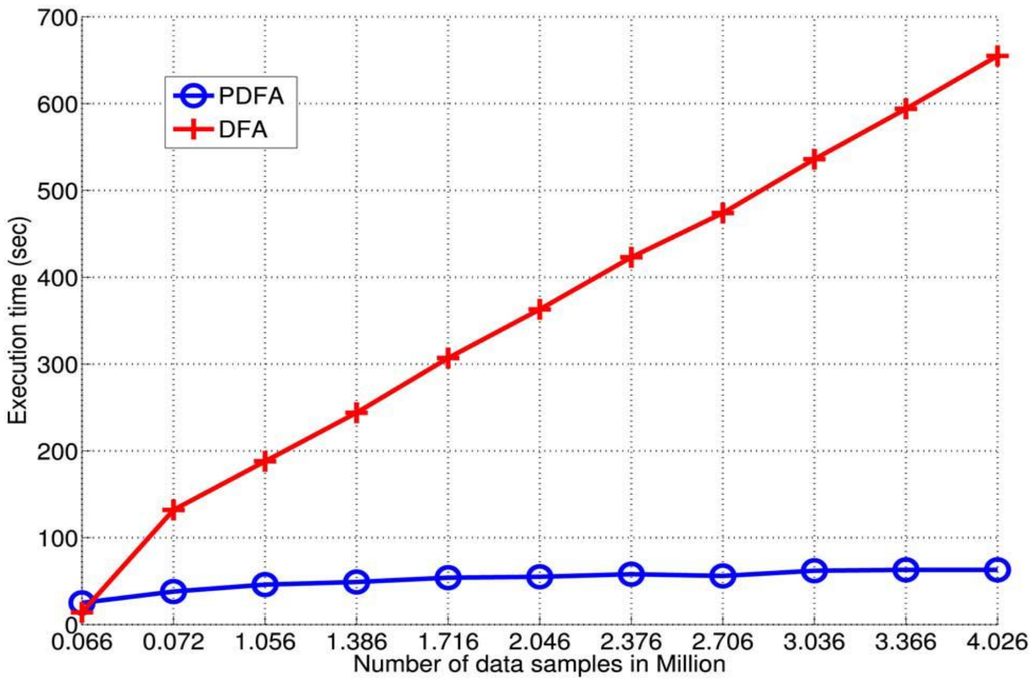

Figure 3.4: Analysis of PDFA efficiency

The DFA algorithm works on a sliding window, so when comparing the output of the sequential DFA with PDFA it is important to note the possibility of discrepancies in results caused by data portioning due to the way in which the datasets are divided up for parallelization. This does not affect the PDFA ability to detect events; it just means that the F values could differ slightly from the DFA results. The results of the PDFA are compared with that of the DFA and are displayed in Figure 3.5, the relative accuracy of PDFA is very close to that of the sequential DFA, especially in the cases of larger datasets, as the difference converges to zero.

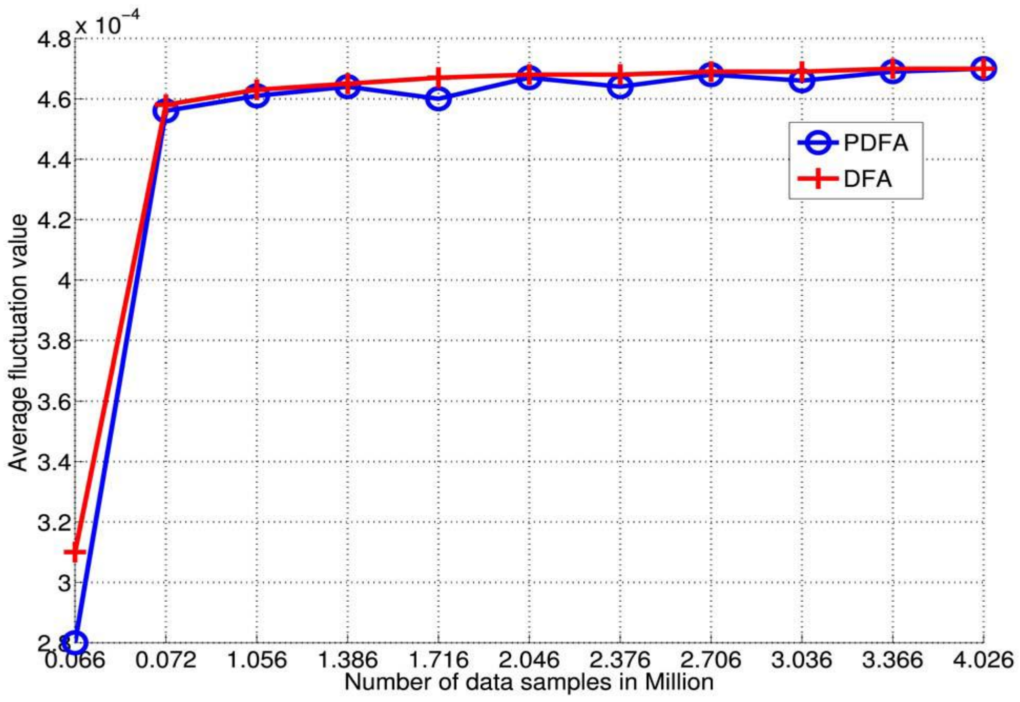

Figure 3.5: Relative accuracy of PDFA compared to DFA.

The scalability of the PDFA in terms of a varied number of both VMs and data samples was evaluated. Figure 3.6 shows the execution times of the PDFA when processing three different sizes of dataset and a varied number of VMs from 1 to 8. The PDFA clearly performs best in scalability on the largest dataset with 32 million data samples. It can be observed that the execution time of the PDFA on each dataset decreases with an increasing number of VMs employed. When processing 8 M data samples, 4 VMs generated 2 times speedup, whereas 8 VMs generated 2.5 times of speedup. However, when the number of data samples is increased to 32 M, 4 VMs generated 3.3 times of speedup whereas 8 VMs generated 5.4 times of speedup. With increasing numbers of data samples, the times of speedup will be increased closer to the number of VMs.

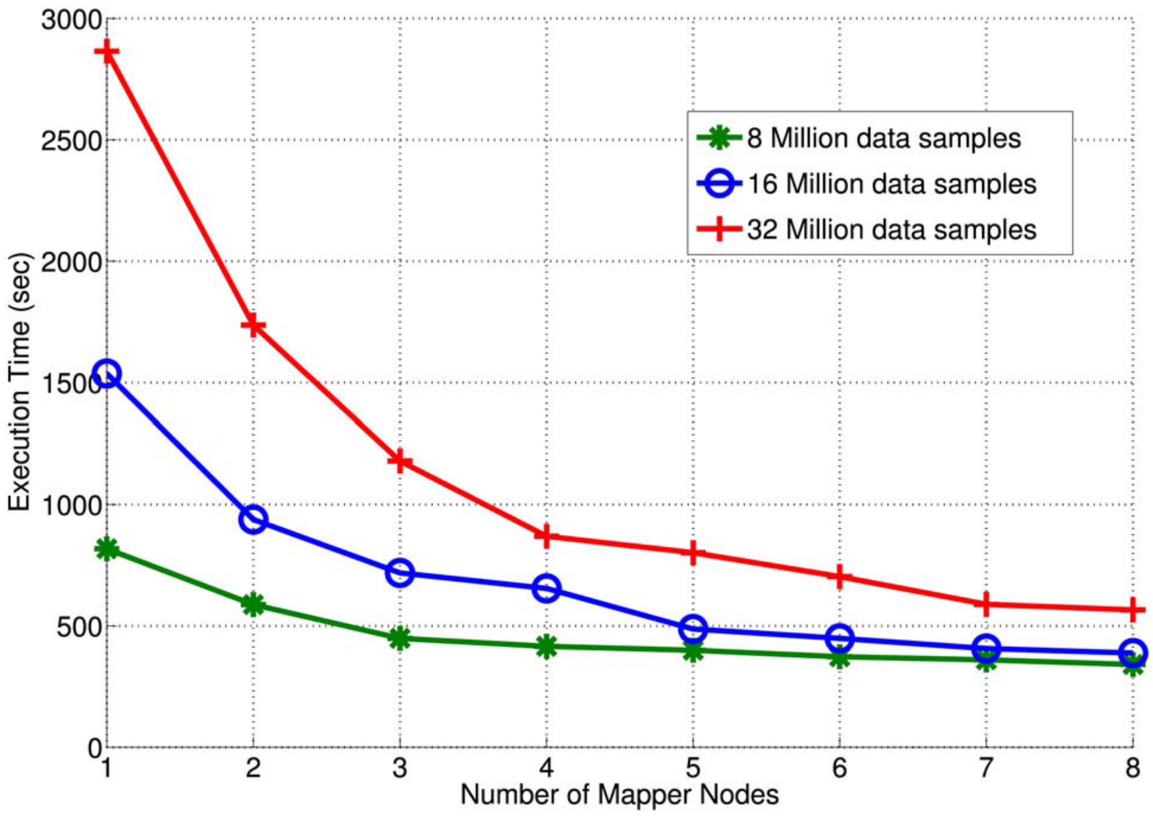

Figure 3.6: Scalability of PDFA, execution time against number of mapper nodes (VMs)

Based on the results presented in the Figure 3.6, the speedup of the PDFA in terms of computation when processing the three different sized datasets was calculated using

$$Speedup = \frac{T_S}{T_N} \tag{3.2}$$

Where, T_S is the execution time of the PDFA on a single VM and T_N represents the execution time of the PDFA on N number of VMs. The results of this calculation are displayed in Figure 3.7. Again, the PDFA achieves the best speedup in computation on the largest dataset with 32 million data samples. However, as shown in the figure by the dotted line, the results never achieve that which are to be expected from Amdahl's law [43].

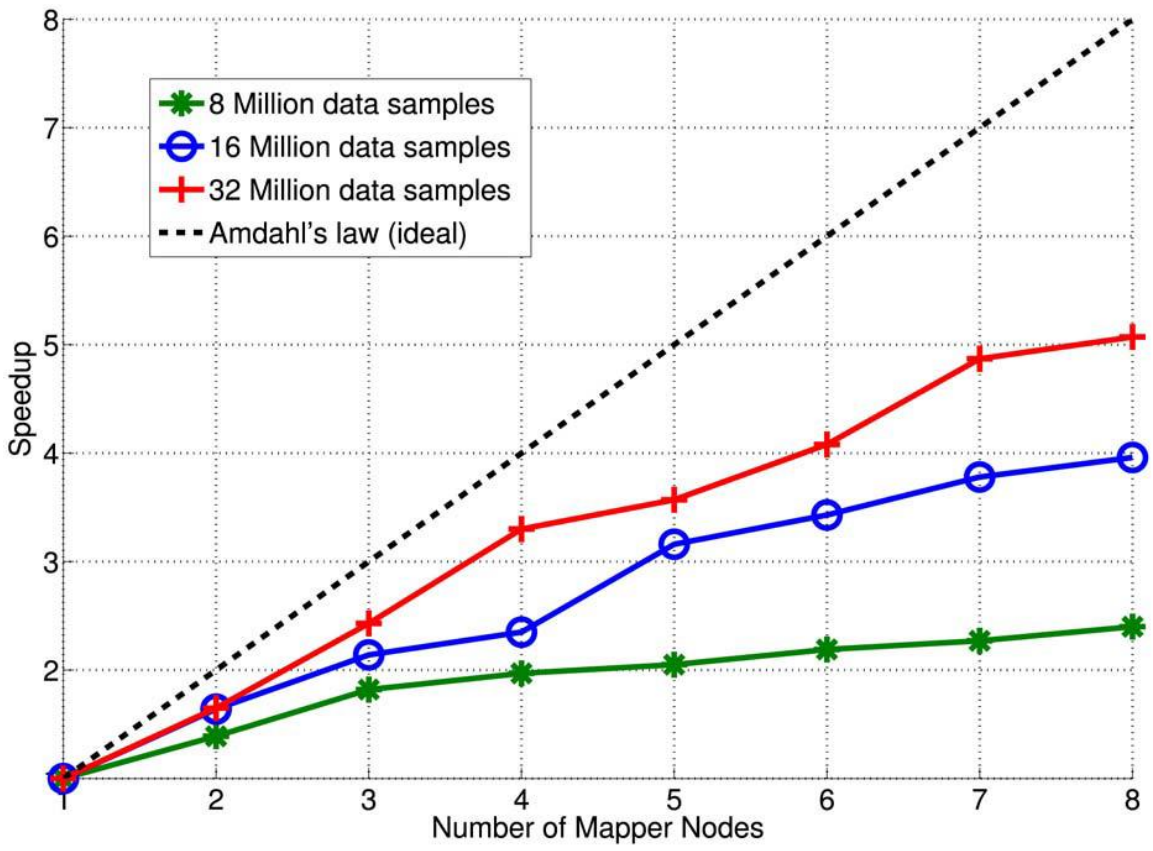

Figure 3.7: Speedup analysis of the PDFA algorithm.

3.5.3 Speedup Analysis

When parallelizing a sequential program, the speedup in computation can be calculated using Amdahl's Law [43], defined in

$$Speedup = \frac{1}{(1-P)+\frac{P}{N}} \tag{3.3}$$

where P, represents the portion of the sequential program in percentage that can be parallelized and N represents the number of computers used in the computation.

Theoretically, in the case when a sequential program can be fully parallelized (P = 1), as was the case with PDFA, the speedup of the parallelized program should be equal to the number of computers used in the computation N. Therefore, we have

$$Speedup = \frac{1}{(1-P)+\frac{P}{N}} \leq N \tag{3.4}$$

However, as shown in Figure 3.7, the closest speedup to Eq.(3.4) that the PDFA achieved in all the computation scenarios was 3.3 times faster than the sequential DFA when 4 VMs were used in the process. The speedup of the PDFA never achieved N times in a Hadoop cluster with N computers even though the sequential DFA was fully parallelized. This means that Amdahl's Law in the form of Eq. (3.4) is not sufficient in calculating the speedup of a parallelized program that is executed in a cluster computing environment. This is because Amdahl's Law in this form does not consider the communication overhead of a user job in cluster computing. For this purpose, a revision to Amdahl's Law is proposed in the form of Eq. (3.5), to better reflect the speedup gain when parallelizing a sequential program in cluster computing.

$$Speedup = \frac{1}{(1-P)+\frac{P}{N}+R} < N \tag{3.5}$$

where R, represents the ratio of the communication overhead to the computation of a user job, and R > 0.

The revised Amdahl's Law Eq. (3.5) better explains the speedup of a parallel program running in cluster computing. The larger a dataset is, the higher overhead in computation will be incurred. As a result, the lower the ratio of communication to computation would be achieved, which leads to a higher speed up in computation. This well explains the speedup of the PDFA in computation when processing the three datasets with varied sizes.

To achieve an optimal performance in speedup, the ratio of communication to the communication of a parallel program should be minimized. In the case of Hadoop MapReduce clusters, the size of the segmented data blocks shall be large. On one hand, a large size of data block will generate a small number of tasks that incurs a small overhead in communication. On the other hand, a large size of data block will lead to a high workload in computation. Therefore, a large size of data block will lead to a low communication to computation ratio generating a high speedup.

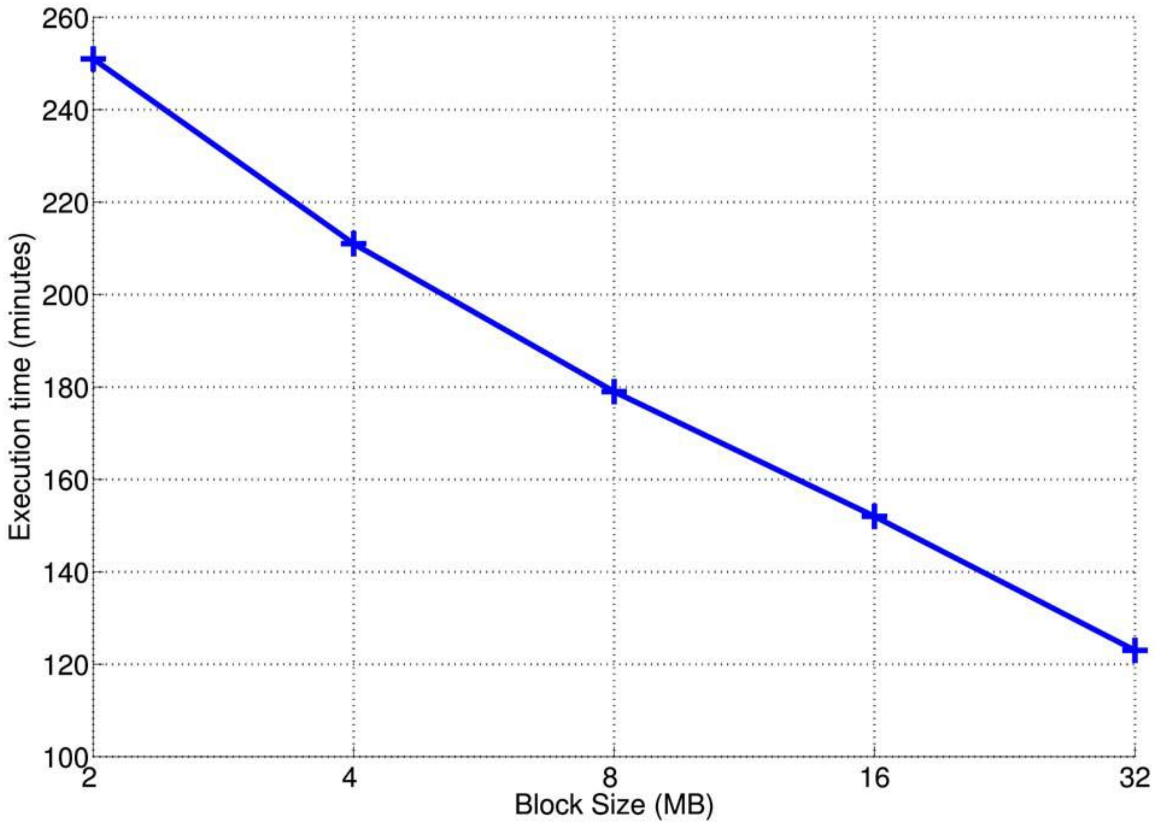

Figure 3.8: Computational overhead of PDFA against data block size.

To evaluate how the size of a data block affects the computational performance of PDFA, the algorithm was run on a dataset of 352 MB using 8 VMs with varied sizes of data blocks ranging from 2 to 32 MB. From Figure 3.8 it can be observed that the execution time of PDFA decreases with an increasing size of data block.

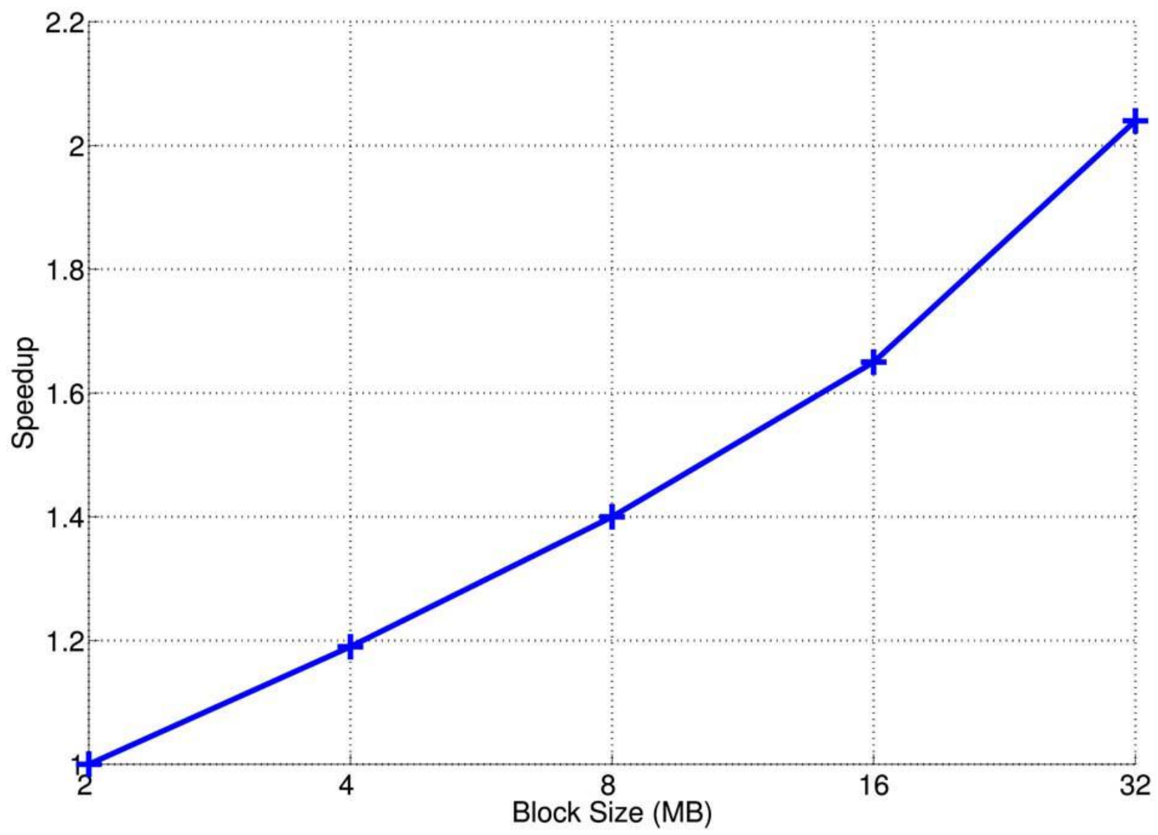

Figure 3.9: Speedup of PDFA against data block size.

The speedup of PDFA in computation goes up with an increasing size of data block, as shown in Figure 3.9. It can be seen that PDFA is 2.04 times faster in computation using 32 MB data blocks than when using 2 MB data blocks, thus confirming a greater improvement in performance with larger block size.

3.6 Summary

This chapter presented a novel PDFA approach, for fast events detection on massive PMU datasets. The PDFA was implemented in two stages, in the first stage, it was implemented in the form of a laboratory based setup for online data collection. In the second stage, it was implemented as an offline approach in the context of data mining. It was built on Hadoop model for data partitioning and distribution amongst a cluster of computer nodes. The performance of the PDFA was evaluated from aspect of scalability, speedup and accuracy. The experimental results have shown the scalability and the speedup of the PDFA in computation whilst maintained relative accuracy in comparison with the sequential DFA. Moreover, the speedup of the PDFA was evaluated through Amdahl's Law and based on the analysis in the speedup of computation, an improvement to Amdahl's law was proposed, introducing the ratio of communication to computation to enhance its capability to analyze the performance gain in computation when parallelizing data intensive applications in a cluster computing environment.

Reference

[1] M. Zima, M. Larson, P. Korba, C. Rehtanz, and G. Andersson, "Design aspect for wide-area monitoring and control system," Proc. IEEE, vol. 93, no. 5, pp. 980–996, May 2005.

[2] P. M. Ashton, G. A. Taylor, M. R. Irving, A. M. Carter, and M. E. Bradley, "Prospective wide area monitoring of the Great Britain transmission system using phasor measurement units," in Proc. IEEE Power Eng. Soc. Gen. Meeting, San Diego, CA, USA, Jul. 2012.

[3] J. F. Hauer, N. B. Bhatt, K. Shah, and S. Kolluri, "Performance of 'WAMS East' in providing dynamic information for the North East blackout of August 14, 2003," in Proc. IEEE Power Eng. Soc. Gen. Meeting, Denver, CO, USA, Jul. 2004, pp. 1685–1690.

[4] M. Rihan, M. Ahmed, and M. S. Beg, "Phasor measurement units in the Indian smart grid," in Proc. IEEE Conf. Innov. Smart Grid Technol. (ISGT) India, Kollam, India, Dec. 2011, pp. 261–267.

[5] P. M. Ashton, G. A. Taylor, A. Carter, and W. Hung, "Application of phasor measurement units to estimate power system inertial frequency response," in Proc. IEEE Power Eng. Soc. Gen. Meeting, Vancouver, BC, Canada, Jul. 2013, pp. 1–5.

[6] J. E. Tate, "Event detection and visualization based on phasor measurement units for improved situational awareness," Ph.D. dissertation, Dept.Elect. Comput. Eng., Univ. Illinois, Urbana-Champaign, IL, USA, 2008.

[7] K. Mei, S. M. Rovnyak, and C. Ong, "Clustering-Based Dynamic Event Location Using Wide-Area Phasor Measurements ," IEEE Trans. Power Syst., vol. 23, no. 2, pp. 673–679, May 2008.

[8] S. Sohn, A. J. Allen, S. Kulkarni, W. M. Grady, and S. Santoso, "Event detection method for PMUs synchrophasor data," in Proc. IEEE Conf. Power Electron. Mach. Wind Appl. (PEMWA), Denver, CO, USA, Jul. 2012, pp. 1–7.

[9] A. J. Allen, S. Sohn, S. Santoso, and W. M. Grady, "Algorithm for screening PMU data for power system events," in Proc. IEEE Int. Conf. Innov. Smart Grid Technol., Berlin, Germany, Oct. 2012, pp. 1–6.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large cluster," Commun. ACM, vol. 51, no. 1, pp. 107–133, Jan. 2008.

[11] P. M. Ashton et al., "Novel application of detrended fluctuation analysis for state estimation using synchrophasor measurements," IEEE Trans. Power Syst., vol. 28, no. 2, pp. 1930–1938, May 2013.

[12] (2013, Apr.). OpenPDC [Online]. Available: http://openpdc.codeplex.com.

[13] W. Xingzhi, Y. Zhen, and L. Li, "A grid computing based approach for the power system dynamic security assessment," J. Comput. Elect. Eng., vol. 36, no. 3, pp. 553–564, May 2010.

[14] G. A. Ezhilarasi and K. S. Swarup, "Parallel contingency analysis in a high performance computing environment," in Proc. Int. Conf. Power Syst. (ICPS), Kharagpur, India, Jul. 2009, pp. 1–6.

[15] E. Gabriel et al., "Open MPI: Goals, concept, and design of a next generation MPI implementation," in Proc. 11th Eur. PVM/MPI Users' Group Meeting, Budapest, Hungary, Sep. 2004, pp. 97–104.

[16] G. William and L. Ewing, "Fault tolerance in message passing interface programs," Int. J. High Perform. Comput. Appl., vol. 18, no. 3, pp. 363–372, Aug. 2004.

[17] B. Wesley, B. Aurelien, H. Thomas, B. George, and D. Jack, "Post-failure recovery of MPI communication capability: Design and rationale," Int. J. High Perform. Comput. Appl., vol. 27, no. 3, pp. 244–254, 2013.

[18] (2014, May). MPI 3.1 / 4.0 Standardization Effort [Online]. Available: http://meetings.mpi-forum.org/MPI_4.0_main_page.php.

[19] I. Gorton et al., “A high-performance hybrid computing approach to massive contingency analysis in the power grid,” in Proc. 5th IEEE Int. Conf. e-Sci., Oxford, U.K., Jul. 2009, pp. 277–283.

[20] J. Interrante and K. S. Aggour, “Applying cluster computing to enable a large-scale smart grid stability monitoring application,” in Proc. IEEE 14th Int. Conf. High Perform. Comput. Commun., Liverpool, U.K., 2012, pp. 328–335.

[21] W. Gao and X. Chen, “Distributed generation placement design and contingency analysis with parallel computing technology,” J. Comput. Elect. Eng. Elsevier, vol. 4, no. 4, pp. 347–354, Apr. 2009.

[22] J. C.-H. Peng, A. Mead, and N.-K. C. Nair, “Exploring parallel processing for wide area measurement data applications,” in Proc. IEEE Power Energy Soc. Gen. Meeting, San Diego, CA, USA, Jul. 2011, pp. 1–8.

[23] L. Wang and C. Singh, “Multi-deme parallel genetic algorithm in reliability analysis of composite power systems,” in Proc. IEEE Bucharest PowerTech Conf., Bucharest, Romania, Jul. 2009, pp. 1–6.

[24] K. Maheshwari, M. Lim, L. Wang, K. Birman, and R. Renesse, “Toward a reliable, secure, and fault tolerant smart grid state estimation in the cloud,” in Proc. IEEE PES Innov. Smart Grid Technol. (ISGT), Washington, DC, USA, Feb. 2013, pp. 1–6.

[25] P. Trachian, “Machine learning and windowed subsecond event detection on PMU data via Hadoop and the openPDC,” in Proc. IEEE Power Energy Soc. Gen. Meeting, Minneapolis, MN, USA, 2010, pp. 1–5.

[26] M. Edwards, A. Rambani, Y. Zhu, and M. T. Musavi, “Design of Hadoop-based framework for analytics of large sychnrophasor datasets,” Procedia Comput. Sci. Complex Adapt. Syst. Elsevier, vol. 12, pp. 254–258, Nov. 2012.

[27] H. Mass, H. K. Camak, F. Bach, and U. G. Kuhnapfel, “One year high rate low voltage recording device, method and results,” in Proc. IEEE Int. Workshop Appl. Meas. Power Syst. (AMPS), Aachen, Germany, Sep. 2013, pp. 68–72.

[28] F. Bach, H. K. Cakmak, H. Mass, and U. Kuehnapfel, “Power grid time series data analysis with pig on a Hadoop cluster compared to multi core system,” in Proc. IEEE 21st Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process. (PDP), Belfast, U.K., Feb. 2013, pp. 208–212.

[29] A. R. Messina, V. Vittal, G. T. Heydt, and T. J. Browne, “Nonstationary approaches to trend identification and denoising of measured power system oscillations,” IEEE Trans. Power Syst., vol. 24, no. 4, pp. 1798–1807, Nov. 2009.

[30] A. Bashan, R. Bartsch, J. W. Kantelhardt, and S. Havlin, “Comparison of detrending methods for fluctuation analysis,” Physica A, vol. 387, pp. 5080–5090, Apr. 2008.

[31] C.-K. Peng et al., “Mosaic organization of DNA nucleotides,” Phys. Rev. E, vol. 49, no. 2, pp. 1685–1689, Feb. 1994.

[32] R. Lammel, “Google’s MapReduce programming model—Revisited,” Sci. Comput. Program., vol. 70, no. 1, pp. 1–30, 2008.

[33] M. Isard et al., “Quincy: Fair scheduling for distributed computing clusters,” in Proc. 22nd Symp. Oper. Syst. Princ. (ACM SIGOPS), New York, NY, USA, 2009, pp. 261–276.

[34] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars: MapReduce framework on graphics processors,” in Proc. ACM 17th Int. Conf. Parallel Archit. Compilat. Tech., Toronto, ON, Canada, 2008, pp. 260–269.

[35] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, “Evaluating MapReduce for multi-core and multiprocessor systems,” in Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit., Feb. 2007, pp. 13–24.

[36] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed data-parallel programs from sequential building blocks,” in Proc. Eur. Conf. Comput. Syst. (EuroSys), 2007, pp. 59–72.

[37] (2013, Aug. 14). Apache Hadoop [Online]. Available: http://hadoop. apache.org

[38] Yahoo! (2013, Apr.). Yahoo! Launches World’s Largest Hadoop Production Application [Online]. Available: http://developer.yahoo.com/ blogs/hadoop

[39] (2013, Apr.). Amazon Elastic Computer Cloud [Online]. Available: http://aws.amazon.com/ec2

[40] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” in Proc. 26th IEEE Symp. Massive Storage Syst. Technol. (MSST), 2010, pp. 1–10.

[41] (2014, Mar.). An Introduction to HDFS High Availability [Online]. Available: http://www.cloudera.com/content/cloudera/en/documentation/cdh4/latest/CDH4-High-Availability-Guide/cdh4hag_topic_2_1.html.

[42] (2013, Apr.). Hadoop Streaming [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/streaming.html.

[43] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in Proc. AFIPS Spring Joint Comput. Conf., 1967, pp. 483–485.

Chapter 4

Hadoop Performance Modeling for Job Estimation and Resource Provisioning

MapReduce has become a major computing model for data intensive applications. Hadoop, an open source implementation of MapReduce, has been adopted by an increasingly growing user community. Cloud computing service providers such as Amazon EC2 Cloud offer the opportunities for Hadoop users to lease a certain amount of resources and pay for their use. However, a key challenge is that cloud service providers do not have a resource provisioning mechanism to satisfy user jobs with deadline requirements. Currently, it is solely the user's responsibility to estimate the required amount of resources for running a job in the cloud. This chapter presents a Hadoop job performance model that accurately estimates job completion time and further provisions the required amount of resources for a job to be completed within a deadline. The proposed model builds on historical job execution records and employs Locally Weighted Linear Regression (LWLR) technique to estimate the execution time of a job. Furthermore, it employs Lagrange Multipliers technique for resource provisioning to satisfy jobs with deadline requirements. The proposed model is initially evaluated on an in-house Hadoop cluster and subsequently evaluated in the Amazon EC2 Cloud. Experimental results show that the accuracy of the proposed model in job execution estimation is in the range of 94.97% and 95.51%, and jobs are completed within the required deadlines following on the resource provisioning scheme of the proposed model.

4.1 Introduction

Many organizations are continuously collecting massive amounts of datasets from various sources such as the World Wide Web, sensor networks and social networks. The ability to perform scalable and timely analytics on these unstructured datasets is a high priority task for many enterprises. It has become difficult for traditional network storage and database systems to process these continuously growing datasets. MapReduce [1], originally developed by Google, has become a major computing model in support of data intensive applications. It is a highly scalable, fault-tolerant and data parallel model that automatically distributes the data and parallelizes the computation across a cluster of computers [2]. Among its implementations such as Mars [3], Phoenix [4], Dryad [5] and Hadoop [6], Hadoop has received a wide uptake by the community due to its open source nature [7][8][9][10].

One feature of Hadoop MapReduce is its support of public cloud computing that enables the organizations to utilize cloud services in a pay-as-you-go manner. This facility is beneficial to small and medium size organizations where the setup of a large scale and complex private cloud is not feasible due to financial constraints. Hence, executing Hadoop MapReduce applications in a cloud environment for big data analytics has become a realistic option for both the industrial practitioners and academic researchers. For example, Amazon has designed Elastic MapReduce (EMR) that enables users to run Hadoop applications across its Elastic Cloud Computing (EC2) nodes.

The EC2 Cloud makes it easier for users to set up and run Hadoop applications on a large-scale virtual cluster. To use the EC2 Cloud, users have to configure the required amount of resources (virtual nodes) for their applications. However, the EC2 Cloud in its current form does not support Hadoop jobs with deadline requirements. It is purely the user's responsibility to estimate the amount of resources to complete their jobs which is a highly challenging task. Hence, Hadoop performance modeling has become a necessity in estimating the right amount of resources for user jobs with deadline requirements. It should be pointed out that modeling Hadoop performance is challenging because Hadoop jobs normally involve multiple processing phases including three core phases (i.e. map phase, shuffle phase and reduce phase). Moreover, the first wave of the shuffle phase is normally processed in parallel with the map phase (i.e. overlapping stage) and the other waves of the shuffle phase are processed after the map phase is completed (i.e. non-overlapping stage).

To effectively manage cloud resources, several Hadoop performance models have been proposed [11][12][13][14]. However, these models do not consider the overlapping and non-overlapping stages of the shuffle phase which leads to an inaccurate estimation of job execution.

Recently, a number of sophisticated Hadoop performance models are proposed [15][16][17][18]. Starfish [15] collects a running Hadoop job profile at a fine granularity with detailed information for job estimation and optimization. On the top of Starfish, Elasticiser [16] is proposed for resource provisioning in terms of virtual machines. However, collecting the detailed execution profile of a Hadoop job incurs a high overhead which leads to an overestimated job execution time. The HP model [17] considers both the overlapping and non-overlapping stages and uses simple linear regression for job estimation. This model also estimates the amount of resources for jobs with deadline requirements. CRESP [18] estimates job execution and supports resource provisioning in terms of map and reduce slots. However, both the HP model and CRESP ignore the impact of the number of reduce tasks on job performance. The HP model is restricted to a constant number of reduce tasks, whereas CRESP only considers a single wave of the reduce phase. In CRESP, the number of reduce tasks has to be equal to number of reduce slots. It is unrealistic to configure either the same number of reduce tasks or the single wave of the reduce phase for all the jobs. It can be argued that in practice, the number of reduce tasks varies depending on the size of the input dataset, the type of a Hadoop application (e.g. CPU intensive, or disk I/O intensive) and user requirements. Furthermore, for the reduce phase, using multiple waves generates better performance than using a single wave especially when Hadoop processes a large dataset on a small amount of resources. While a single wave reduces the task setup overhead, multiple waves improve the utilization of the disk I/O.

Building on the HP model, this chapter presents an improved HP model for Hadoop job execution estimation and resource provisioning. The major contributions of this chapter are as follows:

- The improved HP work mathematically models all the three core phases of a Hadoop job. In contrast, the HP work does not mathematically model the non-overlapping shuffle phase in the first wave.
- The improved HP model employs Locally Weighted Linear Regression (LWLR) technique to estimate the execution time of a Hadoop job with a varied number of reduce tasks. In contrast, the HP model employs a simple linear regress technique for job execution estimation which restricts to a constant number of reduce tasks.
- Based on job execution estimation, the improved HP model employs Lagrange Multiplier technique to provision the amount of resources for a Hadoop job to complete within a given deadline.

The performance of the improved HP model is initially evaluated on an in-house Hadoop cluster and subsequently on Amazon EC2 Cloud. The evaluation results show that the improved HP model outperforms both the HP model and Starfish in job execution estimation with an accuracy of level in the range of 94.97% and 95.51%. For resource provisioning, 4 job scenarios are considered with a varied number of map slots and reduce slots. The experimental results show that the improved HP model is more economical in resource provisioning than the HP model.

4.2 Modeling Job Phases in Hadoop

Normally a Hadoop job execution is divided into a map phase and a reduce phase. The reduce phase involves data shuffling; data sorting and user-defined reduce functions. Data shuffling and sorting are performed simultaneously. Therefore, the reduce phase can be further divided into a shuffle (or sort) phase and a reduce phase performing user-defined functions. As a result, an overall Hadoop job execution work flow consists of a map phase, a shuffle phase and a reduce phase as shown in Figure 4.1. Map tasks are executed in map slots at a map phase and reduce tasks run in reduce slots at a reduce phase. Every task runs in one slot at a time. A slot is allocated with a certain amount of resources in terms of CPU and RAM. A Hadoop job phase can be completed in a single wave or multiple waves. Tasks in a wave run in parallel on the assigned slots.

Herodotou presented a detailed set of mathematical models on Hadoop performance at a fine granularity [19]. For the purpose of simplicity, we only consider the three core phases (i.e. map phase, shuffle phase and reduce phase) in modeling the performance of Hadoop jobs. Table 4.1 defines the variables used in Hadoop job performance modeling.

4.2.1 Modeling Map Phase

In this phase, a Hadoop job reads an input dataset from Hadoop Distributed File System (HDFS), splits the input dataset into data chunks based on a specified size and then passes the data chunks to a user-define map function. The map function processes the data chunks and produces a map output. The map output is called intermediate data. The average map output and the total map phase execution time can be computed using Eq. (4.1) and Eq. (4.2) respectively.

$$D_{m-avg}^{output} = D_{m-avg}^{input} \times M_{selectivity} \tag{4.1}$$

$$T_m^{total} = \frac{T_m^{avg} \times N_m}{N_m^{slot}} \tag{4.2}$$

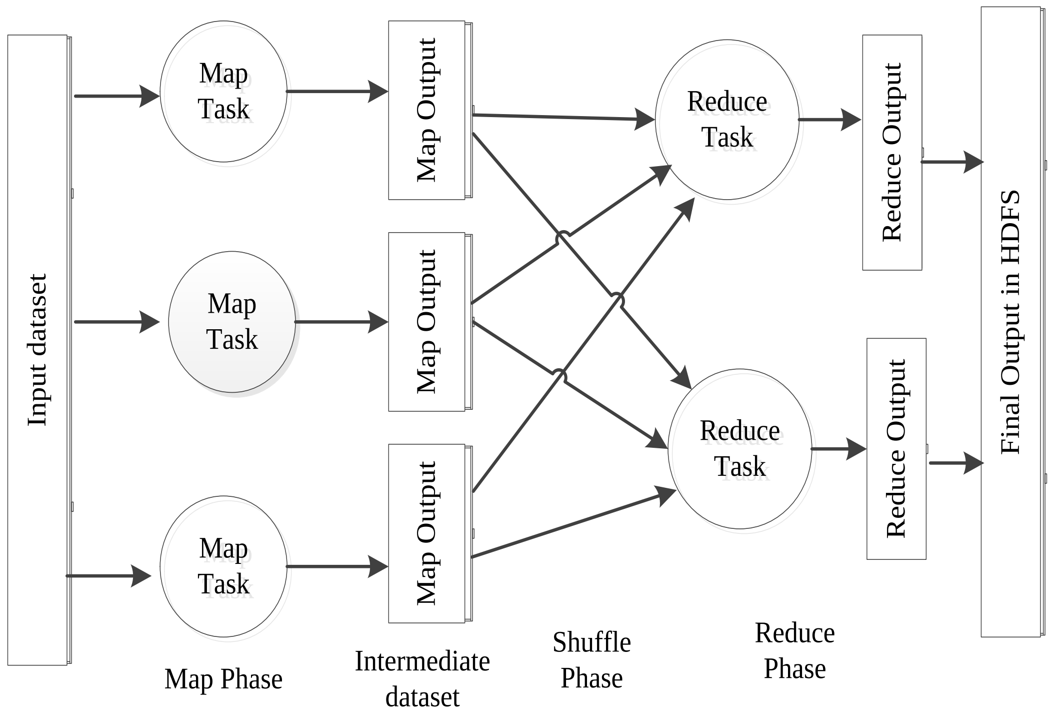

Figure 4.1: Hadoop job execution flow.

4.2.2 Modeling Shuffle Phase

In this phase, a Hadoop job fetches the intermediate data, sorts it and copies it to one or more reducers. The shuffle tasks and sort tasks are performed simultaneously; therefore, we generally consider them as a shuffle phase. The average size of shuffled data can be computed using Eq. (4.3).

$$D_{Sh-avg} = \frac{D_{m-avg}^{output} \times N_m}{N_r} \tag{4.3}$$

If $N_r \leq N_r^{slot}$ then the shuffle phase will be completed in a single wave. The total execution time of a shuffle phase can be computed using Eq. (4.4).

$$T_{sh}^{total} = \frac{T_{sh}^{avg} \times N_r}{N_r^{slot}} \tag{4.4}$$

Table 4.1: Defined variables in modeling job phases

Variables	Expressions
D_{m-avg}^{output}	The average output data size of a map task.
T_{m}^{total}	The total execution time of a map phase.
D_{m-avg}^{input}	The average input data size of a map task.
$M_{selectivity}$	The map selectivity which is the ratio of a map output to a map input.
N_{m}	The total number of map tasks.
T_{m}^{avg}	The average execution time of a map task.
N_{m}^{slot}	The total number of configured map slots.
D_{sh-avg}	The average size of a shuffled data.
T_{sh}^{total}	The total execution time of a shuffle phase.
N_{r}	The total number of reduce tasks.
T_{sh}^{avg}	The average execution duration of a shuffle task.
N_{r}^{slot}	The total number of configured reduce slots.
N_{sh}^{w1}	The total number of shuffle tasks that complete in the first wave.
N_{sh}^{w2}	The total number of shuffle tasks that complete in other waves.
T_{w1}^{avg}	The average execution time of a shuffle task that completes in the first wave.
T_{w2}^{avg}	The average execution time of a shuffle task that completes in other waves.
D_{r-avg}^{output}	The average output data size of a reduce task.
T_{r}^{total}	The total execution time of a reduce phase.
D_{r-avg}^{input}	The average input size of a reduce task.
$R_{selectivity}$	The reduce selectivity which is the ratio of a reduce output to a reduce input.
T_{r}^{avg}	The average execution time of a reduce task.

Otherwise, the shuffle phase will be completed in multiple waves and its execution time can be computed using Eq. (5.5).

$$T_{sh}^{total} = \frac{(T_{w1}^{avg} \times N_{sh}^{w1}) + (T_{w2}^{avg} \times N_{sh}^{w2})}{N_{r}^{slot}} \tag{5.5}$$

4.2.3 Modeling Reduce Phase

In this phase, a job reads the sorted intermediate data as input and passes to a user-defined reduce function. The reduce function processes the intermediate data and produces a final output. In general, the reduce output is written back into the HDFS. The average output of the reduce tasks and the total execution time of the reduce phase can be computed using Eq. (4.6) and Eq. (4.7) respectively.

$$D_{r-avg}^{output} = D_{r-avg}^{input} \times R_{selectivity} \tag{4.6}$$

$$T_r^{total} = \frac{T_r^{avg} \times N_r}{N_r^{slot}} \tag{4.7}$$

4.3 An Improved HP Performance Model

As also mentioned before, Hadoop jobs have three core execution phases – map phase, shuffle phase and reduce phase. The map phase and the shuffle phase can have overlapping and non-overlapping stages. In this section, we present an improved HP model which takes into account both overlapping stage and non-overlapping stage of the shuffle phase during the execution of a Hadoop job. We consider single Hadoop jobs without logical dependencies.

4.3.1 Design Rationale

A Hadoop job normally runs with multiple phases in a single wave or in multiple waves. If a job runs in a single wave then all the phases will be completed without overlapping stages as shown in Figure 4.2.

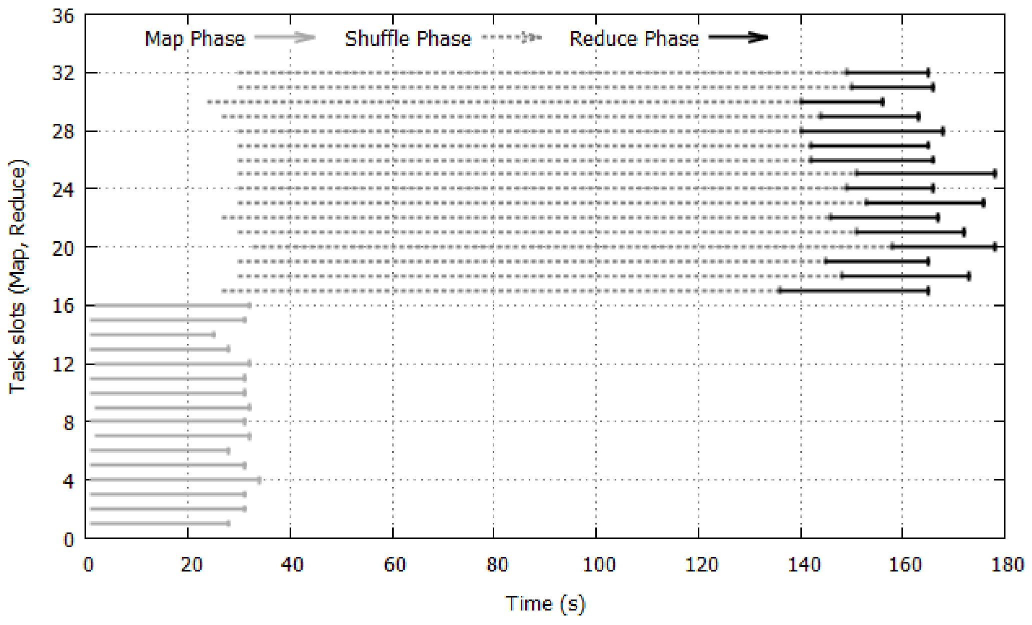

Figure 4.2: A Hadoop job running in a single wave (16 map tasks and 16 reduce tasks).

However, if a job runs in multiple waves, then the job will be progressed through both overlapping (parallel) and non-overlapping (sequential) stages among the phases as show in Figure 4.3.

In the case of multiple waves, the first wave of the shuffle phase starts immediately after the first map task completes. Furthermore, the first wave of the shuffle phase continues until all the map tasks complete and all the intermediate data is shuffled and sorted. Thus, the first wave of the shuffle phase is progressed in parallel with the other waves of the map phase as shown in Figure 4.3. After completion of the first wave of the shuffle phase, the reduce tasks start running and produce output. Afterwards, these reduce slots will become available to the shuffle tasks running in other waves. It can be observed from Figure 4.3 that the shuffle phase takes longer to complete in the first wave than in other waves. In order to estimate the execution time of a job in multiple waves, we need to estimate two sets of parameters for the shuffle phase - the average and the maximum durations of the first wave, together with the average and the maximum durations of the other waves. Moreover, there is no significant difference between the durations of the map tasks running in non-overlapping and overlapping stages due to the equal size of data chunks. Therefore, we only estimate one set of parameters for the map phase which are the average and the maximum durations of the map tasks. The reduce tasks run in a non-overlapping stage, therefore we only estimate one set of parameters for the reduce phase which are the average and the maximum durations of the reduce tasks. Finally, we aggregate the durations of all the three phases to estimate the overall job execution time.

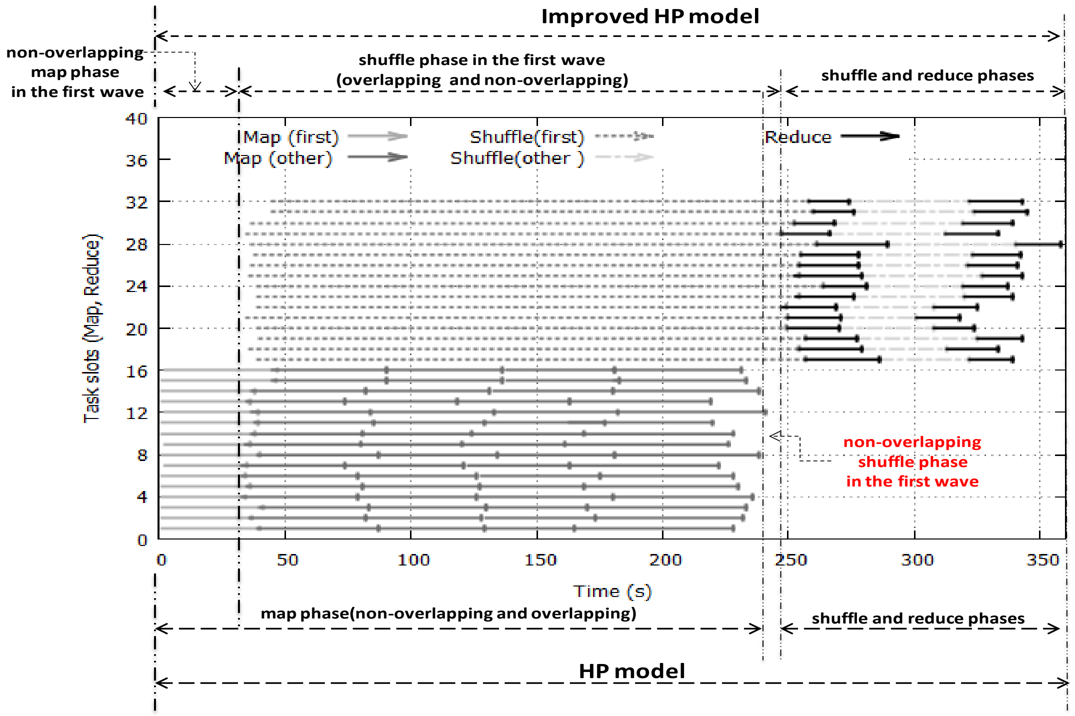

Figure 4.3: A Hadoop job running in multiple waves (80 map tasks, 32 reduce tasks).

It should be pointed out that the Figure 4.3 also shows the differences between the HP model and the improved model in Hadoop job modeling. The HP work mathematically models the whole map phase which includes the non-overlapping stage of the map phase and the stage overlapping with the shuffle phase, but it does not provide any mathematical equations to model the non-overlapping stage of the shuffle phase in the first wave.

Whereas the improved HP work mathematically models the non-overlapping map phase in the first wave, and the shuffle phase in the first wave which includes both the stage overlapping with the map phase and the non-overlapping stage. This can be reflected in the mathematical equations of the improved HP model which are different from the HP model.

4.3.2 Mathematical Expression

In this section, we present the mathematical expressions of the improved HP work in modeling a Hadoop job which completes in multiple waves. Table 4.2 defines the variables used in the improved model.

In practice, job tasks in different waves may not complete exactly at the same time due to varied overhead in disk I/O operations and network communication. Therefore, the improved HP model estimates the lower bound and the upper bound of the execution time for each phase to cover the best-case and the worse-case scenarios respectively.

We consider a job that runs in both non-overlapping and overlapping stages. The lower bound and the upper bound of the map phase in the first wave which is a non-overlapping stage can be computed using Eq.(4.8) and Eq.(4.9) respectively.

$$T_{m-w1}^{low} = \frac{T_m^{avg} \times N_m^{w1}}{N_m^{slot}} \tag{4.8}$$

$$T_{m-w1}^{up} = \frac{T_m^{\max} \times N_m^{w1}}{N_m^{slot}} \tag{4.9}$$

Table 4.2: Defined variables in the improved HP model

Variables	Expressions
T_{m-w1}^{low}	The lower bound duration of the map phase in the first wave (non-overlapping).
T_{m-w1}^{up}	The upper bound duration of the map phase in the first wave (non-overlapping).
N_m^{w1}	The number of map tasks that complete in the first wave of the map phase.
N_m^{w2}	The number of map tasks that complete in other waves of the map phase.
T_m^{max}	The maximum execution time of a map task.

T_{sh-w1}^{low}	The lower bound duration of the shuffle phase in the first wave (overlapping with the map phase).
T_{sh-w1}^{up}	The upper bound duration of the shuffle phase in the first wave (overlapping with the map phase).
T_{sh-w1}^{avg}	The average execution time of a shuffle task that completes in the first wave of the shuffle phase.
$T_{sh-w1}^{\max}$	The maximum execution time of a shuffle task that completes in the first wave of the shuffle phase.
T_{sh-w2}^{low}	The lower bound duration of the shuffle phase in other waves (non-overlapping)
T_{sh-w2}^{up}	The upper bound duration of the shuffle phase in other waves (non-overlapping).
T_{sh-w2}^{avg}	The average execution time of a shuffle task that completes in other waves of the shuffle phase.
T_{sh-w2}^{max}	The maximum execution time of a shuffle task that completes in other waves of the shuffle phase.
T_{r}^{low}	The lower bound duration of the reduce phase.
T_{r}^{up}	The upper bound duration of the reduce phase.
$T_{r}^{\max}$	The maximum execution time of a reduce task.
T_{job}^{low}	The lower bound execution time of a Hadoop job.
T_{job}^{up}	The upper bound execution time of a Hadoop job.
T_{job}^{avg}	The average execution time of a Hadoop job.

In the overlapping stage of a running job, the map phase overlaps with the shuffle phase. Specifically, the tasks running in other waves of the map phase run in parallel with the tasks running in the first wave of the shuffle phase. As the shuffle phase always completes after the map phase which means that the shuffle phase takes longer than the map phase, therefore we use the duration of the shuffle phase in the first wave to compute the lower bound and the upper bound of the overlapping stage of the job using Eq. (4.10) and Eq. (4.11) respectively.

$$T_{sh-w1}^{low} = \frac{T_{sh-w1}^{avg} \times N_{sh}^{w1}}{N_r^{slot}} \tag{4.10}$$

$$T_{sh-w1}^{up} = \frac{T_{sh-w1}^{\max} \times N_{sh}^{w1}}{N_r^{slot}} \tag{4.11}$$

In other waves of the shuffle phase, the tasks run in a non-overlapping stage. Hence, the lower bound and the upper bound of the non-overlapping stage of the shuffle phase can be computed using Eq. (4.12) and Eq. (4.13) respectively.

$$T_{sh-w2}^{low} = \frac{T_{sh-w2}^{avg} \times N_{sh}^{w2}}{N_r^{slot}} \tag{4.12}$$

$$T_{sh-w2}^{up} = \frac{T_{sh-w2}^{\max} \times N_{sh}^{w2}}{N_r^{slot}} \tag{4.13}$$

The reduce tasks start after completion of the shuffle tasks. Therefore, the reduce tasks complete in a non-overlapping stage. The lower bound and the upper bound of the reduce phase can be computed using Eq. (4.14) and Eq. (4.15) respectively.

$$T_r^{low} = \frac{T_r^{avg} \times N_r}{N_r^{slot}} \tag{4.14}$$

$$T_r^{up} = \frac{T_r^{\max} \times N_r}{N_r^{slot}} \tag{4.15}$$

As a result, the lower bound and upper bound of the execution time of a Hadoop job can be computed by combining the execution durations of all the three phases using Eq. (4.16) and Eq. (4.17) respectively.

$$T_{job}^{low} = T_{m-w1}^{low} + T_{sh-w1}^{low} + T_{sh-w2}^{low} + T_{r}^{low} \tag{4.16}$$

$$T_{job}^{up} = T_{m-w1}^{up} + T_{sh-w1}^{up} + T_{sh-w2}^{up} + T_{r}^{up} \tag{4.17}$$

By substituting the values in Eq. (4.16) and Eq. (4.17), we have

$$T_{job}^{low} = \frac{T_m^{avg} \times N_m^{w1}}{N_m^{slot}} + \frac{T_{sh-w1}^{avg} \times N_{sh}^{w1}}{N_r^{slot}} + \frac{T_{sh-w2}^{avg} \times N_{sh}^{w2}}{N_r^{slot}} + \frac{T_r^{avg} \times N_r}{N_r^{slot}} \tag{4.18}$$

$$T_{job}^{up} = \frac{T_m^{\max} \times N_m^{w1}}{N_m^{slot}} + \frac{T_{sh-w1}^{\max} \times N_{sh}^{w1}}{N_r^{slot}} + \frac{T_{sh-w2}^{\max} \times N_{sh}^{w2}}{N_r^{slot}} + \frac{T_r^{\max} \times N_r}{N_r^{slot}} \tag{4.19}$$

Finally, we take an average of Eq. (4.18) and Eq. (4.19) to estimate the execution time of a Hadoop job using Eq. (4.20).

$$T_{job}^{avg} = \frac{T_{job}^{low} + T_{job}^{up}}{2} \tag{4.20}$$

4.3.3 Job Execution Estimation

In the previous section, we have presented the mathematical expressions of the improved HP model. The lower bound and the upper bound of a map phase can be computed using Eq. (4.8) and Eq. (4.9) respectively. However, the durations of the shuffle phase and the reduce phase have to be estimated based on the running records of a Hadoop job.

When a job processes an increasing size of an input dataset, the number of map tasks is proportionally increased while the number of reduce tasks is specified by a user in the configuration file. The number of reduce tasks can vary depending on user's configurations. When the number of reduce tasks is kept constant, the execution durations of both the shuffle tasks and the reduce tasks are linearly increased with the increasing size of the input dataset as considered in the HP model. This is because the volume of an intermediate data block equals to the total volume of the generated intermediate data divided by the number of reduce tasks. As a result, the volume of an intermediate data block is also linearly increased with the increasing size of the input dataset. However, when the number of reduce tasks varies, the execution durations of both the shuffle tasks and the reduce tasks are not linear to the increasing size of an input dataset.

In either the shuffle phase or the reduce phase, we consider the tasks running in both overlapping and non-overlapping stages. Unlike the HP model, the improved model considers a varied number of reduce tasks. As a result, the durations of both the shuffle tasks and the reduce tasks are nonlinear to the size of an input dataset. Therefore, instead of using a simple linear regression as adopted by the HP model, we apply Locally Weighted Linear Regression (LWLR) [20][21] in the improved model to estimate the execution durations of both the shuffle tasks and the reduce tasks.

The LWLR model assigns a weight to each instance x according to its Euclidean distance from the query instance x_q. The LWLR assigns a high weight to an instance x which is close to the query instance x_q and a low weight to the instances that are far away from the query instance x_q. The weight of an instance can be computed using a Gaussian function as illustrated in Eq. (4.21).

$$w_k = \exp\left(-\frac{(distance(x_k, x_q)^2)}{2h^2}\right), (k = 1,2,3,....,m) \tag{4.21}$$

where,

w_k is the weight of the training instance at location k.

x_k is the training instance at location k.

m is the total number of the training instances.

h is a smoothing parameter which determines the width of the local neighborhood of the query instance

The value of h is crucial to LWLR. Users have the option of using a new value of h for each estimation or a single global value of h. However, finding an optimal value for h is a challenging issue itself [22]. In the improved HP model, a single global value of h is used to minimize the estimated mean square errors.

In the improved HP model, the LWLR is used to estimate the durations of both the shuffle tasks and the reduce tasks. First, we estimate T_{sh-w1}^{avg}, which is the average duration of the shuffle tasks running in the first wave of the shuffle phase. To estimate T_{sh-w1}^{avg}, we define a matrix $X \in \Re^{m \times n}$ whose rows contain the training dataset $x_1, x_2, x_3.....,x_m$ and n is the number of feature variables which is set to 2 (i.e. the size of an intermediate dataset and the number of reduce tasks). We define a vector $Y = \lfloor y_1, y_2 ..., y_m \rfloor$ of dependent variables that are used for the average durations of the shuffle tasks. For example, y_i represents the average execution time of the shuffle task that corresponds to the training instance of x_i. We define another matrix X_q whose rows are query instances. Each query instance x_q contains both the size of the intermediate dataset d_{new} and the number of reduce tasks r_{new} of a new job. We calculate d_{new} based on the average input data size of a map task, the total number of map tasks and the map selectivity metric which is $d_{new} = D_{m-input}^{avg} \times N_m \times M_{selectivity}$.

For the estimation of T_{sh-w1}^{avg}, we calculate the weight for each training instance using Eq. (4.21) and then compute the parameter β using Eq. (4.22) which is the coefficient of the LWLR.

$$\beta = (X^T \times W \times X)^{-1}(X^T \times W \times Y) \qquad (4.22)$$

Here $W = diag(w_k)$ is the diagonal matrix where all the non-diagonal cells are 0 values. The value of a diagonal cell is increased when the distance between a training instance and the query instance is decreased.

Finally, the duration of a new shuffle task running in the first wave of the shuffle phase can be estimated using Eq. (4.23).

$$T_{sh-w1}^{avg} = X_q \times \beta \tag{4.23}$$

Similarly, the durations of T_{sh-w1}^{max}, T_{sh-w2}^{avg}, T_{sh-w2}^{max}, T_r^{avg} and T_r^{max} can be estimated.

The estimated values of both the shuffle phase and the reduce phase are used in the improved HP model to estimate the overall execution time of a Hadoop job when processing a new input dataset. Figure 4.4 shows the overall architecture of the improved HP model, which summarizes the work of the improved HP model in job execution estimation. The boxes in gray represent the same work presented in the HP model. It is worth noting that the improved HP model works in an offline mode and estimates the execution time of a job based on the job profile.

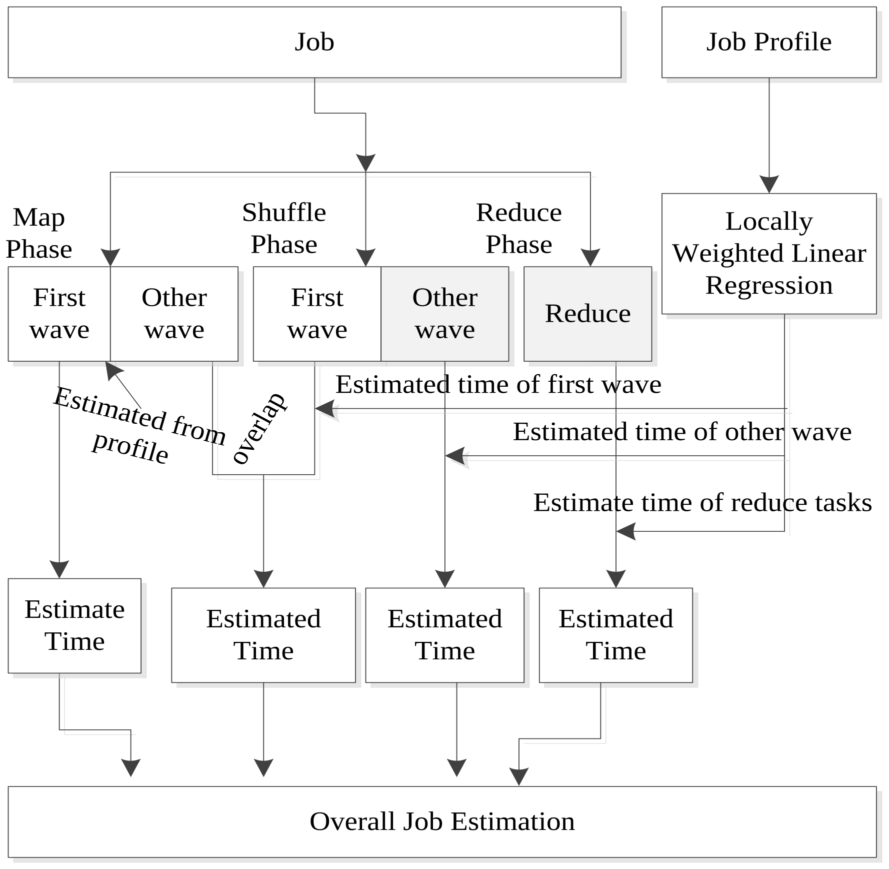

Figure 4.4: The architecture of the improved HP model.

4.4 Resource Provisioning

The improved HP model presented in Section 4.3 can estimate the execution time of a Hadoop job based on the job execution profile, allocated resources (i.e. map slots and reduce slots), and the size of an input dataset. The improved HP model is further enhanced to estimate the amount of resources for Hadoop jobs with deadline requirements.

Consider a deadline for a job that is targeted at the lower bound of the execution time. To estimate the number of map slots and reduce slots, we consider the non-overlapping map phase in the first wave, the map phase in other waves together with the overlapped shuffle phase in the first wave, the shuffle phase in other waves and the reduce phase.

Therefore we simplify Eq. (4.18) into Eq. (4.24) with a modification of Eq. (4.10) for resource estimation.

$$\frac{a}{m}+\frac{b}{m+r}+\frac{c}{r}+\frac{d}{r}=t \tag{4.24}$$

Where

- $t = T_{job}^{low}$
- $a = T_{m}^{avg} \times N_{m}^{w1}$
- $b = (T_{m}^{avg} \times N_{m}^{w2}) + (T_{sh-w1}^{avg} \times N_{sh}^{w1})$
- $c = T_{sh-w2}^{avg} \times N_{sh}^{w2}$
- $d = T_{r}^{avg} \times N_{r}$
- $m = N_{m}^{slot}$
- $r = N_{r}^{slot}$

The method of Lagrange Multipliers [23] is used to estimate the amounts of resources (i.e. map slots and the reduce slots) for a job to complete within a deadline. Lagrange Multipliers is an optimization technique in multivariable calculus that minimizes or maximizes the objective function subject to a constraint function. The objective function is $f(m,r)=m+r$ and the constraint function is $g(m,r)=0$, where $g(m,r)=\frac{a}{m}+\frac{b}{m+r}+\frac{c}{r}+\frac{d}{r}-t$ is derived from Eq. (4.24). To minimize the objective function, the Lagrangian function is expressed as Eq. (4.25).

$$L(m,r,\lambda)= f(m,r)+\lambda g(m,r) \tag{4.25}$$

Where λ is the Lagrange Multiplier. We take partial differentiation of Eq.(4.25) with respect to m, r, λ , we have

$$\frac{\partial L}{\partial m} = 1 - \frac{\lambda a}{m^2} - \frac{\lambda b}{(m+r)^2} = 0 \tag{4.26}$$

$$\frac{\partial L}{\partial r} = 1 - \frac{\lambda b}{(m+r)^2} - \frac{\lambda (c+d)}{r^2} = 0 \tag{4.27}$$

$$\frac{\partial L}{\partial \lambda} = \frac{a}{m} + \frac{b}{m+r} + \frac{c}{r} + \frac{d}{r} - t = 0 \tag{4.28}$$

Solving Eq.(4.26), Eq.(4.27), and Eq.(4.28) simultaneously for m and r, we have

$$m = \frac{\lambda}{x+1}\left[a(x+1)^2 + \frac{ab}{c+d}\right]^{\frac{1}{2}}$$

$$r = \frac{\lambda}{x(x+1)}\left[a(x+1)^2 + \frac{ab}{c+d}\right]^{\frac{1}{2}}$$

where

$$\lambda = \frac{1}{t\left[(x+1)a + \frac{ab}{c+d}\right]^{\frac{1}{2}}}\left[a(x+1) + bx + (c+d)(x(x+1))\right]$$

and $x = \sqrt{\frac{a}{c+d}}$

Here, the values of m and r are the numbers of map slots and reduce slots respectively. As we have targeted at the lower bound of the execution time of a job, the estimated amount of resources might not be sufficient for the job to complete within the deadline.

This is because the lower bound corresponds to the best-case scenario which is hardly achievable in a real Hadoop environment. Therefore, we also target at the upper bound of the execution time of a job. For this purpose we use Eq.(4.19) as a constraint function in Lagrange Multipliers, and apply the same method as applied to Eq.(4.18) to compute the values of both m and r. In this case, the amounts of resources might be overestimated for a job to complete within the deadline. This is because the upper bound corresponds to the worst-case execution of a job. As a result, an average amount of resources between the lower and the upper bounds might be more sensible for resource provisioning for a job to complete within a deadline.

4.5 Performance Evaluation

The performance of the improved HP model was initially evaluated on an in-house Hadoop cluster and subsequently on Amazon EC2 cloud. In this section, we present the evaluation results. First, we give a brief description on the experimental environments that were used in the evaluation process.

4.5.1 Experimental Setup

We set up an in-house Hadoop cluster using an Intel Xeon server machine. The specifications and configurations of the server are shown in Table 4.3. We installed Oracle Virtual Box and configured 8 Virtual Machines (VMs) on the server. Each VM was assigned with 4 CPU cores, 8GB RAM and 150GB hard disk storage. We used Hadoop-1.2.1 and configured one VM as the Name Node and the remaining 7 VMs as Data Nodes. The Name Node was also used as a Data Node. The data block size of the HDFS was set to 64MB and the replication level of data block was set to 2. Two map slots and two reduce slots were configured on each VM. We employed two typical MapReduce applications, i.e. the WordCount application and the Sort application which are CPU intensive and IO intensive applications respectively. The teraGen application was used to generate input datasets of different sizes.

The second experimental Hadoop cluster was setup on Amazon EC2 Cloud using 20 m1.large instances. The specifications of the m1.large are shown in Table 4.3. In this cluster, we used Hadoop-1.2.1 and configured one instance as Name Node and other 19 instances as Data Nodes. The Name Node was also used as a Data Node. The data block size of the HDFS was set to 64MB and the replication level of data block was set to 3. Each instance was configured with one map slot and one reduce slot.

Table 4.3: Experimental Hadoop cluster

Intel Xeon Server 1	CPU	40 cores
	Processor	2.27GHz
	Hard disk	2TB
	Connectivity	100Mbps Ethernet LAN
	Memory	128GB
Amazon m1.large instance	vCPU	2
	Hard disk	420GB
	Memory	7.5GB
Software	Operating System	Ubuntu 12.04 TLS
	JDK	1.6
	Hadoop	1.2.1
	Oracle Virtual Box	4.2.8
	Starfish	0.3.0

4.5.2 Job Profile Information

We run both the WordCount and the Sort applications on the two Hadoop clusters respectively and employed Starfish to collect the job profiles. For each application running on each cluster, we conducted 10 tests. For each test, we run 5 times and took the average durations of the phases. Table 4.4 and Table 4.5 present the job profiles of the two applications that run on the EC2 Cloud.

Table 4.4: The job profile of the WordCount application in EC2 environment.

Data size (GB)	Map tasks	Map task duration (s)		Shuffle duration(s) in the first wave (overlapping)		Shuffle duration(s) in other waves (non-overlapping)		Reduce duration (s)	
		Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
5	80	12	23	69	73	20	22	18	25
10	160	12	24	139	143	26	29	20	32
15	240	13	23	212	215	38	44	23	35
20	320	13	23	274	278	34	39	17	26
25	400	11	25	346	350	41	47	20	27
30	480	11	24	408	411	47	57	22	41
35	560	12	27	486	489	59	71	27	42
40	640	12	24	545	549	45	52	19	30
45	720	11	23	625	629	50	58	20	32
50	800	14	24	693	696	55	65	23	37

Table 4.5:The profile of the Sort application in EC2 environment

Data Size (GB)	Map tasks	Map task duration (s)		Shuffle duration(s) in the first wave (overlapping)		Shuffle duration(s) in other waves (non-overlapping)		Reduce duration (s)	
		Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
5	80	11	15	48	50	15	18	13	24
10	160	12	24	108	111	23	32	30	42
15	240	12	20	161	165	31	41	50	68
20	320	12	22	218	221	29	35	44	63
25	400	13	22	277	281	37	63	57	73
30	480	13	33	325	330	42	56	75	112
35	560	12	27	375	378	55	82	87	132
40	640	13	26	424	428	52	74	71	104
45	720	13	26	484	488	63	94	97	128
50	800	13	29	537	541	71	102	104	144

4.5.2.1 Evaluating the Impact of the Number of Reduce Tasks on Job Performance

In this section we evaluate the impact of the number of reduce tasks on job performance. We run both the WordCount and the Sort applications on the in-house Hadoop cluster with a varied number of reduce tasks. The experimental results are shown in Figure 4.5 and Figure 4.6 respectively. For both applications, it can be observed that when the size of the input dataset is small (e.g. 10GB), using a small number of reduce tasks (e.g. 16) generates less execution time than the case of using a large number of reduce tasks (e.g.

64). However, when the size of the input dataset is large (e.g. 25GB), using a large number of reduce tasks (e.g. 64) generates less execution time than the case of using a small number of reduce tasks (e.g. 16). It can also be observed that when the size of the input dataset is small (e.g. 10GB or 15GB), using a single wave of reduce tasks (i.e. the number of reduce tasks is equal to the number of reduce slots which is 16) performs better than the case of using multiple waves of reduce tasks (i.e. the number of reduce tasks is larger than the number of reduce slots). However, when the size of the input dataset is large (e.g. 25GB), both the WordCount and the Sort applications perform better in the case of using multiple waves of reduce tasks than the case of using a single wave of reduce tasks. While a single wave reduces the task setup overhead on a small dataset, multiple waves improve the utilization of the disk I/O on a large dataset. As a result, the number of reduce tasks affects the performance of a Hadoop application.

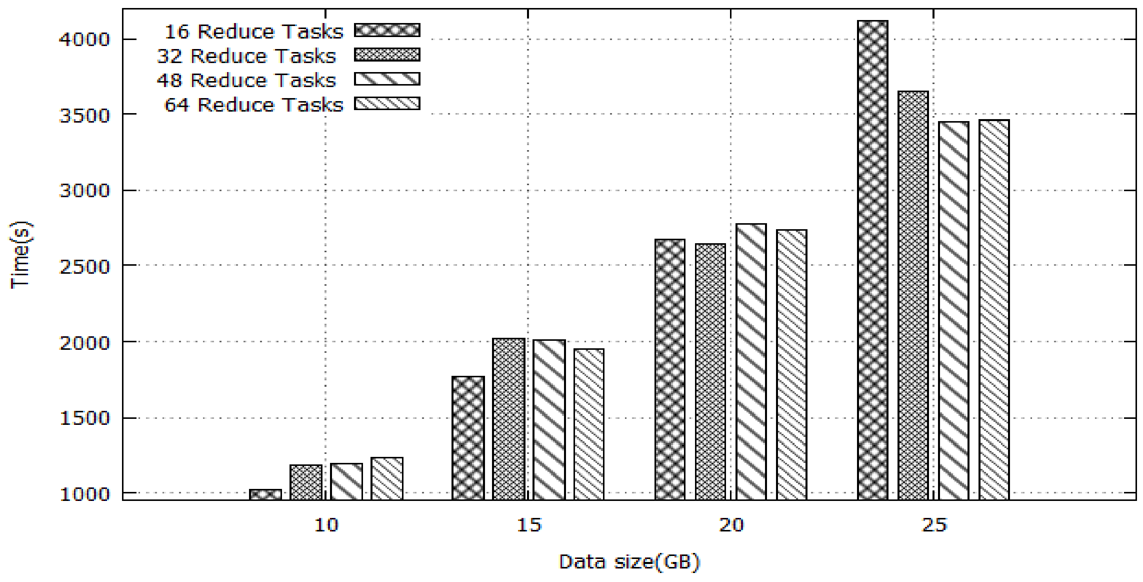

Figure 4.5: The performance of the WordCount application with a varied number of reduce tasks.

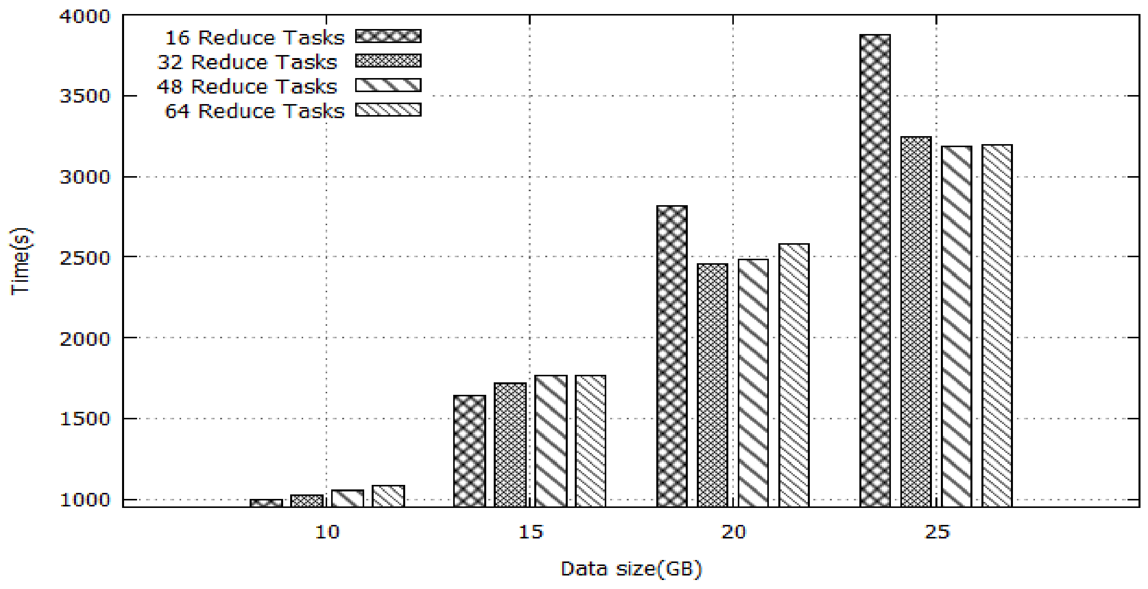

Figure 4.6: The performance of the Sort application with a varied number of reduce tasks.

4.5.2.2 Estimating the Execution Times of Shuffle Tasks and Reduce Tasks

Both the WordCount and the Sort applications processed a dataset on the in-house Hadoop cluster with a varied number of reduce tasks from 32 to 64. The size of the dataset was varied from 2GB to 20GB. Both applications also processed another dataset from 5GB to 50GB on the EC2 Cloud with the number of reduce tasks varying from 40 to 80. The LWLR regression model presented in Section 4.3.3 was employed to estimate the execution times of both the shuffle tasks and the reduce tasks of a new job. The estimated values were used in Eq. (4.18) and Eq. (4.19) to estimate the overall job execution time.

Figure 4.7 and Figure 4.8 show respectively the estimated execution times of both the shuffle tasks and the reduce tasks for both applications running on the Hadoop cluster in EC2. Similar evaluation results were obtained from both applications running on the in-house Hadoop cluster. We can observe that the execution times of both the shuffle tasks

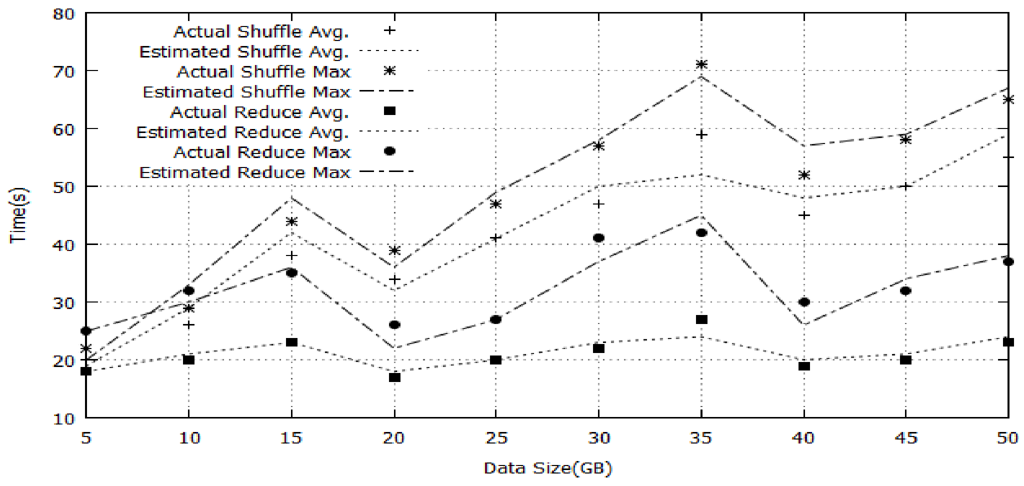

Figure 4.7: The estimated durations of both the shuffle phase (non-overlapping stage) and the reduce phase in the WordCount application. The points represent the actual execution time and dashed lines represent the estimated durations.

(non-overlapping stage) and reduce tasks are not linear to the size of an input dataset. It should be noted that the execution times of the shuffle tasks that run in an overlapping stage are linear to the size of an input dataset because the durations of these tasks depend on the number of map waves, as shown in Table 4.4 and Table 4.5.

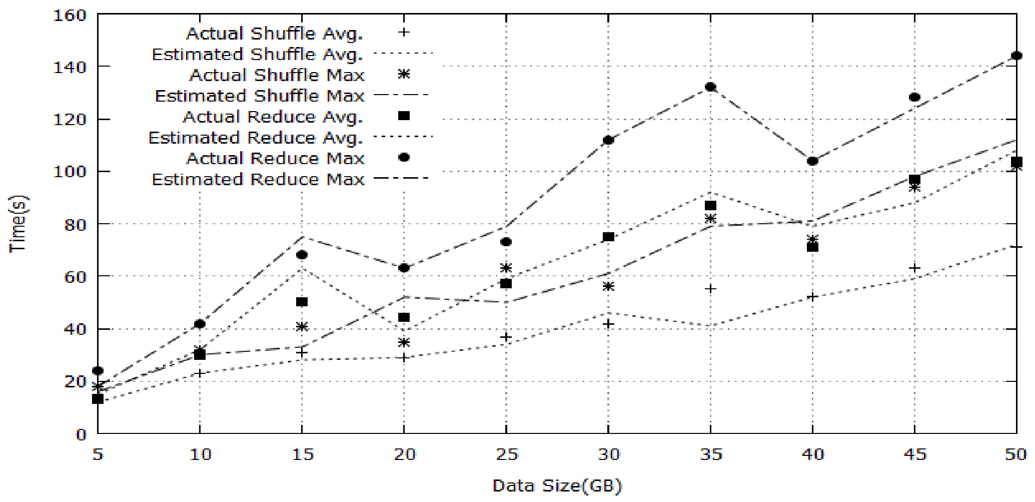

Figure 4.8:The estimated durations of both the shuffle phase (non-overlapping stage) and the reduce phase in the Sort application. The points represent the actual execution time and dashed lines represent the estimated duration.

4.5.3 Job Execution Estimation

A number of experiments were carried out on both the in-house Hadoop cluster and the EC2 Cloud to evaluate the performance of the improved HP model. First, we evaluated the performance of the improved HP model on the in-house cluster and subsequently evaluated the performance of the model on the EC2 Cloud.

For the in-house cluster, the experimental results obtained from both the WordCount and the Sort applications are shown in Figure 4.9 and Figure 4.10 respectively. From these two figures we can observe that the improved HP model outperforms the HP model in both applications. The overall accuracy of the improved HP model in job estimation is within 95% compared with the actual job execution times, whereas the overall accuracy of the HP model is less than 89% which uses a simple linear regression. It is worth noting that the HP model does not generate a straight line in performance as shown in [17]. This is because a varied number of reduce tasks was used in the tests whereas the work presented in [17] used a constant number of reduce tasks.

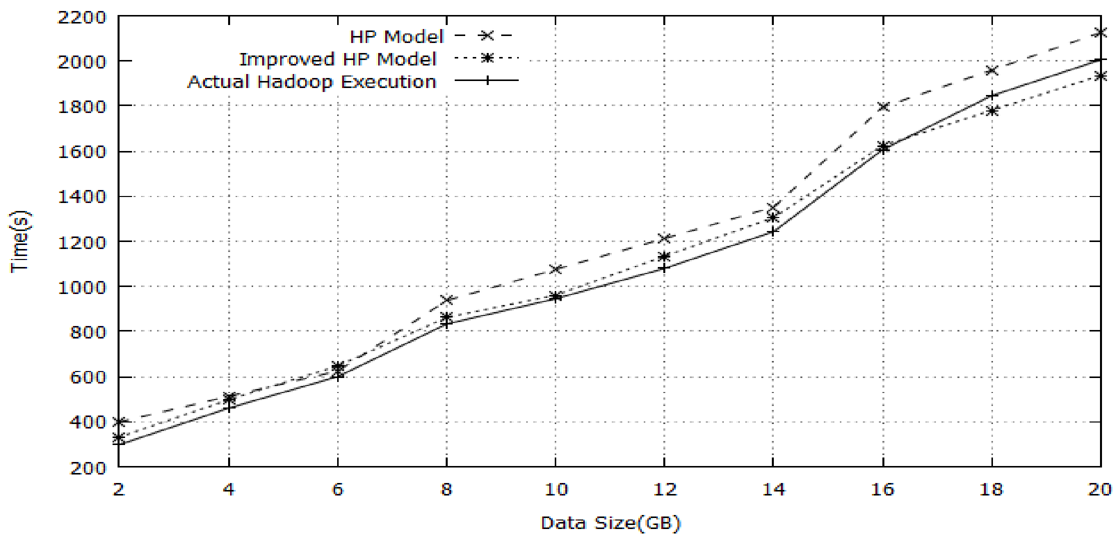

Figure 4.9: The performance of the improved HP model in job estimation of running the WordCount application on the in-house cluster

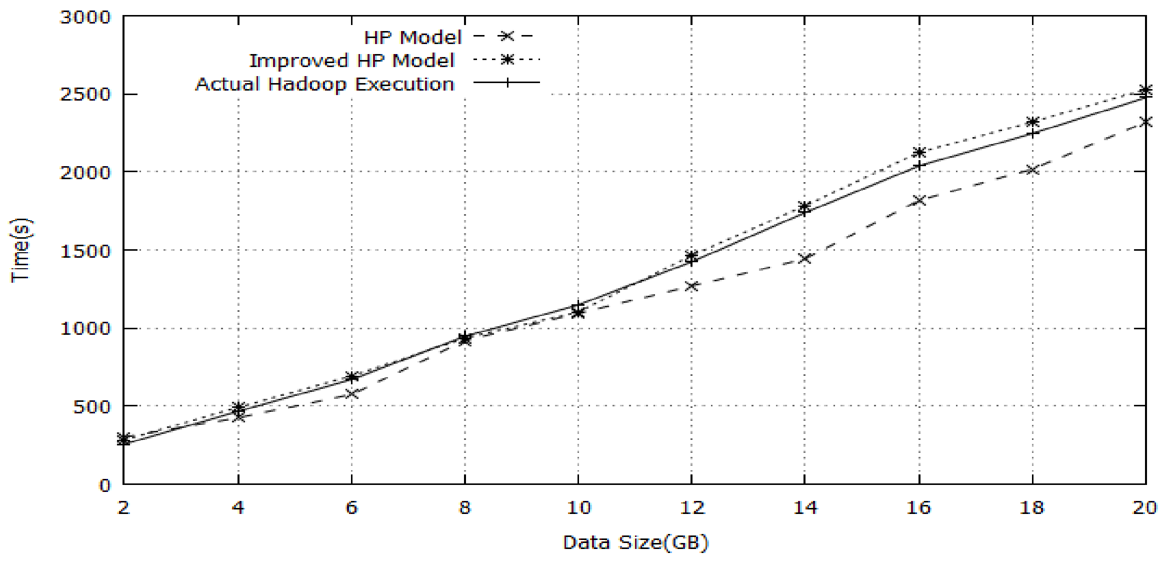

Figure 4.10: The performance of the improved HP model in job estimation of running the Sort application on the in-house cluster.

Next, we evaluated the performance of the improved HP model on the EC2 Cloud. The experimental results in running both applications are shown in Figure 4.11 and Figure 4.12 respectively. It can be observed that the improved HP model also performs better than the HP model. The overall accuracy of the improved HP model in job estimation is over 94% compared with the actual job execution times, whereas the overall accuracy of the HP model is less than 88%. The HP model performs better on small datasets but its accuracy level is decreased to 76.15% when the dataset is large (e.g. 40GB). The reason is that the HP model employs a simple linear regression which cannot accurately estimate the execution times of the shuffle tasks and the reduce tasks which are not linear to the size of an input dataset.

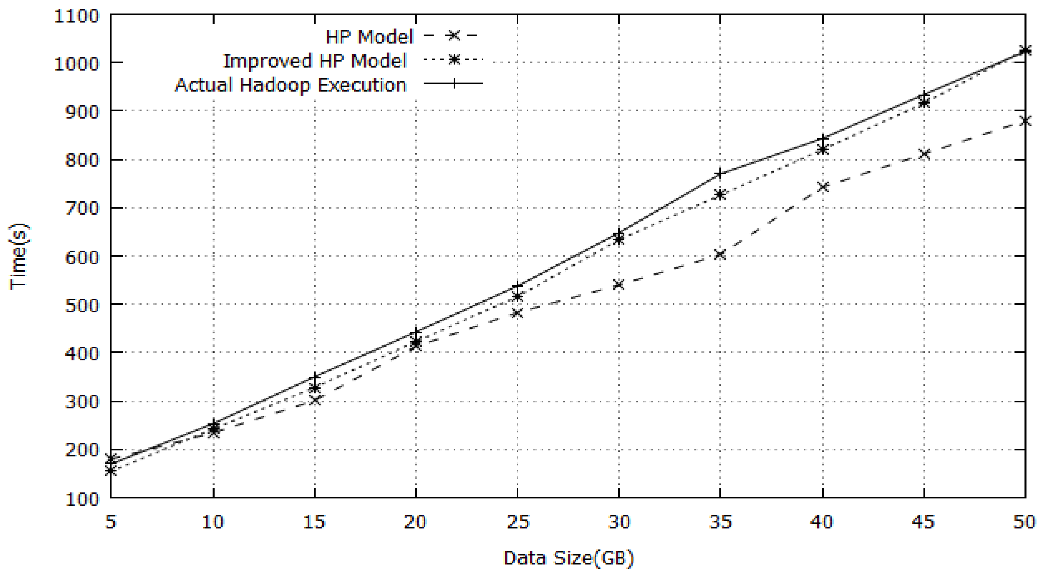

Figure 4.11: The performance of the improved HP model in job estimation of running the WordCount application on the EC2 Cloud.

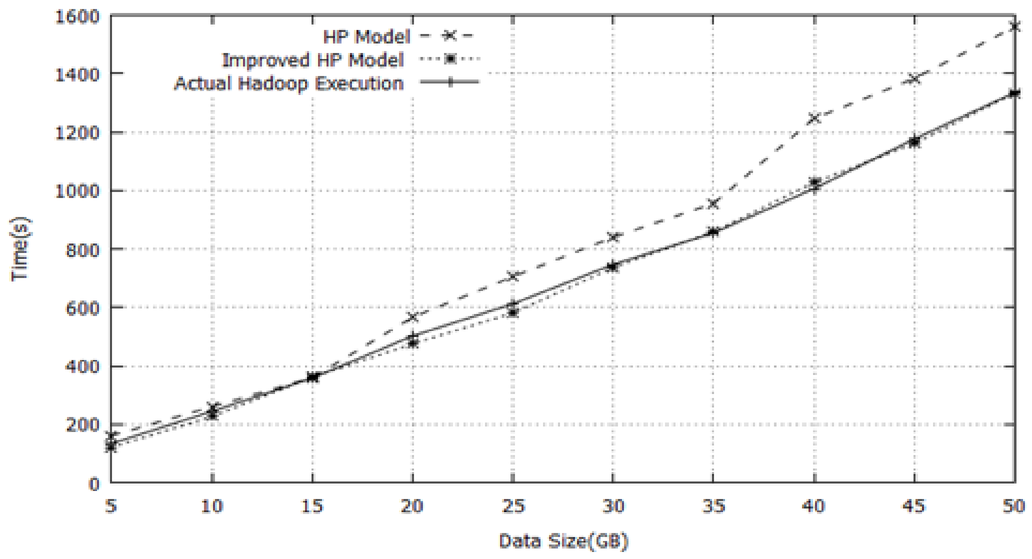

Figure 4.12: The performance of the improved HP model in job estimation of running the Sort application on the EC2 Cloud.

Finally, we compared the performance of the improved HP model in job estimation with that of both Starfish and the HP model collectively. Figure 4.13 and Figure 4.14 show the comparison results of the three models running the two applications on the EC2 Cloud respectively.

It can be observed that the improved HP model produces the best results in job estimation for both applications. Starfish performs better than the HP model on the Sort application in some cases as shown in Figure 4.14. However, Starfish overestimates the job execution times of the WordCount application as shown in Figure 4.13. This is mainly due to the high overhead of Starfish in collecting a large set of profile information of a running job. The Starfish profiler generates a high overhead for CPU intensive applications like WordCount because the Starfish uses Btrace to collect job profiles which requires additional CPU cycles [16]. Starfish performs better on the Sort application because Sort is less CPU-intensive than the WordCount application.

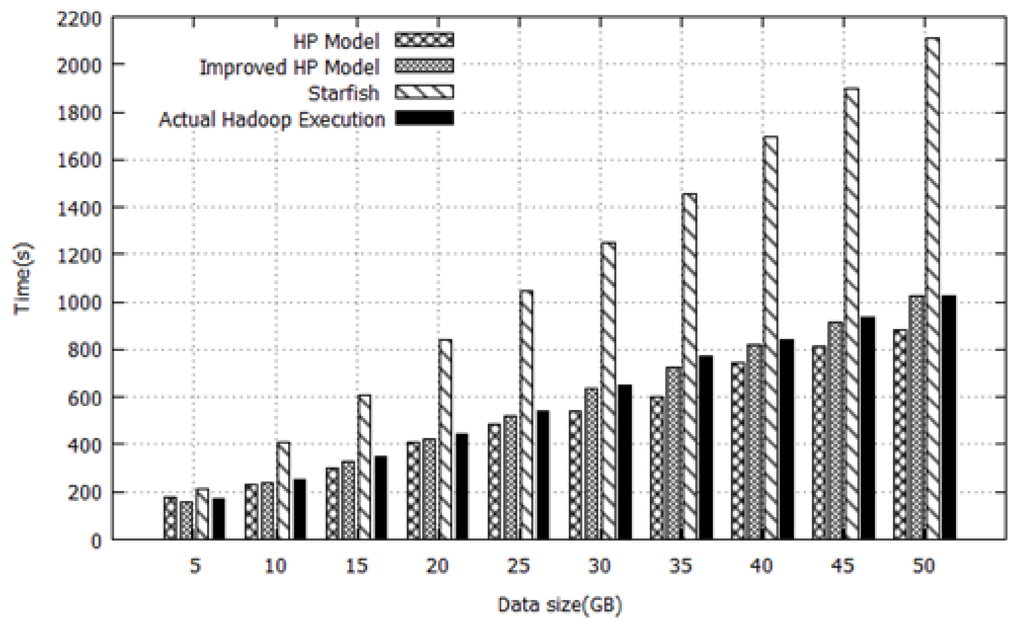

Figure 4.13: A performance comparison among the improved HP model, the HP model and Starfish in running the WordCount application on the EC2 Cloud.

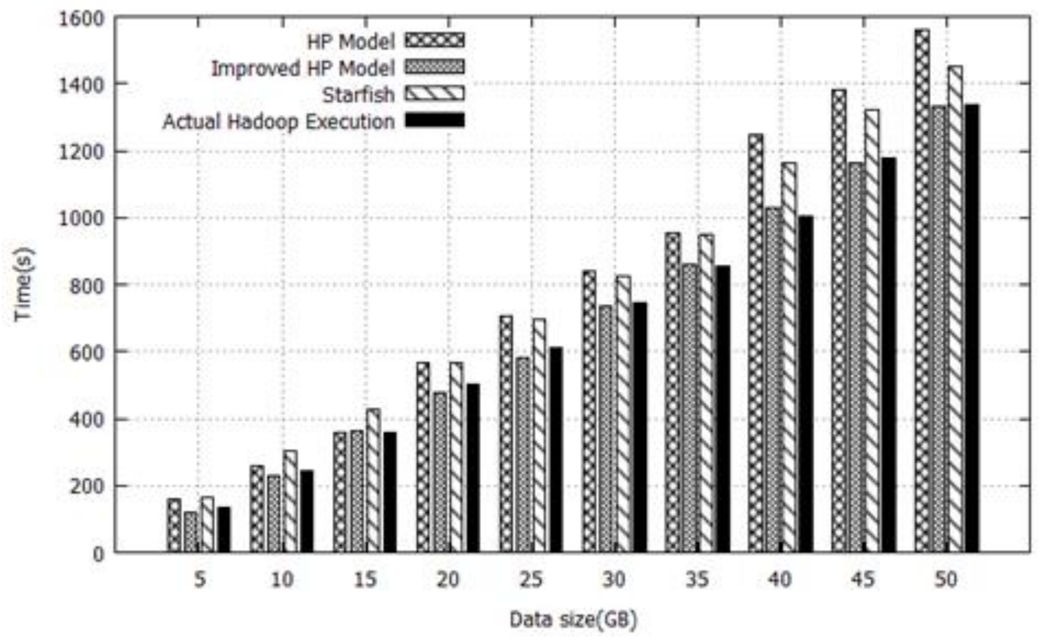

Figure 4.14:A performance comparison among the improved HP model, the HP model and Starfish in running the Sort application on the EC2 Cloud.

We have validated the LWLR regression model in job execution estimation using 10-fold cross validation technique. We considered the execution of an entire job with three phases (i.e. map phase, shuffle phase and reduce phase). The mean absolute percentage errors of the WordCount application and the Sort application are 2.37% and 1.89% respectively which show high generalizability of the LWLR in job execution estimation. Furthermore, the R-squared values of the two applications are 0.9986 and 0.9979 respectively which reflects the goodness of fit of LWLR.

4.5.4 Resource Provisioning

This section present the evaluation results of the improved HP model in resource provisioning using the in-house Hadoop cluster. We considered 4 scenarios as shown in Table 4.6. The intention of varying the number of both map slots and reduce slots from 1 to 4 was twofold. One was to evaluate the impact of the resources available on the performance of the improved HP model in resource estimation. The other was to evaluate the performance of the Hadoop cluster in resource utilization with a varied number of map and reduce slots.

Table 4.6:Scenario configurations.

Scenarios	Number of map slots on each VM	Number of reduce slots on each VM
1	1	1
2	2	2
3	3	3
4	4	4

To compare the performance of the improved HP model with the HP model in resource estimation in the 4 scenarios, we employed the WordCount application as a Hadoop job processing 9.41GB input dataset. In each scenario, we set 7 completion deadlines for the job which are 920, 750, 590, 500, 450, 390 and 350 in seconds. We first built a job profile in each scenario. We set a deadline for the job, and employed both the HP model and the improved HP model to estimate the amount of resources (i.e. the number of map slots and the number of reduce slots). We then assigned the estimated resources to the job using the in-house Hadoop cluster and measured the actual upper bound and the lower bound execution durations. We took an average of an upper bound and a lower bound and compared it with the given deadline. It should be noted that for resource provisioning experiments we configured 16VMs to satisfy the requirement of a job. Therefore, we employed another Xeon server machine with the same specification of the first server as shown in Table 4.3. We installed the Oracle Virtual Box and configured 8 VMs on the second server. Figure from 4.15 to Figure 4.18 shows the results in resource provisioning of the 4 scenarios respectively.

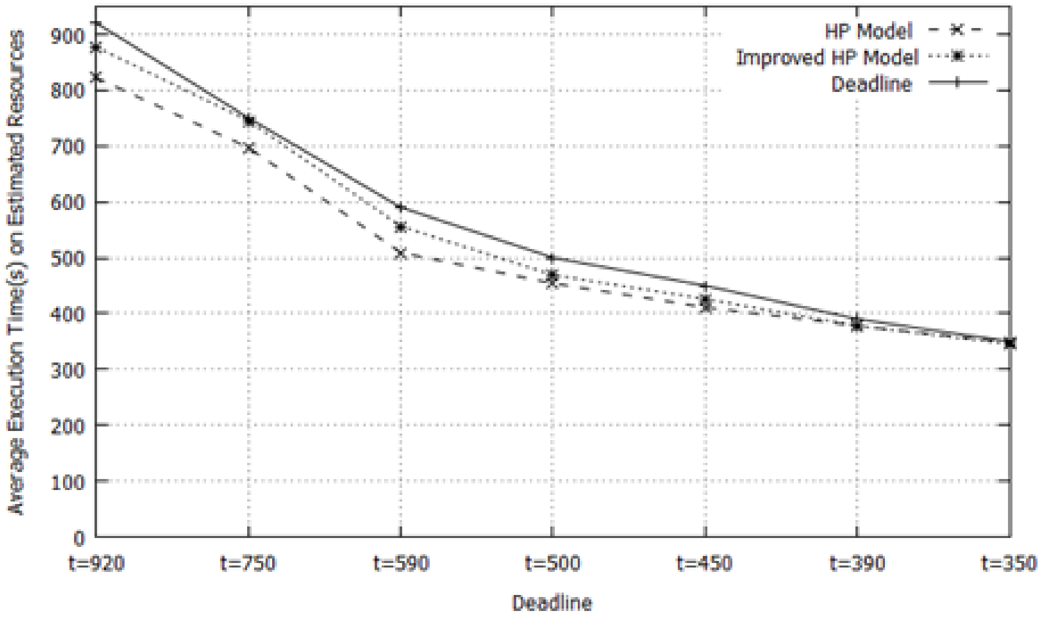

Figure 4.15: Resource provisioning in Scenario 1

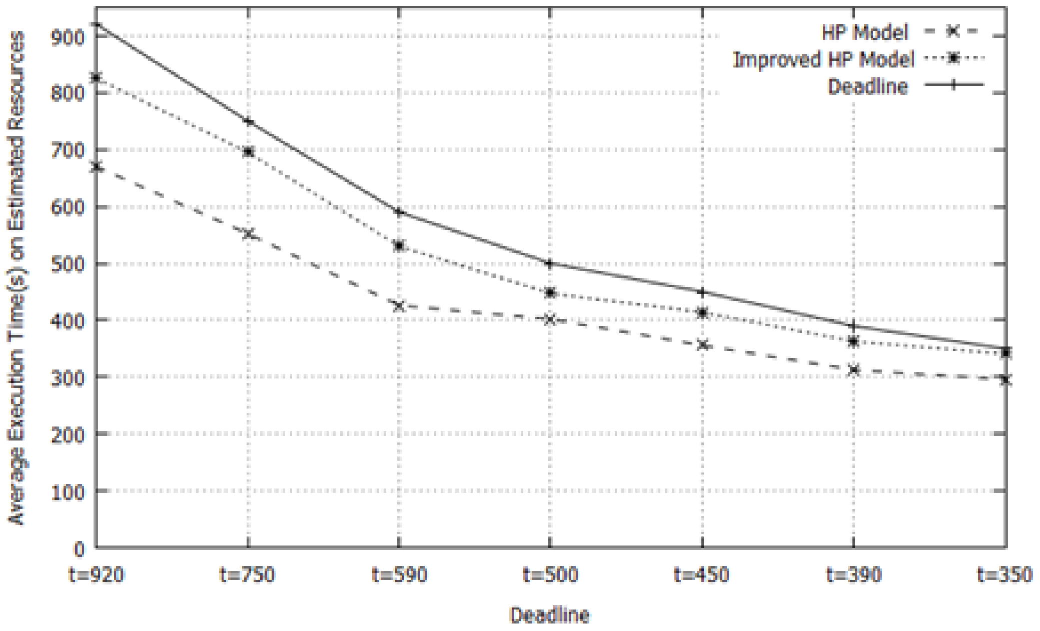

Figure 4.16: Resource provisioning in Scenario 2.

From the 4 scenarios we can see that overall the improved HP model slightly performs better than the HP model in resource provisioning due to its high accuracy in job execution estimation. Both models perform well in the first two scenarios especially in Scenario 1 where the two models generate a near optimal performance. However, the two models over-provision resources in both Scenario 3 and Scenario 4 especially in the cases where the job deadlines are large. The reason is that when we built the training dataset for resource estimation, we run all the VMs in the tests. One rationale was that we consider the worst cases in resource provisioning to make sure all the user job deadlines would be met. However, the overhead incurred in running all the VMs was high and included in resource provisioning for all the jobs. As a result, for jobs with large deadlines, both models overestimate the overhead of the VMs involved. Therefore, both models over-provision the amounts of resources for jobs with large deadlines which can be completed using a small number of VMs instead of all the VMs.

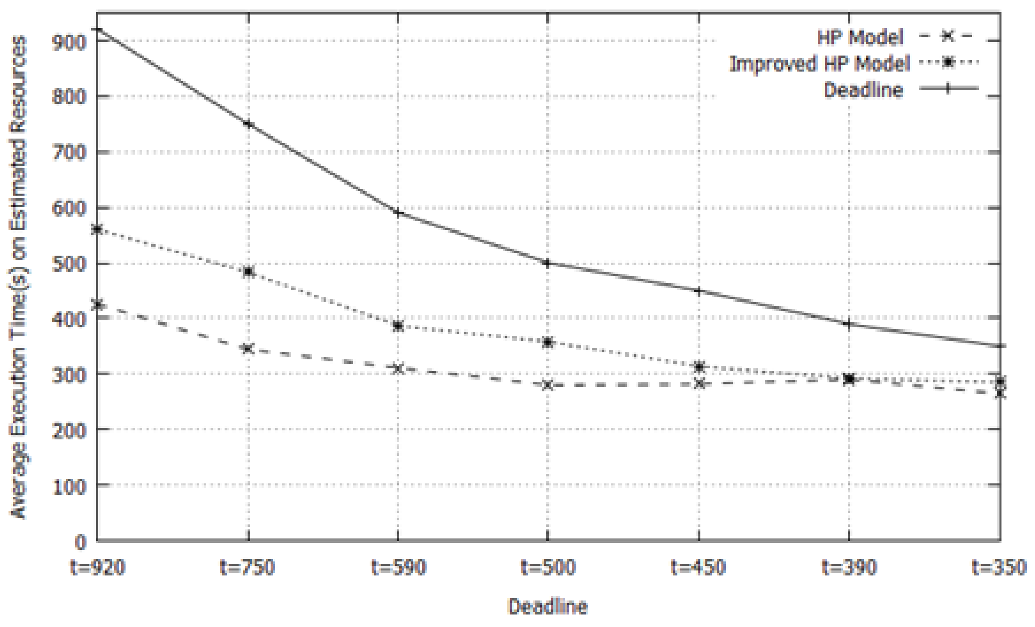

Figure 4.17: Resource provisioning in Scenario 3.

It is worth noting that all the job deadlines are met in the 4 scenarios except the last job deadline in Scenario 4 where t=350. This could be caused by the communication overhead incurred among the VMs running across the two server machines. Although both the improved HP model and the HP model include communication overhead in resource provisioning when the training dataset was built, they only consider static communication overhead. It can be expected that the communication overhead varies from time to time due to the dynamic nature of a communication network.

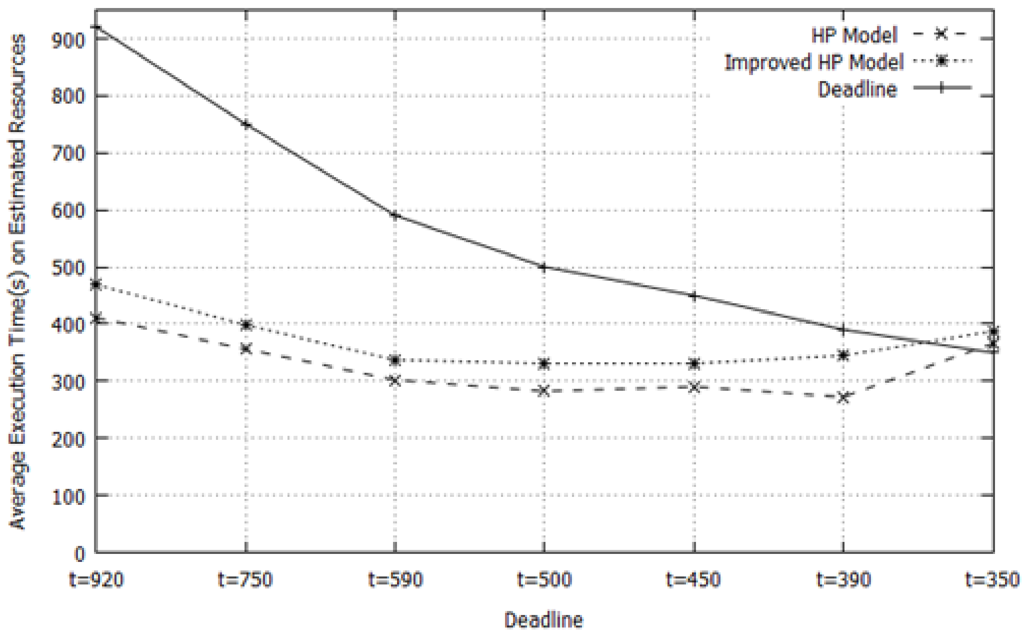

Figure 4.18: Resource provisioning in Scenario 4.

Table 4.7 summarizes the resources estimated by both the HP model and the improved HP model in the 4 scenarios. It can be observed that the HP model recommends more resources in terms of map slots, especially in Scenario 3. This is because the HP model largely considers the map slots in resource provisioning. As a result, the jobs following the HP model are completed quicker than the jobs following the improved HP model but with larger gaps from the given deadlines. Therefore, the improved HP model is more economical than the HP model in resource provisioning due to its recommendations of less map slots.

Table 4.7: The amounts of resources estimated by the HP model and the improved HP model.

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
Deadlines	HP model (m, r)	Improved HP model (m, r)	HP model (m, r)	Improved HP model (m, r)	HP model (m, r)	Improved HP model (m, r)	HP model (m, r)	Improved HP model (m, r)
920	(5,1)	(4,4)	(8,2)	(6,5)	(18,4)	(11,5)	(20,5)	(19,5)
750	(5,2)	(5,5)	(9,3)	(7,6)	(22,5)	(12,6)	(24,6)	(23,6)
590	(7,2)	(6,6)	(12,4)	(9,8)	(28,5)	(16,8)	(30,6)	(29,8)
500	(8,2)	(7,7)	(14,4)	(10,9)	(33,6)	(19,9)	(36,7)	(34,10)
450	(9,3)	(8,8)	(15,5)	(11,10)	(37,7)	(21,10)	(40,8)	(39,10)
390	(10,3)	(9,9)	(18,5)	(13,11)	(42,8)	(24,12)	(46,9)	(44,11)
350	(11,3)	(10,10)	(20,6)	(14,13)	(47,9)	(27,13)	(51,10)	(49,13)

4.6 Related Work

Hadoop performance modeling is an emerging topic that deals with job optimization, scheduling, estimation and resource provisioning. Recently this topic has received a great attention from the research community and a number of models have been proposed.

Morton et al. proposed the parallax model [24] and later the ParaTimer model [25] that estimates the performance of the Pig parallel queries, which can be translated into series of MapReduce jobs. They use debug runs of the same query on input data samples to predict the relative progress of the map and reduce phases. This work is based on simplified suppositions that the durations of the map tasks and the reduce tasks are the same for a MapReduce application. However, in reality, the durations of the map tasks and the reduce tasks cannot be the same because the durations of these tasks are depended on a number of factors. More importantly, the durations of the reduce tasks in overlapping and non-overlapping stages are very different. Ganapathi et al. [26] employed a multivariate Kernel Canonical Correlation Analysis (KCCA) regression technique to predict the performance of Hive query. However, their intention was to show the applicability of KCCA technique in the context of MapReduce.

Kadirvel et al. [27] proposed Machine Learning (ML) techniques to predict the performance of Hadoop jobs. However, this work does not have a comprehensive mathematical model for job estimation. Lin et al. [11] proposed a cost vector which contains the cost of disk I/O, network traffic, computational complexity, CPU and internal sort. The cost vector is used to estimate the execution durations of the map and reduce tasks. It is challenging to accurately estimate the cost of these factors in a situation where multiple tasks compete for resources. Furthermore, this work is only evaluated to estimate the execution times of the map tasks and no estimations on reduce tasks are presented. The later work [12] considers resource contention and tasks failure situations. A simulator is employed to evaluate the effectiveness of the model. However, simulator base approaches are potentially error-prone because it is challenging to design an accurate simulator that can comprehensively simulate the internal dynamics of complex MapReduce applications.

Virajith et al. [13] proposed a system called Bazaar that predicts Hadoop job performance and provisions resources in term of VMs to satisfy user requirements. The work presented in [14] uses the Principle Component Analysis technique to optimize Hadoop jobs based on various configuration parameters. However, these models leave out both the overlapping and non-overlapping stages of the shuffle phase.

There is body of work that focuses on optimal resource provisioning for Hadoop jobs. Tian et al. [28] proposed a cost model that estimates the performance of a job and provisions the resources for the job using a simple regression technique. Chen et al. [18] further improved the cost model and proposed CRESP which employs the brute-force search technique for provisioning the optimal cluster resources in term of map slots and reduce slots for Hadoop jobs. The proposed cost model is able to predict the performance of a job and provisions the resources needed. However, in the two models , the number of reduce tasks have to be equal to the number of reduce slots which means that these two models only consider a single wave of the reduce phase. It is arguable that a Hadoop job performs better when multiple waves of the reduce phase are used in comparison with the use of a single, especially in situations where a small amount of resources is available but processing a large dataset. Lama et al. [29] proposed AROMA, a system that automatically provisions the optimal resources and optimizes the configuration parameters of Hadoop for a job to achieve the service level objectives. AROMA uses clustering techniques to group the jobs with similar behaviors. AROMA uses Support Vector Machine to predict the performance of a Hadoop job and uses a pattern search technique to find the optimal set of resources for a job to achieve the required deadline with a minimum cost. However, AROMA cannot predict the performance of a Hadoop job whose resource utilization pattern is different from any previous ones. More importantly, AROMA does not provide a comprehensive mathematical model to estimate a job execution time as well as optimal configuration parameter values of Hadoop.

There are a few other sophisticated models such as [15][16][17][30] that are similar to the improve HP model in the sense that they use the previous executed job profiles for performance prediction. Herodotou et al. proposed Starfish [15] which collects the past executed jobs profile information at a fine granularity for job estimation and automatic optimization. On the top of the Starfish, Herodotou et al. proposed Elasticiser [16] which provisions a Hadoop cluster resources in term of VMs. However, collecting detailed job profile information with a large set of metrics generates an extra overhead, especially for CPU-intensive applications. As a result, Starfish overestimate the execution time of a Hadoop job. Verma et al. [30] presented the ARIA model for job execution estimations and resource provisioning. The HP model [17] extends the ARIA mode by adding scaling factors to estimate the job execution time on larger datasets using a simple linear regression. The work presented in [31] divides the map phase and reduce phase into six generic sub-phases (i.e. read, collect, spill, merge, shuffle and write), and uses a regression technique to estimate the durations of these sub-phases. The estimated values are then used in the analytical model presented in [30] to estimate the overall job execution time. In [32], Zhang et al. employed the bound-based approach [30] in heterogeneous Hadoop cluster environments.

It should be pointed out that the aforementioned models are limited to the case that they only consider a constant number of the reduce tasks. As a result, the impact of the number of reduce tasks on the performance of a Hadoop job is ignored. The improved HP model considers a varied number of reduce tasks and employs a sophisticated LWLR technique to estimate the overall execution time of a Hadoop job.

4.7 Summary

This chapter proposed an improved HP model. The improved HP model mathematically modeled three core phases i.e. map phase, shuffle phase and reduce phase included overlapping and non-overlapping stages of a Hadoop job. The proposed model employed LWLR to estimates execution duration of a job that takes into account a varied number of reduce tasks The LWLR model was validated through 10-fold cross-validation technique and its goodness of fit was assessed using R-Squared. For resources provisioning, the model applied Lagrange Multiplier technique to provision right amount of resources for a job to be completed within a given deadline. The performance of the improved HP model in resource provisioning was evaluated in 4 scenarios. The intention was to extensively analyzed the performance of a Hadoop cluster in resource utilization with varied number of map slots and reduce slots. The performance of the improved HP model was intensively evaluated on both an in-house Hadoop cluster and on the EC2 Cloud. The experimental results have shown that the improved HP model outperforms both the Starfish and the HP model in job execution estimation. Both the HP model and the improved HP model provisioned resources for Hadoop jobs with deadline requirements. However, the improved HP model was more economical in resource provisioning than the HP model.

Reference

[1] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[2] R. Lämmel, “Google’s MapReduce programming model — Revisited,” Sci. Comput. Program., vol. 70, no. 1, pp. 1–30, 2008.

[3] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars: a MapReduce framework on graphics processors,” in Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT ’08, 2008, p. 260.

[4] K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, “Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources,” in SIGPLAN Not., 2003, vol. 38, no. 10, pp. 216–229.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” ACM SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 59–72, Mar. 2007.

[6] “Apache Hadoop.” [Online]. Available: http://hadoop.apache.org/. [Accessed: 21-Oct-2013].

[7] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, “The Performance of MapReduce: An In-depth Study,” Proc. VLDB Endow., vol. 3, no. 1–2, pp. 472–483, Sep. 2010.

[8] U. Kang, C. E. Tsourakakis, and C. Faloutsos, “PEGASUS: Mining Peta-scale Graphs,” Knowl. Inf. Syst., vol. 27, no. 2, pp. 303–325, May 2011.

[9] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, “PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce,” Proc. VLDB Endow., vol. 2, no. 2, pp. 1426–1437, Aug. 2009.

[10] A. Pavlo, E. Paulson, and A. Rasin, “A comparison of approaches to large-scale data analysis,” in SIGMOD ’09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009, pp. 165–178.

[11] X. Lin, Z. Meng, C. Xu, and M. Wang, “A Practical Performance Model for Hadoop MapReduce,” in Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on, 2012, pp. 231–239.

[12] X. Cui, X. Lin, C. Hu, R. Zhang, and C. Wang, “Modeling the Performance of MapReduce under Resource Contentions and Task Failures,” in Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on, 2013, vol. 1, pp. 158–163.

[13] J. Virajith, B. Hitesh, C. Paolo, K. Thomas, and R. Antony, “Bazaar: Enabling Predictable Performance in Datacenters,” Microsoft Reasearch, MSR-TR- 2012-38,[Online].Available: http://research.microsoft.com/apps/pubs/default.aspx?id=162192.

[14] H. Yang, Z. Luan, W. Li, D. Qian, and G. Guan, “Statistics-based Workload Modeling for MapReduce,” in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, 2012, pp. 2043–2051.

[15] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, “Starfish: A Self-tuning System for Big Data Analytics,” in In CIDR, 2011, pp. 261–272.

[16] H. Herodotou, F. Dong, and S. Babu, “No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics,” in Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC ’11), 2011, pp. 1–14.

[17] A. Verma, L. Cherkasova, and R. H. Campbell, “Resource provisioning framework for mapreduce jobs with performance goals,” in Proceedings of the 12th ACM/IFIP/USENIX international conference on Middleware, 2011, pp. 165–186.

[18] K. Chen, J. Powers, S. Guo, and F. Tian, “CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds,” IEEE Transcation Parallel Distrib. Syst., vol. 25, no. 6, pp. 1403 – 1412, 2014.

[19] H. Herodotou, “Hadoop Performance Models,” 2011. [Online]. Available: http://www.cs.duke.edu/starfish/files/hadoop-models.pdf. [Accessed: 22-Oct-2013].

[20] W. S. Cleveland and S. J. Delvin, “Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting.,” J. Am. Stat. Assoc., vol. 83, no. 403, pp. 596–610, 1988.

[21] M. Rallis and M. Vazirgiannis, “Rank Prediction in graphs with Locally Weighted Polynomial Regression and EM of Polynomial Mixture Models,” in Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on, 2011, pp. 515–519.

[22] J. Fan and I. Gijbels, Local Polynomial Modelling and Its Applications: Monographs on Statistics and Applied Probability 66. CRC Press, 1996.

[23] A. George, W. Hans, and H. Frank, Mathematical Methods for Physicists, 6th ed. Orlando, FL: Academic Press, 2005, p. 1060.

[24] K. Morton, A. Friesen, M. Balazinska, and D. Grossman, “Estimating the progress of MapReduce pipelines,” in Data Engineering (ICDE), 2010 IEEE 26th International Conference on, 2010, pp. 681–684.

[25] K. Morton, M. Balazinska, and D. Grossman, “ParaTimer: A Progress Indicator for MapReduce DAGs,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 2010, pp. 507–518.

[26] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, “Statistics-driven workload modeling for the Cloud,” in Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on, 2010, pp. 87–92.

[27] S. Kadirvel and J. A. B. Fortes, “Grey-Box Approach for Performance Prediction in Map-Reduce Based Platforms,” in Computer Communications and Networks (ICCCN), 2012 21st International Conference on, 2012, pp. 1–9.

[28] F. Tian and K. Chen, “Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds,” in 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 155–162.

[29] P. Lama and X. Zhou, “AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud,” in Proceedings of the 9th International Conference on Autonomic Computing, 2012, pp. 63–72.

[30] A. Verma, L. Cherkasova, and R. H. Campbell, “ARIA: automatic resource inference and allocation for MapReduce environments.,” in 8th ACM International conference on autonomic computing, 2011, pp. 235–244.

[31] Z. Zhang, L. Cherkasova, and B. T. Loo, “Benchmarking Approach for Designing a Mapreduce Performance Model,” in Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, 2013, pp. 253–258.

[32] Z. Zhang, L. Cherkasova, and B. T. Loo, “Performance Modeling of MapReduce Jobs in Heterogeneous Cloud Environments,” in Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, 2013, pp. 839–846.

Chapter 5

Optimizing Hadoop Configuration Parameter Settings for Enhanced Performance

Hadoop MapReduce has become a major computing technology in support of big data analytics. The Hadoop framework has over 190 configuration parameters and some of them can have a significant effect on the performance of a Hadoop job. Manually tuning the optimum or near optimum values of these parameters is a challenging task and also a time consuming process. This chapter optimizes the performance of Hadoop by automatically tuning its configuration parameter settings. The proposed work first employs Gene Expression Programming technique to build an objective function based on historical job running records, which represents a correlation among the Hadoop configuration parameters. It then employs Particle Swarm Optimization technique which makes use of the objective function to search for optimal or near optimal parameter settings. Experimental results show that the proposed work enhances the performance of Hadoop significantly compared with the default settings. Moreover, it outperforms both Rule-Of-Thumb settings and the Starfish model in Hadoop performance optimization.

5.1 Introduction

Many organizations are continuously collecting massive amounts of datasets from various sources such as the World Wide Web, sensor networks and social networks. The ability to perform scalable and timely analytics on these unstructured datasets is a high priority for many enterprises. It has become difficult for traditional database systems to process these continuously growing datasets. Hadoop MapReduce has become a major computing technology in support of big data analytics [1] [2]. Hadoop has received a wide uptake from the community due to its remarkable features such as high scalability, fault-tolerance and data parallelization. It automatically distributes data and parallelizes computation across a cluster of computer nodes [3]–[7].

Despite these remarkable features, Hadoop is a large and complex framework which has a number of components that interact with each other across multiple computer nodes. The performance of a Hadoop job is sensitive to each component of the Hadoop framework including the underlying hardware, network infrastructure and Hadoop configuration parameters which are over 190. Recent researches show that the parameter settings of the Hadoop framework play a critical role in the performance of Hadoop. A small change in the configuration parameter settings can have a significant impact on the performance of a Hadoop job [8]. Manually tuning the optimum or near optimum values of these parameters is a challenging task and also a time consuming process. In addition, the Hadoop framework has a black box like feature which makes it extremely difficult to find a mathematical model or an objective function which represents a correlation among the parameters. The large parameter space together with the complex correlations among the configuration parameters further increases the complexity of a manual tuning process. Therefore, an effective and automatic approach to tuning Hadoop parameters has become a necessity.

A number of research works have been proposed to automatically tune Hadoop parameter settings. The Rule-Of-Thumb (ROT) proposed by industrial professionals [9][10][11] is just a common practice to tune Hadoop parameter settings. The Starfish optimizer [12][13] optimizes the performance of a Hadoop job based on the job profile and a cost model [14]. The job profile is collected at a fine granularity with detailed information. However, collecting the detailed execution profile of a job incurs a high overhead which overestimates the values for some configuration parameters. Moreover, the Starfish optimizer divides the search space into subspaces in the optimization process which ignores the correlations among the configuration parameters. PPABS [15] automatically tunes Hadoop parameter settings based on the executed job profiles. PPABS employs K-means++ to classify the jobs into equivalent classes. It applies Simulated Annealing to search for optimum parameter values and implements a pattern recognition technique to determine the class that a new job belongs to. However, PPABS is unable to tune the parameter settings for a new job which does not belong to any of the pre-classified classes. Gunther, a search based system proposed in [16] automatically searches for optimum parameter values for the configuration parameters using Genetic Algorithm. One critical limitation of Gunther is that it does not have a fitness function in the implemented Genetic Algorithm. Gunther evaluates the fitness of a set of parameter values by running a Hadoop job physically which is a time consuming process. Panacea [17] optimizes Hadoop applications based on a process of tuning the configuration parameter settings. Similar to Starfish, Panacea also divides the search space into subspaces and then searches for optimal values within pre-defined ranges. The work presented in [18] proposes a performance evaluation model which focuses on the impact of the Hadoop configuration settings from the aspects of hardware, software and network.

Tuning the configuration parameters of Hadoop requires the knowledge of the internal dynamics of the Hadoop framework and the inter-dependencies among its configuration parameters. This is because the value of one parameter can have a significant impact on the other parameters. It should be pointed out that none of the aforementioned works considers the inter-dependencies among Hadoop configuration parameters. In this paper, we optimize the performance of Hadoop by automatically tuning its configuration parameter settings. The major contributions of this chapter are as follows:

- Based on the running records of Hadoop jobs which can be either CPU intensive or IO intensive, we employ Gene Expression Programming technique (GEP) to build an objective function which represents a correlation among the Hadoop configuration parameters. To the best of our knowledge, this is the first work that mathematically describes the inter-dependencies among the Hadoop configuration parameters when tuning the performance of Hadoop.

- For the purpose of configuration parameter optimization, Particle Swarm Optimization (PSO) [19], [20] is employed that makes use of the GEP constructed objective function to search for a set of optimal or near optimal values of the configuration parameters. Unlike other optimization works that divide the search space into subspaces, the implemented PSO considers the whole search space in the optimization process in order to maintain the inter-dependencies among the configuration parameters.

To evaluate the performance of the proposed work, we run two typical Hadoop MapReduce applications, i.e. WordCount and Sort which are CPU and IO intensive respectively. The performance of the proposed work is initially evaluated on an experimental Hadoop cluster configured with 8 Virtual Machines (VMs) and subsequently on another Hadoop cluster configured with 16 VMs. The experimental results show that the proposed work enhances the performance of Hadoop by on average 67% on the WordCount application and 46% on the Sort application respectively compared with its default settings. The proposed work also outperforms both ROT and the Starfish model in Hadoop performance optimization.

5.2 Hadoop Core Parameters

The Hadoop framework has more than 190 tunable configuration parameters that allow users to manage the flow of a Hadoop job in different phases during the execution process. Some of them are core parameters and have a significant impact on the performance of a Hadoop job [12][16]. The core parameters are briefly presented in Table 5.1.

Table 5.1: Hadoop core configuration parameters

Configuration Parameters	Default Values	Brief Descriptions
io.sort.factor	10	The number of streams that can be merged while sorting.
io.sort.mb	100	The size of the in-memory buffer assigned to each task.
io.sort.spill.percent	0.8	A threshold which determines when to start the spill process, transferring the in-memory data into the hard disk.
mapred.reduce.tasks	1	The number of reduce task(s) configured for a Hadoop job.
mapreduce.tasktracker.	2	The number of map slots configured on each worker

map.tasks.maximum		node.
mapreduce.tasktracker. reduce.tasks.maximum	2	The number of reduce slots configured on each worker node.
mapred.child.java.opts	200	The maximum size of the physical memory of JVM for each task.
mapreduce.reduce.shuffle.input. buffer.percent	0.70	The amount of memory in percentage assigned to a reducer to store map results during the shuffle process.
mapred.reduce.parallel.copies	5	The number of parallel data transfers running in the reduce phase.
mapred.compress.map.output	False	Compression of map task outputs.
mapred.output.compress	False	Compression of reduce task outputs.

***io.sort.factor*:** This parameter determines the number of files (streams) to be merged during the sorting process of map tasks. The default value is 10, but increasing its value improves the utilization of the physical memory and reduces the overhead in IO operations.

***io.sort.mb**:* During a job execution, the output of a map task is not directly written into the hard disk but is written into an in-memory buffer which is assigned to each map task. The size of the in-memory buffer is specified through the *io.sort.mb* parameter. The default value of this parameter is 100MB. The recommended value for this parameter is between 30% and 40% of the *Java_Opts* value and should be larger than the output size of a map task which minimizes the number of spill records [11].

io.sort.spill.percent: The default value of this parameter is 0.8 (80%). When an in-memory buffer is filled up to 80%, the data of the in-memory buffer (*io.sort.mb*) should be spilled into the hard disk. It is recommended that the value of *io.sort.spill.percent* should not be less than 0.50.

***mapred.reduce.tasks*:** This parameter can have a significant impact on the performance of a Hadoop job [21]. The default value is 1. The optimum value of this parameter is mainly dependent on the size of an input dataset and the number of reduce slots configured in a Hadoop cluster. Setting a small number of reduce tasks for a job decreases the overhead in setting up tasks on a small input dataset while setting a large number of reduce tasks improves the hard disk IO utilization on a large input dataset. The recommended number of reduce tasks is 90% of the total number of reduce slots configured in a cluster [8].

mapreduce.tasktracker.map.tasks.maximum,
mapreduce.tasktracker.reduce.tasks.maximum:

These parameters define the number of the map and reduce tasks that can be executed simultaneously on each cluster node. Increasing the values of these parameters increases the utilization of CPUs and physical memory of the cluster node which can improve the performance of a Hadoop job. The optimum values of these parameters are dependent on the number of CPUs, the number of cores in each CPU, multi-threading capability and the computational complexity of a job. The recommended values for these parameters are the number of CPU cores minus 1 as long as the cluster node has sufficient physical memory [9], [11]. One CPU is reserved for other services in Hadoop such as DataNode and TaskTracker.

mapred.child.java.opts: This is a memory related parameter and the main candidate for JVM tuning. The default value is *–Xmx200m* which gives at most 200MB physical memory to each child task. Increasing the value of *Java_Opt* reduces spill operations to output map results into the hard disk which can improve the performance of a job. By default, each work node utilizes 2.8GB physical memory [11]. The worker node assigns 400MB to the map phase (i.e. 2 map slots), 400MB to the reduce phase (i.e. 2 reduce slots) and 1000MB to each DataNode and TaskTracker that run on the worker node.

mapred.compress.map.output, mapred.output.compress:

These two parameters are related to the hard disk IO and network data transfer operations. *Boolean* values are used to determine whether or not the map output and the reduce output need to be compressed. Enabling the compression of the map and reduce outputs for a job can speed up the hard disk IO and minimize the overhead in data shuffling across the network.

5.3 Mining Hadoop Parameter Correlations with GEP

GEP [22] is a new type of Evolutionary Algorithm (EA) [23]. It is developed based on a similar idea to Genetic Algorithms (GA) [24] and Genetic Programming (GP) [25]. Using a special format of the solution representation structure, GEP overcomes some limitations of both GA and GP. GEP brings a significant improvement on problems such as combinatorial optimization, classification, time series prediction, parametric regression and symbolic regression. GEP has been applied to a variety of domains such as data analysis in high energy physics, traffic engineering for IP networks, designing electronic circuits, and evolving classification rules. It has also been applied to data mining field especially for the investigation of an internal correlation among the involved parameters.

GEP uses a chromosome and expression tree combined structure [22] to represent a targeted problem being investigated. The factors of the targeted problem are encoded into a linear chromosome format together with some potential functions which can be used to describe a correlation of the factors. Each chromosome generates an expression tree, and the chromosomes containing these factors are evolved during the evolutionary process.

5.3.1 GEP Design

The execution time of a Hadoop job can be expressed in Eq.(5.1) where $x_0, x_1, \dots, x_n$ are the configuration parameters of Hadoop.

$$ExecutionTime = f(x_0, x_1, \dots, x_n) \tag{5.1}$$

In this work, we consider 10 core parameters of Hadoop as listed in Table 5.2.

Table 5.2: Hadoop core configuration parameters in GEP.

GEP Variables	Hadoop Configuration Parameters	Data Types
x_0	*io.sort.factor*	*integer*
x_1	*io.sort.mb*	*integer*
x_2	*io.sort.spill.percent*	*float*
x_3	*mapred.reduce.tasks*	*integer*
x_4	*mapreduce.tasktracker.map. tasks.maximum*	*integer*
x_5	*mapreduce.tasktracker.reduce.tasks.maximum*	*integer*
x_6	*mapred.child.java.opts*	*integer*
x_7	*mapreduce.reduce.shuffle.input.buffer.percent*	*float*
x_8	*mapred.reduce.parallel.copies*	*integer*
x_9	*input dataset size (GB)*	*integer*

Based on the data types of these Hadoop configuration parameters, the mathematic functions shown in Table 5.3 are used in GEP. A correlation of the Hadoop parameters can be represented by a combination of these mathematical functions. Fig.5.1 shows an example of mining a correlation of 2 parameters (x_0 and x_1) which is conducted in the following steps in GEP:

- Based on the data types of x_0 and x_1, find a mathematical function which has the same input data type as either x_0 or x_1 and has 2 input parameters.
- Calculate the estimated execution time of the selected mathematical function using the parameter setting samples.
- Find the best mathematical function between x_0 and x_1 which produces the closest estimated execution time to the actual execution time. In this case, the *Plus* function is selected.

Table 5.3: Mathematic functions used in GEP.

Functions	Function Descriptions	Input Data Types
Plus	f(a,b)= a + b	*integer* or *float*
Minus	f(a,b)= a – b	*integer* or *float*
multiply	f(a,b)= a * b	*integer* or *float*
Divide	f(a,b)= a / b	*integer* or *float*
Sin	f(a)= sin(a)	*integer* or *float*
Cos	f(a)= cos(a)	*integer* or *float*
Tan	f(a)= tan(a)	*integer* or *float*
Acos	f(a)= acos(a)	*integer* or *float*
Asin	f(a)= asin(a)	*integer* or *float*
Atan	f(a)= atan(a)	*integer* or *float*
Exp	f(a) returns the exponential e^{a}	*integer* or *float*
Log	f(a)= log(a)	*positive integer* or *float*
log10	f(a) returns the (base-10) logarithm of a	*positive integer* or *float*
Pow	f(a,b) returns base a raised to the power exponent b	*integer* or *float*
Sqrt	f(a)= sqrt(x)	*positive integer* or *float*
Fmod	f(a,b) returns the floating-point remainder of a/b (rounded towards zero)	*integer* or *float*
pow10	f(a) returns base 10 raised to the power exponent a	*integer* or *float*
Inv	f(a)= 1/a	*integer* or *float*
Abs	f(a) returns absolute value of parameter a	*integer*
Neg	f(a)= -a;	*integer* or *float*

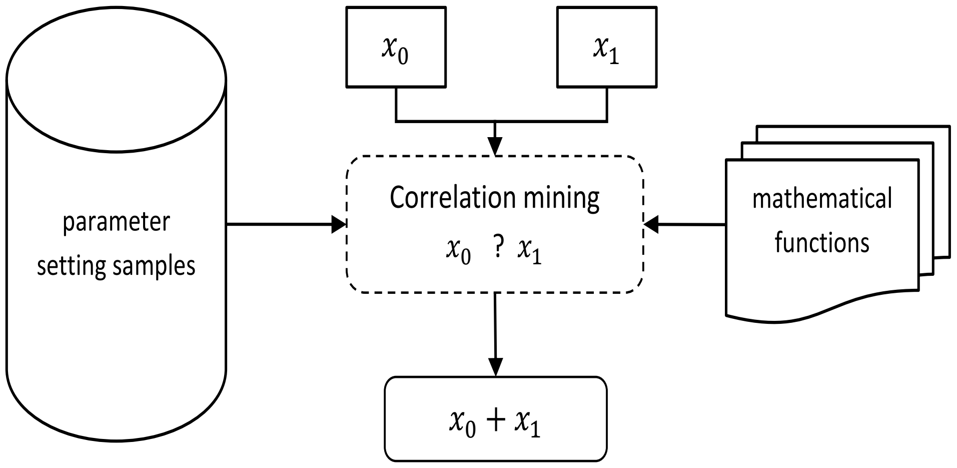

Figure 5.1: An example of parameter correlation mining.

Similarly, a correlation of $x_0, x_1, \ldots, x_n$ can be mined using the GEP method. The chromosome and expression tree structure of GEP is used to hold the parameters and mathematical functions. A combination of mathematical functions which takes $x_0, x_1, \ldots, x_n$ as inputs is encoded into a linear chromosome which is maintained and developed during the evolution process. Meanwhile, the expression tree generated from the linear chromosome produces a form of $f(x_0, x_1, \ldots, x_n)$ based on which an estimated execution time is computed and compared with the actual execution time. A final form of $f(x_0, x_1, \ldots, x_n)$ will be produced at the end of the evolution process whose estimated execution time is the closest to the actual execution time.

In GEP, a chromosome can consist of one or more genes. For simplicity in computation, each chromosome has only one gene in this work. A gene is composed of a head and a tail. The elements of the head are selected randomly from the set of Hadoop parameters (listed in Table 5.2) and the set of mathematical functions (listed in Table 5.3). However, the elements of the tail are selected only from the Hadoop parameter set. The length of a gene head is set to 20 which cover all the possible combinations of the mathematical functions. The length of a gene tail can be computed using Eq.(5.2).

$$Length(Gene_{Tail}) = Length(Gene_{Head}) \times (n-1) + 1 \tag{5.2}$$

Where *n* is the number of input arguments of a mathematical function which has the most number of input arguments among the functions. Fig.5.2 shows an example of a chromosome and expression tree structure taking into account 5 parameters - x_0, x_1, x_2, x_3, x_4 .

In Fig.5.2, the size of the gene head is 4 and n is 2. Then the size of the gene tail is 5 based on Eq.(5.2). Four mathematical functions $(+, -, /, pow)$ are selected to represent a correlation of the parameters x_0, x_1, x_2, x_3, x_4. As a result, a form of $f(x_0, x_1, \ldots, x_n)$ is generated from the expression tree as illustrated in Eq.(5.3).

$$f(x_0, x_1, x_2, x_3, x_4) = (pow(x_3, x_4) - x_0) + (x_1/x_2) \tag{5.3}$$

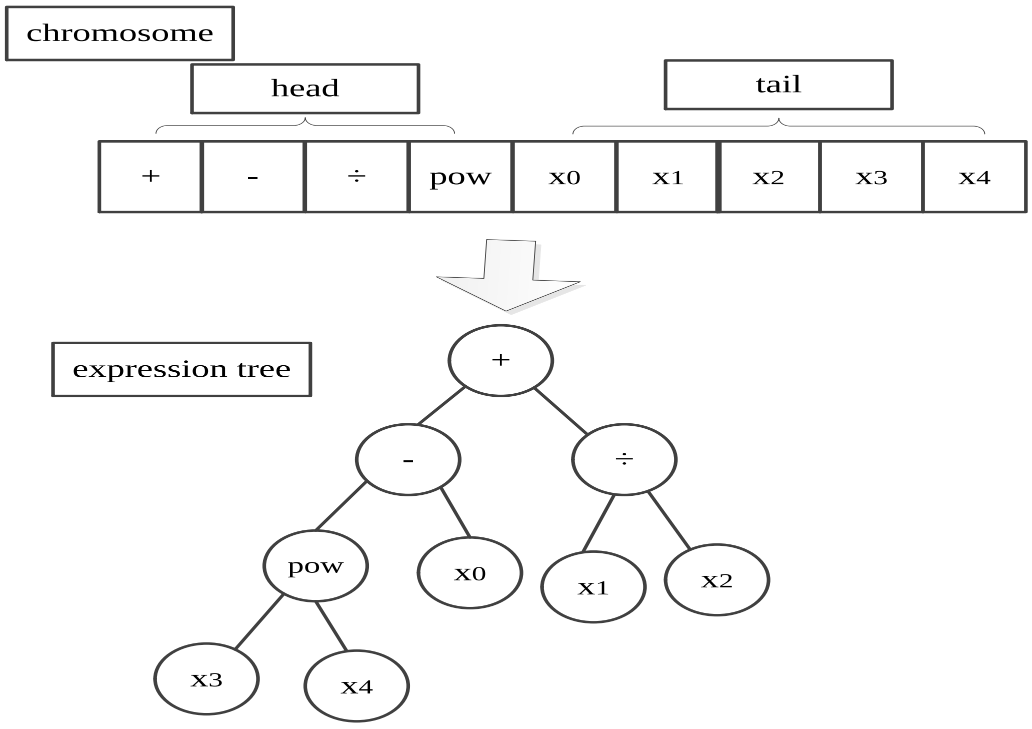

Figure 5.2: An example of chromosome and expression tree structure.

In the following section, we present how the GEP method evolves in mining a correlation among the Hadoop configuration parameters.

5.3.2 GEP Implementation

Algorithm 5.1 shows the implementation of the GEP method. The input of the Algorithm 5.1 is a set of Hadoop job running samples which are used as a training dataset. To build the training dataset, we conducted 320 experiments on a Hadoop cluster which is presented in Section 5.5. We run two typical Hadoop applications (i.e. WordCount and Sort) to process an input dataset of different sizes ranging from 5GB to 15GB. For each experiment, we manually tuned the configuration parameter values and run the two applications 3 times each and took an average of the execution times. A small portion of the training dataset is presented in Table 5.4.

In Algorithm 5.1, Lines 1 to 5 initialize the first generation of 500 chromosomes which represent 500 possible correlations among the Hadoop parameters. Lines 8 to 29 implement an evolution process in which a single loop represents a generation of the evolution process. For each chromosome, it is translated into an expression tree. Lines 11 to 17 calculate the fitness value of a chromosome. For each training sample, GEP produces an estimated execution time of a Hadoop job and makes a comparison with the actual execution time of the job. If the difference is less than a pre-defined bias window, the fitness value of the current chromosome will be increased by 1.

Table 5.4: Training data samples

X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	Time (s)
20	70	0.70	4	2	2	150	0.70	5	5	539
40	83	0.80	15	3	2	170	0.79	5	5	493
40	80	0.81	16	3	1	200	0.80	8	5	518
50	75	0.73	14	3	2	210	0.85	4	5	510
100	75	0.83	8	2	2	150	0.73	5	5	452
120	65	0.85	8	1	1	150	0.75	5	5	540
140	90	0.75	12	2	1	200	0.83	7	5	536
200	66	0.85	12	2	2	160	0.75	5	5	464
200	70	0.80	8	2	1	180	0.71	5	5	454
150	100	0.85	6	1	1	200	0.74	2	5	585
200	73	0.82	8	3	3	260	0.79	8	10	898
200	66	0.85	12	2	2	160	0.75	5	10	857
200	70	0.81	8	2	1	180	0.73	5	10	877
150	100	0.85	6	1	1	200	0.78	2	10	1044
230	75	0.84	7	2	1	190	0.65	5	10	869
100	100	0.66	16	2	2	200	0.70	5	15	1387
30	75	0.73	16	2	2	140	0.69	5	15	1336

```
Input: A set of Hadoop job running samples;
Output: A correlation of the Hadoop parameters;
1:  FOR x=1 TO size of population DO
2:      create chromosome(x) with the combination of mathematic function and parameter ;
3:      fitness value(x) = 0 ;
4:      x++;
5:  ENDFOR
6:  best chromosome = chromosome(1);
7:  best fitness value = 0;
8:  WHILE  i< termination generation number DO
9:        FOR  x=1 TO size of population DO
10:           Translate chromosome(x) into expression tree(x);
```

```
11:              FOR y=1 TO the number of training samples DO
12:                evaluate the estimated execution time for case(y)
13:                  IF ABS(timeDiff)< bias window THEN
14:                      fitness value(x)++;
15:                  ENDIF
16:                y++;
17:              ENDFOR
18:              IF fitness value(x) = the number of training samples THEN
19:                best chromosome = Chromosome(x) GOTO 29;
20:              ELSE IF fitness value(x) > best fitness value THEN
21:                best chromosome = Chromosome(x);
22:                best fitness value = fitness value(x) ;
23:              ENDIF
24:          Apply replication, selection and genetic modification on chromosome(x)
             proportionally;
25:          Use the modified chromosome(x) to overwrite the original one;
26:          x++;
27:      ENDFOR
28:    i++;
29:  ENDWHILE
30: Return best chromosome
```

Algorithm 5.1: GEP implementation.

The size of the bias window is set to 50 seconds which allows a maximum of 10% of the error space taking into account the actual execution time of a Hadoop job sample. Line 18 shows that the evolution process terminates in an ideal case when the fitness value is equal to the number of training samples. Otherwise, the evolution process continues and the chromosome with the best fitness value will be kept as shown in Lines 20 to 23. At the end of each generation as shown in Lines 24 to 25, a genetic modification is applied to the current generation to generate variations of the chromosomes for the next generation.

We varied the number of generations from 20000 to 80000 in the GEP evolution process and found that the quality of a chromosome (the ratio of the fitness value to the number of training samples) was finally higher than 90%. As a result, we set 80000 as the number of generations. The genetic modification parameters were set using the classic values [22] as shown in Table 5.5.

Table 5.5: GEP parameter settings

Genetic modification parameters of GEP	Values
one-point recombination rate	30%
insertion sequence transposition rate	10%
inversion rate	10%
mutation rate	0.44%

After 80000 generations, GEP generates Eq.(5.4) which represents a correlation of the Hadoop parameters listed in Table 5.2.

$$f(x_0, x_1, \ldots, x_9) = (x_7 * x_6) + (sqrt(1/((log10(x_6) + mod\left(sqrt\left((x_0 * x_8) + (x_3 * x_1)\right), power\left(x_5, (x_2 + x_1)\right)\right)) + (x_6 + x_4))) * (x_8 + x_9)) \tag{5.4}$$

5.4 Hadoop Parameter Optimization with PSO

In this section, we employ PSO to optimize Hadoop parameter settings. We use Eq.(5.4) generated by the GEP method in Section 5.3 as an objective function in PSO optimization.

PSO is a kind of an evolutionary computational algorithm introduced by Eberhart and Kennedy in 1995. The algorithm is inspired by the social behaviors of bird flocking, fish schooling, and swarm theory [19][20]. PSO has been successfully applied in a wide range of problem domains due to its rapid convergence process towards an optimum solution [26]–[30]. In PSO, particles can be considered as agents that fly through a multidimensional search space and record the best solution that they have discovered. Each particle of the swarm adjusts its path according to its own flying experience and also the flying experiences of its neighborhood particles in a multidimensional search space.

Let

- d be the number of dimensions of a search space. In this work, d is set to 9 which represents the 9 Hadoop configuration parameters listed in Table 5.2.
- n be the total number of particles in a swarm.
- $X_{i,j}$ be the list of positions of the particle i , $X_{i,j} = (x_{i,1}, x_{i,2}, x_{i,3}, \ldots\ldots x_{i,d})$, j is a dimension of the search space.
- $P_{i,j}$ be a list of the locally best positions of the particle i, $P_{i,j} = (p_{i,1}, p_{i,2}, p_{i,3}, \ldots\ldots p_{i,d})$.
- $V_{i,j}$ be the velocity of the particle i, $V_{i,j} = (v_{i,1}, v_{i,2}, v_{i,3}, \ldots\ldots v_{i,d})$.
- G be the list of the globally best positions of a swarm, $G = (g_1, g_2, \ldots\ldots g_d)$.

To implement the PSO algorithm, we first initialize the positions of the particles randomly within the bounds of the search space so that the search space is uniformly covered, while the velocities of the particles are initialized to zeros as suggested in [31]. Then the PSO algorithm updates the swarm by updating the velocity and position of each particle in every dimension using Eq.(5.5) and Eq.(5.6) respectively.

$$v_{i,j}^{t+1} = w \times v_{i,j}^{t} + c_1 \times r_1 (p_{i,j}^{t} - x_{i,j}^{t}) + c_2 \times r_2 (g_j^{t} - x_{i,j}^{t}) \qquad (5.5)$$

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} \qquad (5.6)$$

Where

- r_1 and r_2 are cognitive and social randomization parameters respectively. They have random values between 0 and 1.

- c_1 and c_2 are local and global weights respectively. They are acceleration constants.
- w is an inertia weight that balances the global and local search capabilities [32].
- t is a relative time index.
- $v_{i,j}^{t+1}$ is the velocity of the particle i at time step t+1.
- $v_{i,j}^{t}$ is the velocity of the particle i at time step t.
- $p_{i,j}^{t}$ is the locally best position of the particle i at time step t.
- $x_{i,j}^{t}$ is the current position of the particle i at time step t.
- g_{j}^{t} is the globally best position visited by any particle at time step t .
- $x_{i,j}^{t+1}$ is the new position of the particle i at time step t+1.

In each iteration, the new position of a particle is evaluated using the objective function $f(x_0, x_1, \dots, x_9)$. The locally best value is compared with the new fitness value and updated accordingly. Similarly, the globally best position is updated.

In the PSO algorithm, clamping the velocity and position of a particle within a feasible search area is a challenging task. This task becomes even more complicated if the optimization problem has bounds. If the optimization problem has bounds then it is important to handle the particle positions along with the velocities flying out of the feasible area (i.e. out of boundary). In addition, it has been shown that as the number of problem parameters increases, the probability of the particles flying out of the feasible space increases dramatically [33], [34]. For this purpose, we employ the nearest method presented in [34] to handle bound violations.

To handle bound violations of a particle, we define $v_{j,\min}$ and $v_{j,\max}$ which represent a lower bound and an upper bound of the velocity of the particle respectively. Similarly, we define $x_{j,\min}$ and $x_{j,\max}$ representing a lower bound and an upper bound of the position of the particle respectively. The values of the lower bound and the upper bound of the position of a particle are set according to the range of each Hadoop parameter listed in Table 5.6.

Table 5.6: Hadoop parameter setting in PSO.

Hadoop Parameters	Values	Explanations
x_0	10~230	Empirically.
x_1	65~100	Based on the block size of an input dataset. We use 64MB block size in Hadoop.
x_2	0.6 ~0 .85	Empirically.
x_3	1~16	Based on the total number of reduce slots configured in a Hadoop cluster.
x_4	1~3	Based on the specification of a worker node.
x_5	1~3	Based on the specification of a worker node.
x_6	180~6000	Based on the physical memory of a worker node and the x_1 value.
x_7	0.70~0.85	Empirically.
x_8	1~10	Empirically.
x_9	The size of an input dataset in MB	Specified by user.

However, setting the values for the lower bound and the upper bound of the velocity of a particle is problem dependent and the values can be found empirically. We set the value of $v_{j,\min}$ to (-10%) of ($x_{j,\max} - x_{j,\min}$) and the value of $v_{j,\max}$ to (+10%) of ($x_{j,\max} - x_{j,\min}$). Each particle moves in a search space following the upper and lower bounds of its position and velocity. If any particle is roaming then its velocity and position values are set back to the nearest bound values. Algorithm 5.2 shows the PSO implementation.

It is worth pointing out that sometimes PSO can be trapped in a local optimum. This issue can be avoided by adjusting the inertia weight (w) factor used in Eq.(5.5). Instead of using a constant value for w, we use a dynamic inertia weight that linearly decreases in every iteration to overcome the local optima problem [32]. The dynamic inertia weight can be computed using Eq.(5.7).

$$w = w_{\max} - (current\_iteration / total\_iterations) \times (w_{\max} - w_{\min}) \qquad (5.7)$$

where $w_{\min}$= 0 and $w_{\max}$=1.

Input: The size of an input dataset in MB;

Output: A set of PSO recommended Hadoop parameter settings;

```
1.  Initialization process;
2.  FOR each particle i=1 to the number of particles   DO
3.    FOR each dimension j=1 to the number of dimensions   DO
4.        Initialize randomly the position $x_{i,j}$ within a search space ;
5.        Initialize the velocity $v_{i,j}$ = 0;
6.     ENDFOR
7.    fitness_value = $f(x_{i,j})$ ;
8.     IF ( fitness_value <locally_best_value) THEN
9.          locally_best _value = fitness_value;
10.         locally_best _position= $x_{i,j}$ ;
11.     ENDIF
12.     IF ( fitness_value <globally_best_value) THEN
13.          globally_best_value = fitness_value;
14.          globally_best _position= $x_{i,j}$ ;
15.      ENDIF
16.  ENDFOR
17.  WHILE (iteration < the number of iterations) DO
```

```
18.     Compute the new velocity ( $v_{i,j}$ ) using Eq.(5)
19.      IF ( $v_{i,j} < v_{j,\min}$ ) THEN
20.          $v_{i,j} = v_{j,\min}$ ;
21.      ELSE IF ( $v_{i,j} > v_{j,\max}$ ) THEN
22.          $v_{i,j} = v_{j,\max}$ ;
23.      ENDIF
24.     Compute the new position ( $x_{i,j}$ ) using Eq.(6) ;
25.      IF ( $x_{i,j} < x_{j,\min}$ ) THEN
26.          $x_{i,j} = x_{j,\min}$ ;
27.      ELSE IF ( $x_{i,j} > x_{j,\max}$ ) THEN
28.          $x_{i,j} = x_{j,\max}$ ;
29.      ENDIF
30.     Evaluate the new position on fitness function $f$ ;
31.     Update locally_best _position and globally_best _position ;
32. ENDWHILE
33. Output globally_best _position ;
```

Algorithm 5.2. PSO implementation.

5.5 Performance Evaluation

The performance of the proposed optimization work was initially evaluated on an experimental Hadoop cluster using a single Intel Xeon server machine configured with 8 VMs and subsequently on another Hadoop cluster using 2 Intel Xeon Server machines configured with 16 VMs. The intuition of using 2 Hadoop clusters was to intensively evaluate the performance of the proposed work by considering the network overhead across the 2 server machines. In this section, we first give a brief introduction to the experimental environments that were set up in the evaluation process and then present performance evaluation results.

5.5.1 Experimental Setup

We set up a Hadoop cluster using one Intel Xeon server machine. The specification of the server is shown in Table 5.7. We installed Oracle Virtual Box and configured 8 VMs on the server. Each VM was assigned with 4 CPU cores, 8GB RAM and 150GB hard disk storage. We installed Hadoop-1.2.1 and configured one VM as the Name Node and the remaining 7 VMs as Data Nodes. The Name Node was also used as a Data Node. The data block size of the HDFS was set to 64MB and the replication level of data block was set to 2.

Table 5.7: Hadoop cluster setup.

Intel Xeon Server 1 and Server 2	CPU	40 cores
	Processor	2.27GHz
	Hard disk	2TB
	Connectivity	100Mbps Ethernet LAN
	Memory	128GB
Software	Operating System	Ubuntu 12.04 TLS
	JDK	1.6
	Hadoop	1.2.1
	Oracle Virtual Box	4.2.8
	Starfish	0.3.0

The second experimental Hadoop cluster was set up on 2 Intel Xeon server machines. The specification of second server machine was the same as the first server machine as shown in Table 5.7. The total number of VMs in the second Hadoop cluster was 16. The Hadoop-1.2.1 version was installed and we configured one VM as Name Node and the remaining 15 VMs as Data Nodes. The data block size of the HDFS was set to 64MB and the replication level of data block was set to 3. We run two typical Hadoop applications (i.e. WordCount and Sort) as Hadoop jobs. The TeraGen application of Hadoop was used to generate an input dataset of different sizes.

5.5.2 The Impact of Hadoop Parameters on Performance

We run the WordCount application as a Hadoop job to evaluate the impacts of the configuration parameters listed in Table 5.1 on Hadoop performance. From Fig.5.3 it can be observed that the execution time of the job decreases with an increasing size of the *io.sort.mb* value. The larger size the parameter value has, the less operations will be incurred in writing the spill records to the hard disk leading to a less overhead in output.

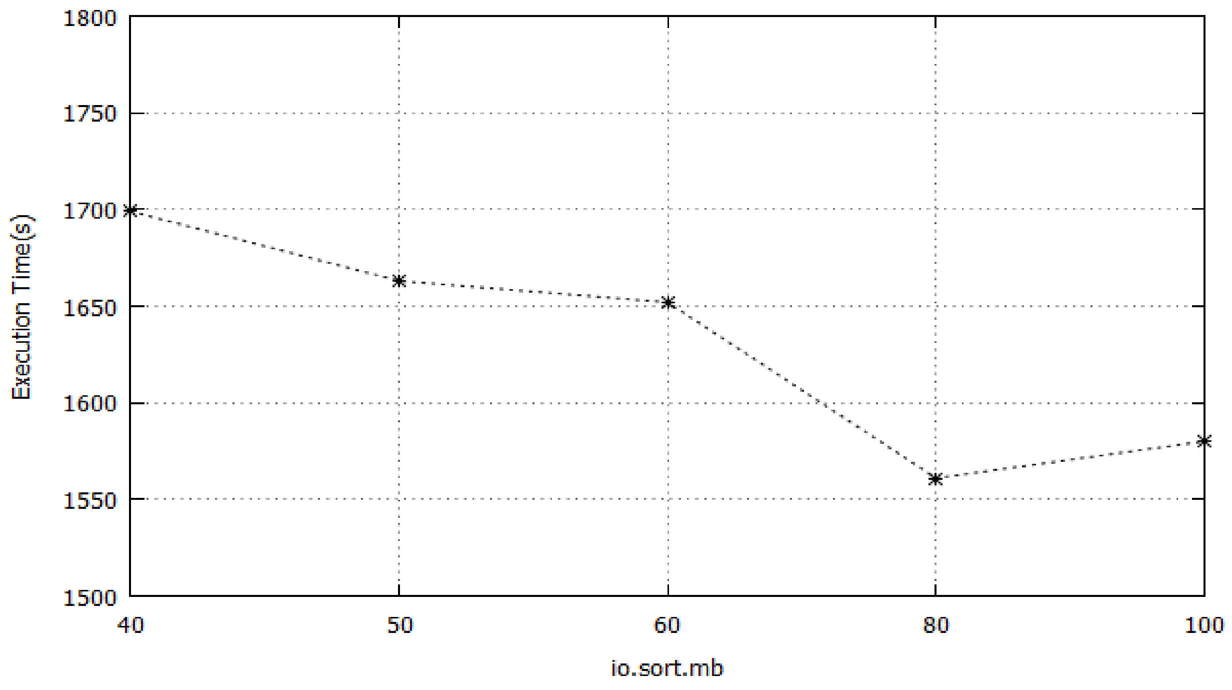

Figure 5.3: The impact of the io.sort.mb parameter

The *io-sort-factor* parameter determines the number of data streams that can be merged in the sorting process. Initially, the execution time of the job goes down with an increasing value of the parameter as shown in Fig.5.4 that the value of 200 represents the best value of the parameter. Subsequently, the execution time goes up when the value of the parameter further increases. This is because that there is a tradeoff between the reduced overhead incurred in IO operations when the value of the parameter increases and the added overhead incurred in merging the data streams.

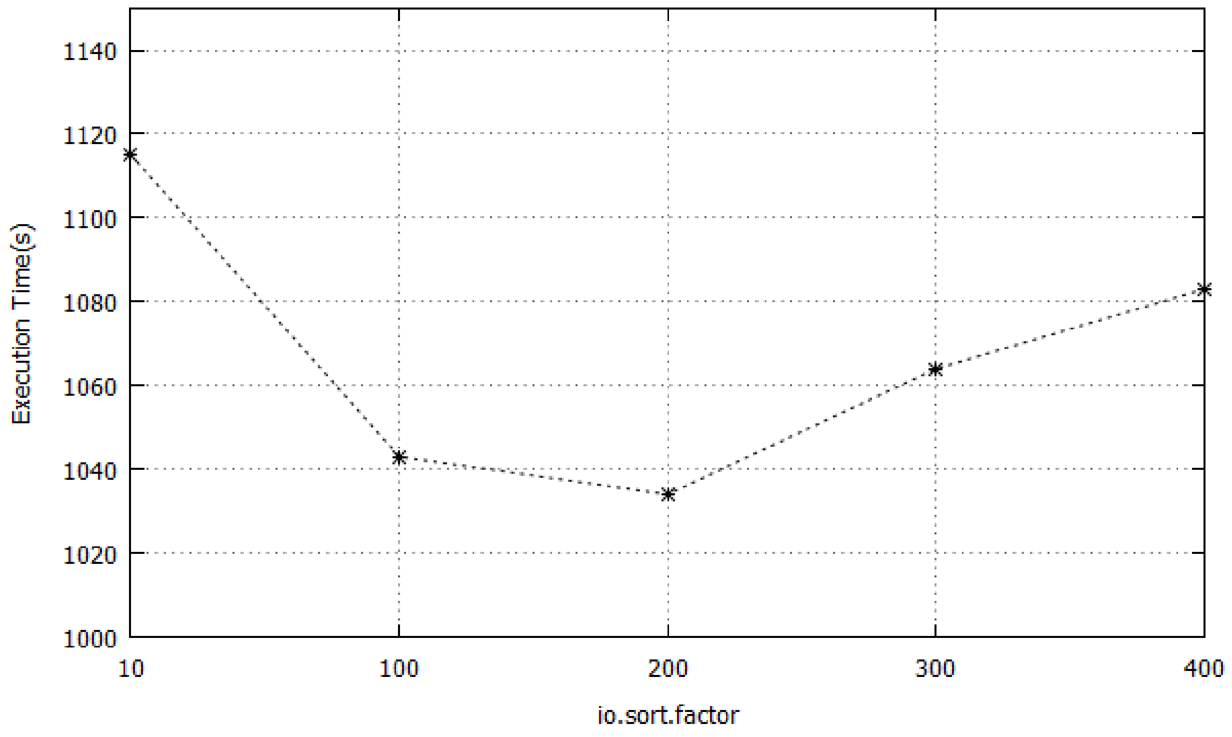

Figure 5.4: The impact of the io-sort-factor parameter.

Fig.5.5 shows the impact of the number of reduce tasks on the job performance. There is a tradeoff between the overhead incurred in setting up reduce tasks and the performance gain in utilizing resources. Initially increasing the number of reduce tasks better utilizes the available resources which leads to a decreased execution time. However, a large number of reduce tasks incurs a high overhead in the setting up process which leads to an increased execution time.

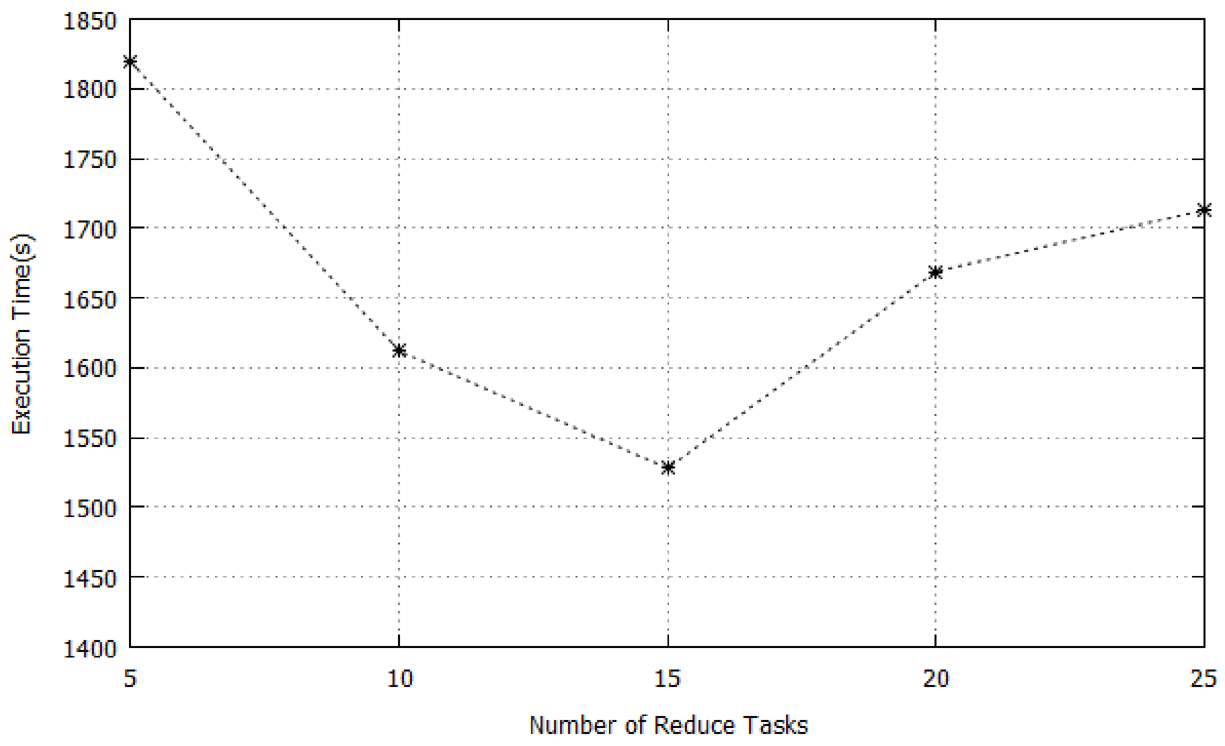

Figure 5.5: The impact of the number of reduce tasks.

Increasing the number of map and reduce slots better utilizes available resources which leads to a decreased execution time which can be observed in Fig.5.6 when the number of slots increases from 1 to 2. However, resources might be over utilized when the number of slots further increases which slows down a job execution.

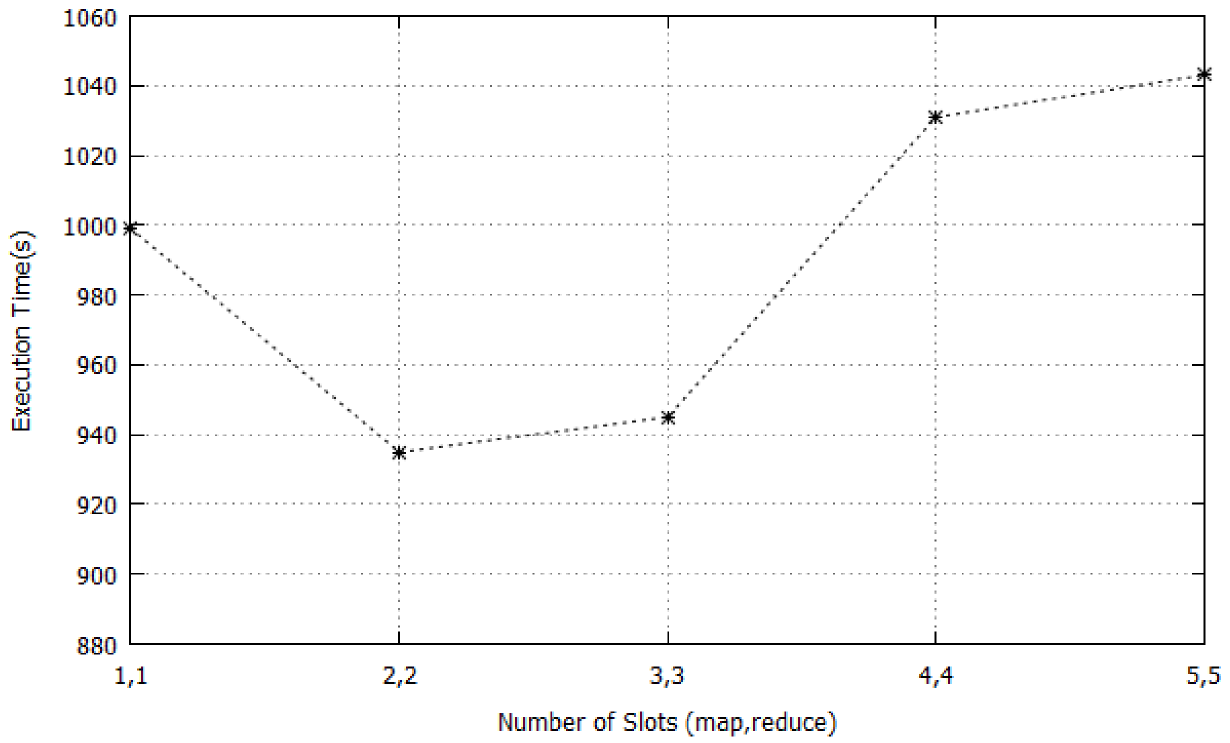

Figure 5.6: The impact of the number of map and reduce slots.

Increasing the value of *Java_opts* parameter utilizes more memory which leads to a decreased execution time as shown in Fig.5.7. However, a large value of the parameter would over utilize the available memory space. In this case, the hard disk is used as a virtual memory which slows down a job execution.

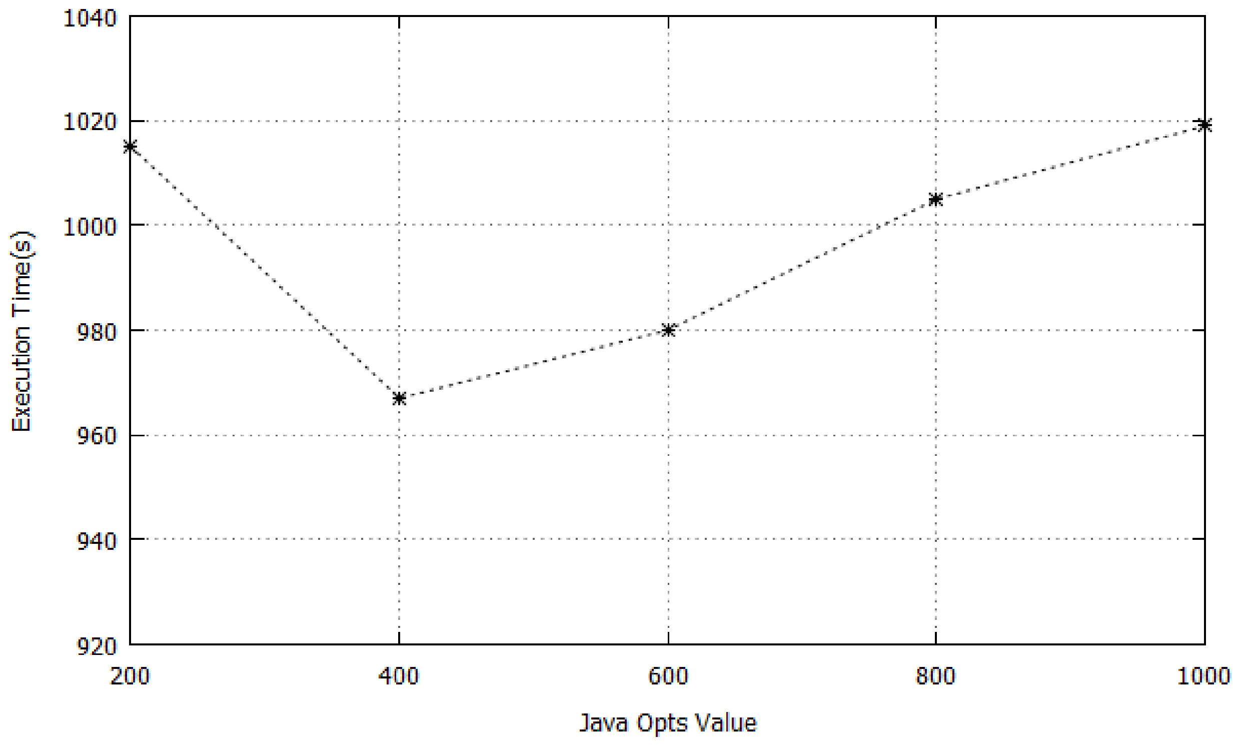

Figure 5.7: The impact of the Java_opts parameter.

.

Fig.5.8 shows the impact of the compression parameter on the performance of a Hadoop job. The results generated by map tasks or reduce tasks can be compressed to reduce the overhead in IO operations and data transfer across network which leads to a decreased execution time. It is worth noting that the performance gap between the case of using the *compression* feature and the case of using *uncompressing* feature gets large with an increasing size of the input data.

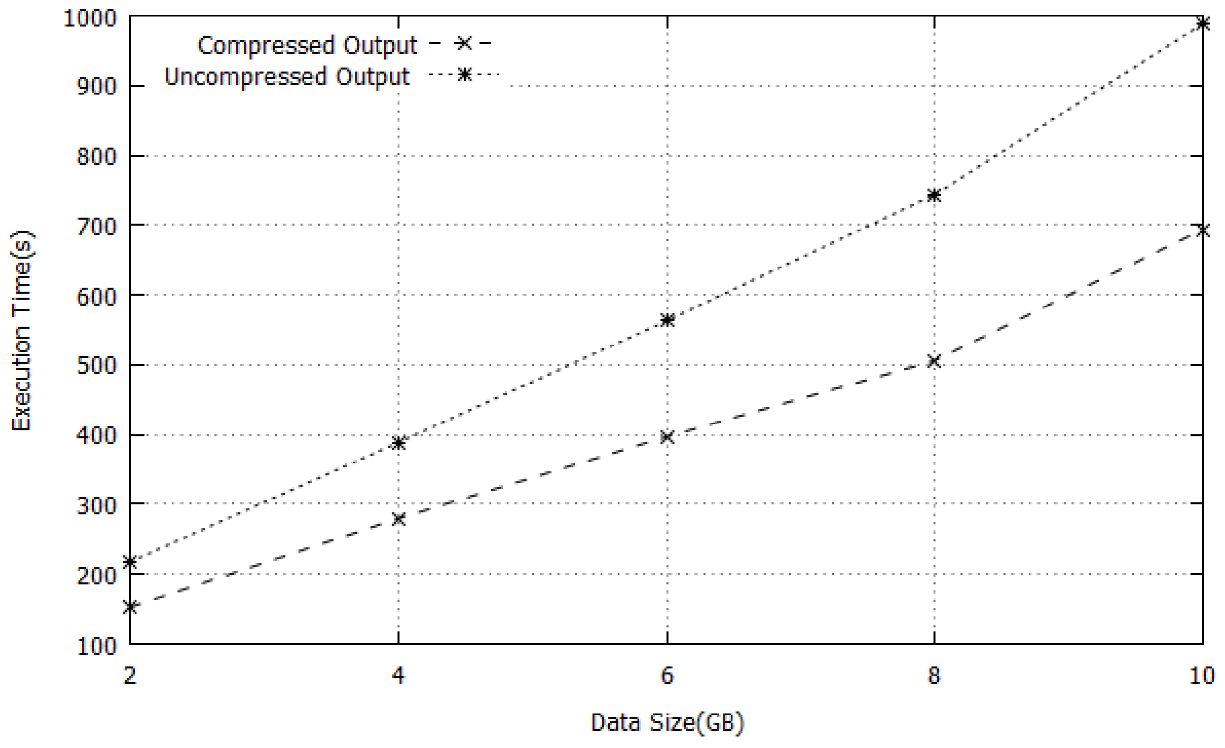

Figure 5.8: The impact of the compression parameter.

5.5.3 PSO Setup

The parameters used in the PSO algorithm are presented in Table 8. We set 20 for the particle swarm size and 100 for the number of iterations as suggested in the literature [35], [36]. The values of c_1 and c_2 were set to 1.4269 as proposed in [37], the value of w was set dynamically between 0 and 1, and the values of r_1 and r_2 were selected randomly between 0 and 1 in every iteration. The PSO algorithm processes *real number* values while some of the Hadoop configuration parameters accept only *integer number* values (e.g. the number of map slots). We rounded the values of these PSO parameters to *integer* values. We set two configuration parameters which have a *Boolean* value (i.e. *mapred.compress.map.output* and *mapred.out.compress*) to *True*. This is because empirically we found that the *True* values of these two parameters showed a significant improvement on the performance of a Hadoop job as shown in Fig.5.8.

Table 5.8: PSO parameter settings.

Swarm size	20
No. of iterations	100
c_1	1.4269
c_2	1.4269
W	[0,1]
r_1	Random [0,1]
r_2	Random [0,1]

Table 5.9 presents the PSO recommended configuration parameter settings for a Hadoop job with an input dataset of varied sizes ranging from 5GB to 20GB.

Table 5.9: PSO recommend Hadoop parameter settings on 8 VMs.

Configuration Parameters	Optimized Values			
input dataset (GB)	5	10	15	20
io.sort.factor	230	228	213	155
io.sort.mb	100	93	100	91
io.sort.spill.percent	0.85	0.70	0.69	0.76
mapred.reduce.tasks	16	9	10	9
mapreduce.tasktracker.map. tasks.maximum	3	2	2	2
mapreduce.tasktracker.reduce.tasks.maximum	3	2	2	2
mapred.child.java.opts	280	335	420	553
mapreduce.reduce.shuffle.input.buffer.percent	0.7	0.7	0.7	0.7
mapred.reduce.parallel.copies	10	7	6	7
mapred.compress.map. output	True	True	True	True
mapred.output.compress	True	True	True	True

5.5.4 Starfish Job Profile

In order to collect a job profile for the Starfish optimizer, we first run both WordCount and Sort in the Starfish environment with profiler enabled. Both applications processed an input dataset of 5GB. Then the Starfish optimizer was invoked to generate configuration parameter settings. The recommended configuration parameter settings recommended by Starfish for both applications are presented in Table 5.10 and Table 5.11 respectively.

Table 5.10: Starfish recommend parameter settings for the WordCount application on 8 VMs.

Configuration Parameters	Optimized Values			
input dataset (GB)	5	10	15	20
io.sort.mb	117	129	128	120
io.sort.factor	35	50	17	76
mapred.reduce.tasks	32	128	176	192
shuffle.input.buffer percentage	0.43	0.72	0.63	0.83
min.num.spills.for.combine	3	3	3	3
io.sort.spill.percent	0.86	0.85	0.79	.085
io.sort.record.percent	0.23	0.33	0.33	0.31
mapred.job.shuffle.merge.percent	0.86	0.85	0.83	0.69
mapred.inmem.merge. threshold	660	816	827	765
mapred.output.compress	True	True	True	True
mapred.compress.map.output	True	True	True	True
mapred.job.reduce. input.buffer.percent	0.42	0.43	0.60	0.77

Table 5.11: Starfish recommend parameter settings for the Sort application on 8 VMs

Configuration Parameters	Optimized Values			
input dataset (GB)	5	10	15	20
io.sort.mb	110	127	109	123
io.sort.factor	48	35	54	27
mapred.reduce.tasks	48	112	160	176
shuffle.input.buffer percentage	0.76	0.66	0.63	0.88
io.sort.spill.percent	0.84	0.68	0.87	0.82
io.sort.record.percent	0.21	0.15	0.23	0.11
mapred.job.shuffle.merge.percent	0.77	0.88	0.89	0.76
mapred.inmem.merge. threshold	393	787	783	972
mapred.output.compress	True	True	True	True
mapred.compress.map.output	True	True	True	True
mapred.job.reduce. input.buffer.percent	0.65	0.63	0.52	0.79

5.5.5 Experimental Results on Hadoop Performance

In this section we compare the performance of the proposed work with that of Starfish, ROT and the default configuration parameter settings in Hadoop optimization. Both WordCount and Sort applications were deployed on the Hadoop cluster with 8 VMs to process an input dataset of 4 different sizes varying from 5GB to 20GB. We run both applications 3 times each using the PSO recommended parameter settings and an average of the execution times was taken. The performance results of the two applications are shown in Fig.5.9 and Fig.5.10 respectively.

It can be observed that overall the implemented PSO improves the performance of the WordCount application by an average of 67% in the 4 input data scenarios compared with the default Hadoop parameter settings, 28% compared with Starfish and 26% compared with ROT. The improvement reaches a maximum of 71% when the input data size is 20GB. The performance improvement of the PSO optimization on the Sort application is on average 46% over the default Hadoop parameter settings, 16% over Starfish and 37% over ROT. The improvement reaches a maximum of 65% when the input data size is 20GB.

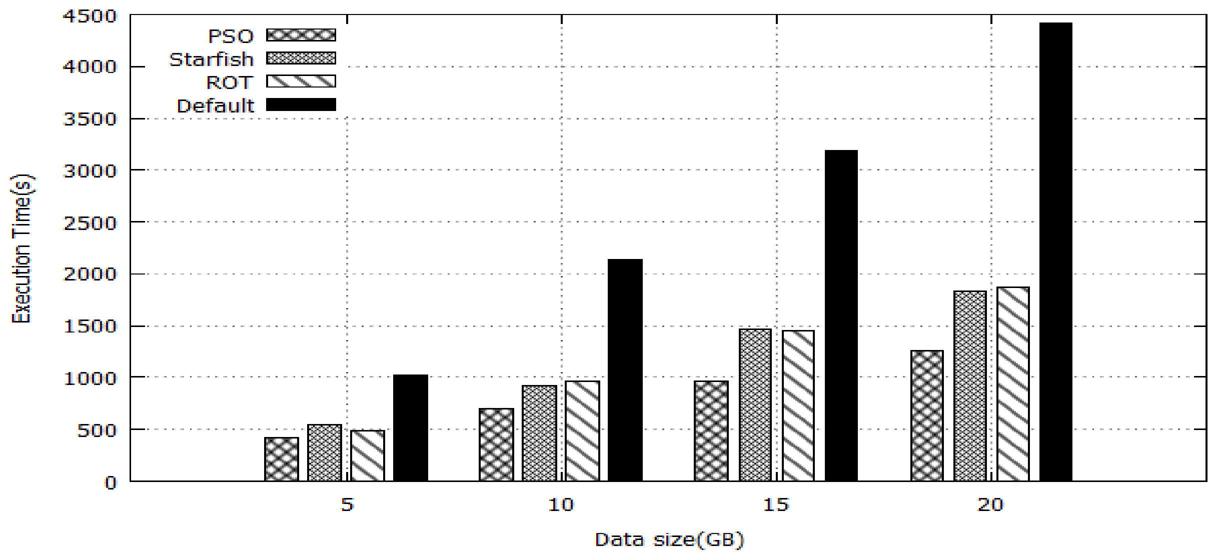

Figure 5.9: The performance of the PSO optimized WordCount application using 8 VMs.

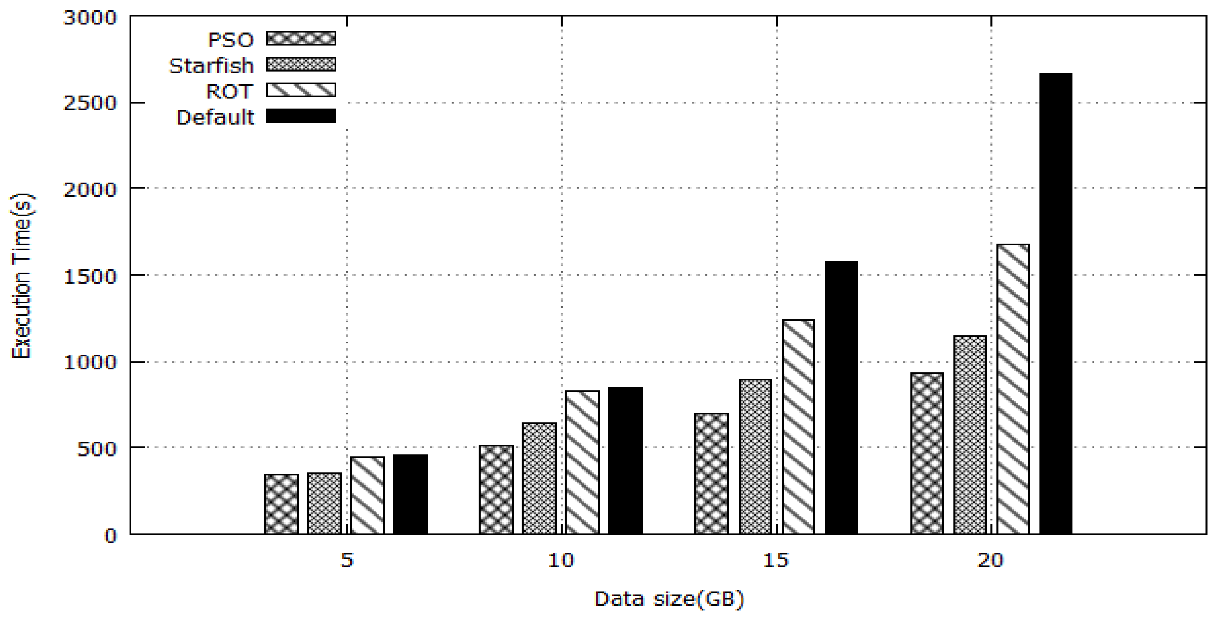

Figure 5.10: . The performance of the PSO optimized Sort application using 8 VMs.

It should be pointed out that the implemented PSO algorithm considers both the underlying hardware resources and the size of an input dataset and then recommends configuration parameter settings for both applications. The ROT work only considers the underlying hardware resources (i.e. CPUs and physical memory) and ignores the size of an input dataset. The Starfish model also considers both the underlying hardware resources and the size of an input dataset. However, Starfish overestimates the number of reduce tasks. For example, Starfish recommended 192 reduce tasks for the WordCount application and 176 reduce tasks for the Sort application on a 20GB dataset. A large number of reduce tasks improves hard disk utilization through task parallelization but generates a high overhead in setting up these reduce tasks in Hadoop. ROT ignores the input dataset size, therefore, the recommended parameter settings of ROT are the same for all the input datasets as shown in Table 5.12. It is worth noting that ROT performs slightly better than Starfish on the WordCount application. This is because Starfish suggests a large number of reduce tasks which generates a high overhead in setting up these reduce tasks, especially in the case of using a small input dataset (e.g. 5GB). Whereas ROT suggests a small number of reduce tasks which are completed in a single wave generating a low overhead in setting up the reduce tasks. ROT estimates the number of reduce tasks based on the total number of reduce slots configured in the Hadoop cluster.

Table 5.12: ROT recommend parameter settings on 8 VMs.

Configuration Parameter name	Value
io.sort.factor	25
io.sort.mb	250
io.sort.spill.percent	0.8
mapred.reduce.Tasks	14
mapreduce.tasktracker.map. tasks.maximum	3
mapreduce.tasktracker.reduce.tasks.maximum	3
mapred.child.java.opts	600
mapreduce.job.shuffle.input.buffer.percent	0.7
mapred.reduce.parallel.copies	20
mapred.compress.map. output	True
mapred.output.compress	False

We have further evaluated the performance of the PSO optimization work on another Hadoop cluster configured with 16 VMs. From Fig.5.11 and Fig.5.12 it can be observed that the PSO work improves the performance of both applications on average by 65% and 86% compared with ROT and the default Hadoop settings respectively. The improvement reaches a maximum of 87% when the input data size is 35GB on the WordCount application. The performance gains of the PSO work over the Starfish model on the WordCount application and the Sort application are on average 20% and 21 % respectively. It is worth noting that the Starfish model performs better than ROT in the case of using 16 VMs. In this case, a large dataset with a size varying from 25GB to 40GB was used. ROT recommends *False* for the *mapred.output.compress* parameter (as shown in Table 5.12). As a result, both applications took a long time in the reduce phase when writing the reduce task outputs into the hard disk. For example, it took WordCount 19 minutes to process the 40GB dataset in the map phase and 61 minutes in the reduce phase following the ROT recommended parameter settings. Whereas it took WordCount

13 minutes to process the same amount of data in the map phase and only 23 minutes in the reduce phase following the Starfish recommended parameter settings. This is because Starfish enabled the *mapred.output.compress* parameter which reduces the overhead in writing the reduce task outputs into the hard disk.

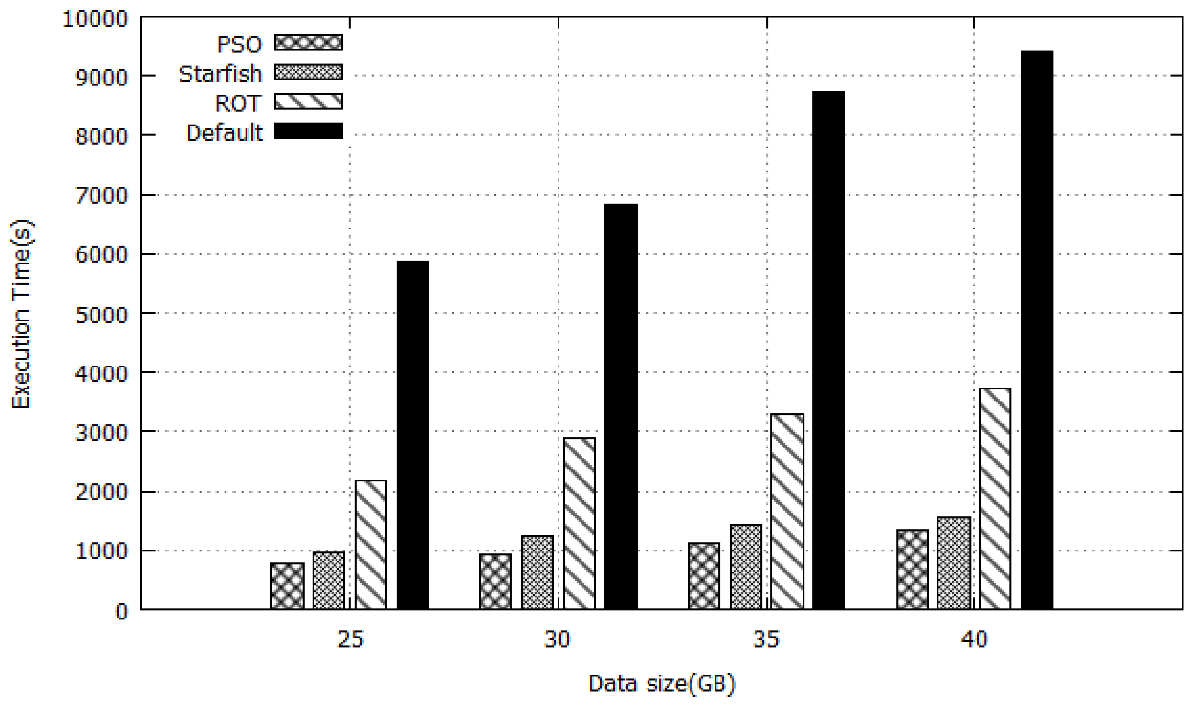

Figure 5.11: The performance of PSO optimized WordCount application using 16 VMs.

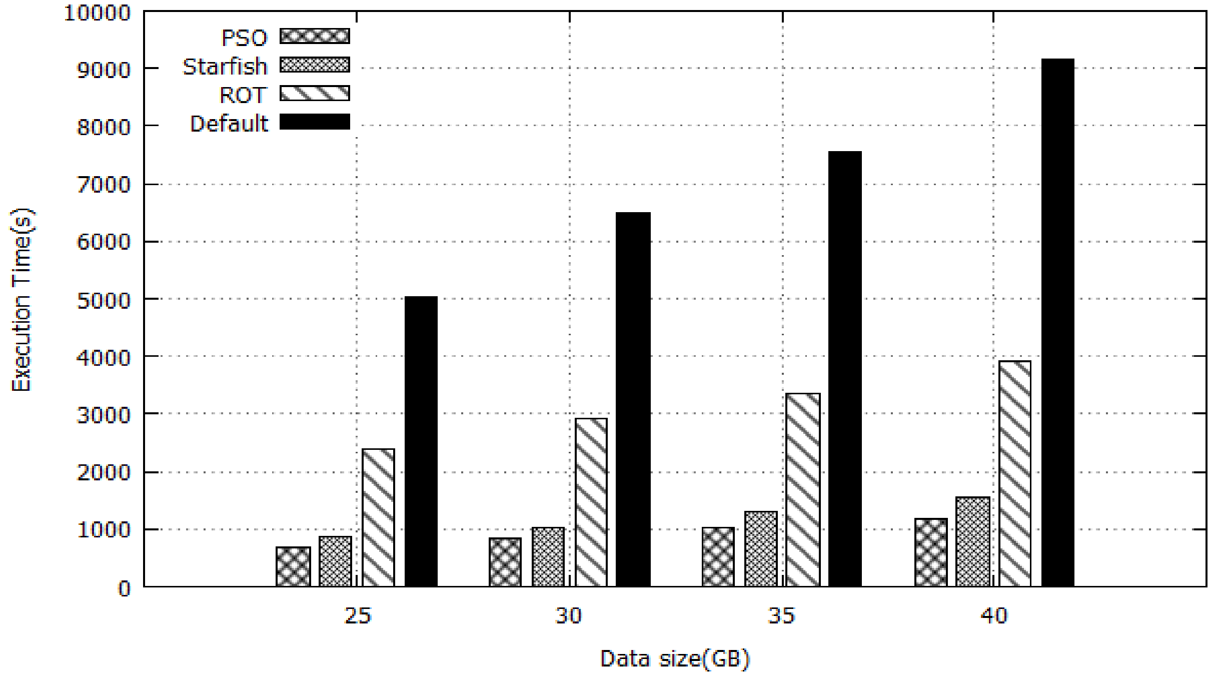

Figure 5.12: The performance of the PSO optimized Sort application using 16 VMs.

5.6 Related Work

In recent years, numerous researches have been carried out to optimize the performance of Hadoop from different aspects. The methodologies of these studies are diverse and range from optimizing Hadoop job scheduling mechanisms to tuning the configuration parameter settings. For example, many researchers have focused on developing adaptive load balancing mechanisms [38]–[41] and data locality algorithms [42]–[45] to improve the performance of Hadoop.

A group of researchers have proposed optimization approaches for a particular type of jobs such as short jobs and query based jobs [46]–[49]. Jahani et al. proposed the MANIMAL model [46] which automatically analyzes a Hadoop program using a static analyzer tool for optimization. However, the MANIMAL model only focuses on relational style programs employing the selection and projection operators and does not consider text-processing programs. Moreover, it only optimizes the map phase in Hadoop. Elmeleegy et al. presented Piranha [49], a system which optimizes short jobs (i.e. query base jobs) by minimizing their response times. They suggested that fault-tolerance facilities are not necessary for short running jobs because the jobs are small and they are unlikely to incur failures. The works presented in [47], [48] focus on optimizing short Hadoop jobs by enhancing tasks execution mechanisms. They optimized task initialization and termination stages by removing the constant heartbeat which is used for the tasks setup and cleanup process in Hadoop. They proposed a push-model for heartbeat communication to reduce delays between the JobTracker and a TaksTracker, and implemented an instance communication mechanism between the JobTraker and a TaskTracker in order to separate message communication from the heartbeat.

Many researchers have also researched into resources provisioning for Hadoop jobs. Palanisamy et al. presented the Cura model [50] that allocates an optimum number of VMs to a user job. The model dynamically creates and destroys the VMs based on the user workload in order to minimize the overall cost of the VMs. Virajith et al. [51] proposed Bazaar that predicts Hadoop job performance and provisions the resources in term of VMs to satisfy user requirements. A model proposed in [52] optimizes Hadoop resource provisioning in the Cloud. The model employed a brute-force search to find optimum values for map slots and reduce slots over the resource configuration space. Tian et al. [53] proposed a cost model that estimates the performance of a Hadoop job and provisions the resources for the job using a simple regression technique. Chen et al. [54] further improved the cost model and proposed CRESP which employs a brute-force search technique for provisioning optimal resources in term of map slots and reduce slots for Hadoop jobs. Lama et al. [55] proposed AROMA, a system that automatically provisions the optimal resources of a job to achieve service level objectives. AROMA builds on a clustering technique to group the jobs with similar behaviors. It employed Support Vector Machine to predict the performance of a Hadoop job and a pattern search technique to find an optimal set of resources for a job to achieve the required deadline with a minimum cost. However, AROMA cannot predict the performance of a job whose resource utilization pattern is different from any previous ones. More importantly, AROMA does not provide a comprehensive mathematical model to estimate a job execution time.

There are a few other sophisticated models such as [12], [13], [15]–[17] that are similar to the proposed work in the sense that they optimize a Hadoop job by tuning the configuration parameter settings. Wu et al. proposed PPABS [15] which automatically tunes the Hadoop framework configuration parameter settings based on executed job profiles. The PPABS framework consists of Analyzer and Recognizer components. The Analyzer trains the PPABS to classify the jobs having similar performance into a set of equivalent classes. The Analyzer uses *K-means++* to classify the jobs and Simulated Annealing to find optimal settings. The Recognizer classifies a new job into one of these equivalent classes using a pattern recognition technique. The Recognizer first runs the new job on a small dataset using default configuration settings and then applies the pattern recognition technique to classify it. Each class has the best configuration parameter settings. Once the Recognizer determines the class of a new job then it automatically uploads the best configuration settings for this job. However, PPABS is unable to find the fine-tuned configuration settings for a new job which does not belong to any of these equivalent classes. Moreover, PPABS does not consider the correlations among the configuration parameters. Herodotou et al. proposed Starfish [12], [13] that employs a mixture of cost model [14] and simulator to optimize a Hadoop job based on previously executed job profile information. Starfish divides the search space into subspaces. It considers the configuration parameters independently for optimization and combines the optimum configuration settings found in each subspace as a group of optimum configuration settings. Starfish collects the running job profile information at a fine-granularity for job estimation and automatic optimization. However, collecting detailed job profile information with a large set of metrics generates an extra overhead. As a result, the Starfish model is unable to accurately estimate the job execution time due to which it overestimates the values for some configuration parameters especially for the number of reduce tasks. As Starfish divides the configuration parameter space into subspaces which may ignore the correlations among the parameters. Liao et al. proposed

Gunther [16], a search based model that automatically tunes the configuration parameters using genetic algorithm. One critical limitation of Gunther is that it does not have a fitness function in the implemented genetic algorithm. The fitness of a set of parameter values is evaluated through physically running a Hadoop job using these parameters which is a time consuming process. Liu et al. [17] proposed Panacea with two approaches to optimizing Hadoop applications. In the first approach, it optimizes the compiler at run time and a new API was developed on top of Soot [56] to reduce the overhead of iterative Hadoop applications. In the second approach, it optimizes a Hadoop application by tuning Hadoop configuration parameters. In this approach, it divides the parameters search space into sub-search spaces and then searches for optimum values by trying different values for parameters iteratively within the range. However, Panacea is unable to provide a sophisticated search technique and a mathematical function which represents a correlation of the Hadoop configuration parameters. Li et al. [18] proposed a performance evaluation model for the whole system optimization of Hadoop. The model analyzes the hardware and software levels and explores the performance issues in these layers. The model mainly focuses on the impact of different configuration settings on a job performance instead of tuning the configuration parameters.

5.7 Summary

Hadoop framework has more than 190 configuration parameters and some of them can a have significant effect on the performance of a Hadoop job. Manually tuning of these parameters is a challenging task and also a time consuming process. This chapter optimizes the performance of a Hadoop job by automatically tuning its configuration parameter settings. The proposed work first employed GEP to build an objective function based on provided training dataset. The objective function represents the correlation among the parameters and also represents job execution duration. It then employed PSO which make use of the objective function to search a set of optimum or near optimum parameter settings. The advantage of the PSO algorithm over other algorithms on problem optimization is that the PSO is rapidly converging towards an optimum solution; however, sometimes it easily traps in local optima. This issue was avoided by using dynamic inertia weight that linearly decreases in every iteration. The performance of the proposed model was extensively evaluated in comparison with the performance of default setting, the ROT settings and the Starfish model. The experimental results showed that the proposed model significantly enhanced the performance of a Hadoop job compared with default settings. Furthermore, the proposed model performed better than both the ROT and the Starfish model.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6, 2004, p. 10.

[2] "Apache Hadoop," Apache. [Online]. Available: http://hadoop.apache.org/. [Accessed: 18-Feb-2015].

[3] M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, "Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data," Smart Grid, IEEE Trans., vol. 6, no. 1, pp. 360–368, Jan. 2015.

[4] M. Khan, M. Li, P. Ashton, G. Taylor, and J. Liu, "Big data analytics on PMU measurements," in Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference on, 2014, pp. 715–719.

[5] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: Mining Peta-scale Graphs," Knowl. Inf. Syst., vol. 27, no. 2, pp. 303–325, May 2011.

[6] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce," Proc. VLDB Endow., vol. 2, no. 2, pp. 1426–1437, Aug. 2009.

[7] A. Pavlo, E. Paulson, and A. Rasin, "A comparison of approaches to large-scale data analysis," in SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009, pp. 165–178.

[8] S. Babu, "Towards automatic optimization of MapReduce programs," Proc. 1st ACM Symp. Cloud Comput. - SoCC '10, p. 137, Jun. 2010.

[9] "Hadoop Performance Tuning." [Online]. Available: https://hadoop-toolkit.googlecode.com/files/White paper-HadoopPerformanceTuning.pdf. [Accessed: 23-Feb-2015].

[10] "7 tips for Improving MapReduce Performance," 2009. [Online]. Available: http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/. [Accessed: 20-Feb-2015].

[11] T. White, Hadoop:The Definitive Guide, 3rd ed. Yahoo press, 2012.

[12] H. Herodotou and S. Babu, "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs.," PVLDB, vol. 4, no. 11, pp. 1111–1122, 2011.

[13] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A Self-tuning System for Big Data Analytics," in In CIDR, 2011, pp. 261–272.

[14] H. Herodotou, "Hadoop Performance Models," 2011. [Online]. Available: http://www.cs.duke.edu/starfish/files/hadoop-models.pdf. [Accessed: 22-Oct-2013].

[15] D. Wu and A. S. Gokhale, "A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration," in 20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013, 2013, pp. 89–98.

[16] G. Liao, K. Datta, and T. L. Willke, "Gunther: Search-based Auto-tuning of Mapreduce," in Proceedings of the 19th International Conference on Parallel Processing, 2013, pp. 406–419.

[17] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir, "Panacea: Towards Holistic Optimization of MapReduce Applications," in Proceedings of the Tenth International Symposium on Code Generation and Optimization, 2012, pp. 33–43.

[18] Y. Li, K. Wang, Q. Guo, X. Li, X. Zhang, G. Chen, T. Liu, and J. Li, "Breaking the boundary for whole-system performance optimization of big data," in Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on, 2013, pp. 126–131.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Neural Networks, 1995. Proceedings., IEEE International Conference on, 1995, vol. 4, pp. 1942–1948 vol.4.

[20] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in Micro Machine and Human Science, 1995. MHS ’95., Proceedings of the Sixth International Symposium on, 1995, pp. 39–43.

[21] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, “Hadoop Performance Modeling for Job Estimation and Resource Provisioning,” Parallel Distrib. Syst. IEEE Trans., vol. PP, no. 99, p. 1, PrePrints, doi:10.1109/TPDS.2015.2405552, 2015.

[22] C. Ferreira, “Gene Expression Programming: A New Adaptive Algorithm for Solving Probles,” Complex Syst., vol. 13, no. 2, pp. 87–129, 2001.

[23] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford, UK: Oxford University Press, 1996.

[24] J. H. Holland, Adaptation in Natural and Artificial Systems. Cambridge, MA, USA: MIT Press, 1992.

[25] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press, 1992.

[26] J. Kennedy, R. C. Eberhart, and Y. Shi, Swarm Intelligence. Elsevier, 2001, pp. 369–392.

[27] K. E. Parsopoulos and M. N. Vrahatis, “Recent approaches to global optimization problems through Particle Swarm Optimization,” Nat. Comput., vol. 1, no. 2–3, pp. 235–306, 2002.

[28] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, “Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems,” Evol. Comput. IEEE Trans., vol. 12, no. 2, pp. 171–195, Apr. 2008.

[29] G. S. Sadasivam and D. Selvaraj, “A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids,” in Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on, 2010, pp. 377–382.

[30] S. Ludwig, “Scalability Analysis: Reconfiguration of Overlay Networks Using Nature-Inspired Algorithms,” in Advances in Intelligent Modelling and Simulation, vol. 422, J. Kołodziej, S. U. Khan, and T. Burczy´nski, Eds. Springer Berlin Heidelberg, 2012, pp. 137–154.

[31] A. Engelbrecht, “Particle swarm optimization: Velocity initialization,” in Evolutionary Computation (CEC), 2012 IEEE Congress on, 2012, pp. 1–8.

[32] B. Jiao, Z. Lian, and X. Gu, “A dynamic inertia weight particle swarm optimization algorithm,” Chaos, Solitons & Fractals, vol. 37, no. 3, pp. 698–705, 2008.

[33] S. Helwig and R. Wanka, “Particle Swarm Optimization in High-Dimensional Bounded Search Spaces,” in Swarm Intelligence Symposium, 2007. SIS 2007. IEEE, 2007, pp. 198–205.

[34] S. Helwig, J. Branke, and S. M. Mostaghim, “Experimental Analysis of Bound Handling Techniques in Particle Swarm Optimization,” Evol. Comput. IEEE Trans., vol. 17, no. 2, pp. 259–271, Apr. 2013.

[35] Z. Li-ping, Y. Huan-jun, and H. Shang-xu, “Optimal choice of parameters for particle swarm optimization,” J. Zhejiang Univ. Sci. A, vol. 6, no. 6, pp. 528–534, 2005.

[36] L. Ai-min, H. Zhang, G. Yang, and X. Lin, “Research and application of multi-objective particle swarm optimization in linear induction motor operating mechanism,” in Electric Power Equipment - Switching Technology (ICEPE-ST), 2011 1st International Conference on, 2011, pp. 291–295.

[37] J. McCaffrey, “Artificial Intelligence:Particle Swarm Optimization,” Microsoft-MSDN Magazine, 2011.

[38] H. Mao, S. Hu, Z. Zhang, L. Xiao, and L. Ruan, “A Load-Driven Task Scheduler with Adaptive DSC for MapReduce,” in Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on, 2011, pp. 28–33.

[39] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, “Job Aware Scheduling Algorithm for MapReduce Framework,” in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 724–729.

[40] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, “Adaptive MapReduce Using Situation-aware Mappers,” in Proceedings of the 15th International Conference on Extending Database Technology, 2012, pp. 420–431.

[41] H.-H. You, C.-C. Yang, and J.-L. Huang, “A Load-aware Scheduler for MapReduce Framework in Heterogeneous Cloud Environments,” in Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 127–132.

[42] M. Hammoud and M. F. Sakr, “Locality-Aware Reduce Task Scheduling for MapReduce,” in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 570–576.

[43] C. He, Y. Lu, and D. Swanson, “Matchmaking: A New MapReduce Scheduling Technique,” in Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, 2011, pp. 40–47.

[44] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters.,” in IPDPS Workshops, 2010, pp. 1–9.

[45] B. Heintz, A. Chandra, and R. Sitaraman, K., “Optimizing MapReduce for Highly Distributed Environments,” 2012.

[46] E. Jahani, M. J. Cafarella, and C. Ré, “Automatic Optimization for MapReduce Programs,” Proc. VLDB Endow., vol. 4, no. 6, pp. 385–396, Mar. 2011.

[47] J. Yan, X. Yang, R. Gu, C. Yuan, and Y. Huang, “Performance Optimization for Short MapReduce Job Execution in Hadoop,” in Cloud and Green Computing (CGC), 2012 Second International Conference on, 2012, pp. 688–694.

[48] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, “SHadoop: Improving MapReduce Performance by Optimizing Job Execution Mechanism in Hadoop Clusters,” J. Parallel Distrib. Comput., vol. 74, no. 3, pp. 2166–2179, Mar. 2014.

[49] K. Elmeleegy, “Piranha: Optimizing Short Jobs in Hadoop,” Proc. VLDB Endow., vol. 6, no. 11, pp. 985–996, Aug. 2013.

[50] B. Palanisamy, A. Singh, and B. Langston, “Cura: A Cost-Optimized Model for MapReduce in a Cloud,” in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 1275–1286.

[51] J. Virajith, B. Hitesh, C. Paolo, K. Thomas, and R. Antony, “Bazaar: Enabling Predictable Performance in Datacenters,” 2012.

[52] K. Kambatla, A. Pathak, and H. Pucha, “Towards Optimizing Hadoop Provisioning in the Cloud,” in Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, 2009.

[53] F. Tian and K. Chen, “Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds,” in 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 155–162.

[54] K. Chen, J. Powers, S. Guo, and F. Tian, “CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds,” IEEE Transcation Parallel Distrib. Syst., vol. 25, no. 6, pp. 1403 – 1412, 2014.

[55] P. Lama and X. Zhou, “AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud,” in Proceedings of the 9th International Conference on Autonomic Computing, 2012, pp. 63–72.

[56] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, “Soot - a Java Bytecode Optimization Framework,” in Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research, 1999, p. 13–.

Chapter 6

Conclusion and Future Work

This chapter concludes the major contributions of the thesis. It also outlines potential opportunities to further improve or extend the work presented in the thesis.

6.1 Conclusion

To put it short, this thesis first evaluated the performance of Hadoop in parallelization of detrended fluctuation analysis algorithm for fast event detection on massive volumes of PMU data [1]. It then built a Hadoop performance model and employed LWLR for job execution estimation and Lagrange Multiplier for resource provisioning [2]. Finally, the thesis presented the research to enhance the performance of Hadoop by automatically tuning its configuration parameter settings.

The PDFA was evaluated in comparison with the original sequential DFA from the aspects of accuracy, scalability and efficiency in computation. Experimental results have shown significant improvements of PDFA over DFA, especially the larger the dataset is, the better performance gain can be achieved using the parallel DFA. This work shows that the Hadoop framework is highly effective in support of data intensive applications, and it scales well with an increasing size of dataset. This work is one of the pioneering works in applying high performance computing techniques to smart grid for which big data has become a critical issue due to the rapid deployment of digital devices such as PMUs, smart meters etc.

However, Hadoop only supports off-line data analytics. In this work, the PMU data was collected from the OpenPDC data concentrator and stored in the Hadoop file system (HDFS). A software agent was implemented for this purpose. It is worth noting that PMU devices generate data in the form of streams. How to employ Hadoop to deal with online (real time) data analytics becomes a research issue which can be considered in the future work.

The second work of the thesis was focused on Hadoop performance modeling. This work mathematically modeled the three core phases of a Hadoop job execution (i.e. the map phase, shuffle phase and reduce phase).

It employed LWLR model for job execution estimation and Lagrange Multipliers technique for resources provisioning. The LWLR model generalization was validated through a 10-fold cross-validation and its goodness of fit was assessed using R-squared. The improved model works in an offline mode and considered a single Hadoop job without logical dependencies. The performance of the improved HP model was extensively evaluated on both an in-house Hadoop cluster and the Amazon EC2 Cloud. The performance was compared with both the HP model and the Starfish model and comparison results showed that the improve HP model outperforms both the HP model and the Starfish model. For resource provisioning, the improved model considered 4 scenarios with a varied number of map slots and reduce slots. The experimental results showed that the improved HP model more cost-effective in resource provisioning than the HP model.

The third research of the thesis was focused on Hadoop performance optimization by automatically tuning its configuration parameter settings. The Hadoop framework has more than 190 tunable configuration parameters that control the flow of a job execution. Some of them are critical to job performance i.e. a small change in one of the parameter values can have a huge impact on the performance of a job when the job runs on the same resources and processes the same amount of dataset. Manually tuning these parameters is a challenging task and also a time consuming process. Moreover, the large number of configuration parameters and the complex inter-connection among the parameters further increase the complexity of manual tuning process. This work employed GEP to build an objective function that represents the correlations among the configuration parameters.

This is a significant step in Hadoop performance optimization. To the best of our knowledge, this is the first work to find the dependencies among Hadoop parameters, albeit only 9 core parameters were considered in this work. Furthermore, PSO was employed to use the objective function to search for optimal or near optimal parameter settings. This work was initially evaluated on a single server machine with 8 VMs and subsequently on 2 server machines using 16 VMs. The performance of the proposed model was compared with default setting, ROT and Starfish model. The experimental results have shown that the presented work improve the performance of Hadoop significantly compared with the default settings. Furthermore, it performed better than both the ROT and the Starfish model.

6.2 Future Work

Although the contributions of the thesis are significant in modeling and optimizing Hadoop performance, a number of works can be explored for future considerations. For examples, the PDFA model collects online historian data from installed PMU through OpenPDC software and stream the data into HDFS. The OpenPDC collect the historian data in .d extension (compress format). The Hadoop MapReduce is unable to process a compressed dataset. Therefore, the PDFA model was manually converted .d format data into .csv format using the OpenPDC historian playback module. The manual process of the PDFA can be made automatic which further improve the competence of the PDFA.

Similarly, the future work opportunities is exist in the improved HP model work, for examples,

- The improved HP model only considers the three core phases i.e. map phase, shuffle phase and reduce phase. These phases can be further divided into sub-phases and then these sub-phases can be mathematically model accordingly. The fine granularity modeling of a job phases can further improve the performance of job modeling process.

- Currently the improved HP model only considers individual Hadoop job without logical dependencies. Modeling multiple Hadoop jobs with execution conditions would be a future work to further enhance the performance of the improved HP model.
- Both the HP model and the improved HP model over-provision resources for a user jobs with large deadlines cases where VMs are configured with large number of both map slots and reduce slots. The reason of over-provisioning of resources is that both the models only consider static overhead of the VMs. Another future work in resources provisioning direction would be to consider dynamic overhead of the VMs involved in running the user jobs to minimize resource over-provisioning.

The job optimization model presented in this thesis has built single objective function for both CPU-intensive jobs and I/O-intensive jobs. A future work in this direction could be to build multiple objective functions, one for each type of job (i.e**.** CPU-intensive, I/O-intensive) and then classify the jobs into CPU-intensive and I/O-intensive classes. The classification can be performed based on the resources utilization. When a user submit a new job, first the job will run on a small dataset for a specific period of time and performance metric (i.e. CPU utilization, memory, disk I/O and network utilization) will be collected online (during execution of the job). The pattern recognition technique can be used to determine the equivalence class of the new job. Once the class of the new job is determined, the relevant objective function can be used to search a set of optimum values of the configuration parameter settings for the new job. K-mean technique can be used for the jobs classification purpose and dstate command can be used to collect online job performance metrics. Similar idea is presented in [3].

References

[1] M. Khan, P. M. Ashton, M. Li, G. A. Taylor, I. Pisica, and J. Liu, "Parallel Detrended Fluctuation Analysis for Fast Event Detection on Massive PMU Data," Smart Grid, IEEE Trans., vol. 6, no. 1, pp. 360–368, Jan. 2015.

[2] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop Performance Modeling for Job Estimation and Resource Provisioning," Parallel Distrib. Syst. IEEE Trans., vol. PP, no. 99, p. 1, PrePrints, doi:10.1109/TPDS.2015.2405552, 2015.

[3] D. Wu and A. S. Gokhale, "A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration," in 20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013, 2013, pp. 89–98