# A Resource Aware Distributed LSI Algorithm for Scalable Information Retrieval

A Thesis submitted for the Degree of

Doctor of Philosophy

By

Yang Liu


Department Electronic and Computer Engineering

School of Engineering and Design

Brunel University

April 2011

# Abstract

Latent Semantic Indexing (LSI) is one of the popular techniques in the information retrieval fields. Different from the traditional information retrieval techniques, LSI is not based on the keyword matching simply. It uses statistics and algebraic computations. Based on Singular Value Decomposition (SVD), the higher dimensional matrix is converted to a lower dimensional approximate matrix, of which the noises could be filtered. And also the issues of synonymy and polysemy in the traditional techniques can be overcome based on the investigations of the terms related with the documents. However, it is notable that LSI suffers a scalability issue due to the computing complexity of SVD.

This thesis presents a resource aware distributed LSI algorithm MR-LSI which can solve the scalability issue using Hadoop framework based on the distributed computing model MapReduce. It also solves the overhead issue caused by the involved clustering algorithm. The evaluations indicate that MR-LSI can gain significant enhancement compared to the other strategies on processing large scale of documents. One remarkable advantage of Hadoop is that it supports heterogeneous computing environments so that the issue of unbalanced load among nodes is highlighted. Therefore, a load balancing algorithm based on genetic algorithm for balancing load in static environment is proposed. The results show that it can improve the performance of a cluster according to heterogeneity levels.

Considering dynamic Hadoop environments, a dynamic load balancing strategy with varying window size has been proposed. The algorithm works depending on data selecting decision and modeling Hadoop parameters and working mechanisms. Employing improved genetic algorithm for achieving optimized scheduler, the algorithm enhances the performance of a cluster with certain heterogeneity levels.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Maozhen Li. With his guidance, invaluable advice and support, I've been encouraged greatly on researching. With the helps from him, I overcame the difficulties no matter how hard they were. I also give my honest and sincere appreciation to the School of Engineering and Design. I achieved the ultimate supports from the school that I've never thought that I could have.

I would like to thank my parents and all my families. 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 today.

I give my sincere gratitude to Suhel Hammoud, Bin Yu, Xiaotian Yang, Nasullah Khalid Alham, Zelong Liu and Yu Zhao. They support and take care of me so much in my study and in my life. Without them, things would not go this smoothly. I also should say thanks to the friends who have provided not just academic but moral support for my research.

# Author’s Declaration

The work described in this thesis has not been previously submitted for a degree in this or any other university and unless otherwise referenced it is the author’s own work.

# Statement of Copyright

# Publications

The following papers have been published (or have been submitted for publication) as a direct or indirect result of the research discussed in this thesis.

**Journal papers:**

- X. Liu, M. Li, Y. Liu and M. Qi, OBIRE: Ontology based Bibliographic Information Publication and Retrieval, Int'l Journal of Distributed Systems and Technologies, vol.1, no. 4, pp.59-74, October-December 2010.
- Y. Liu, M. Li, N. K. Alham, S. Hammoud, HSim: A MapReduce Simulator in Enabling Cloud Computing, Future Generation Computer Systems (FGCS), the International Journal of Grid Computing and e-Science, Elsevier Science (accepted for publication)
- N. K. Alham, M. Li, Y. Liu and S. Hammoud, A MapReduce-based Distributed SVM Algorithm for Automatic Image Annotation, Computers & Mathematics with Applications, Elsevier (accepted for publication)
- Y. Liu, M. Li, N. K. Alham, S. Hammoud, A Resource Aware Distributed LSI for Scalable Information Retrieval, Information Processing and Management, Elsevier Science (under review)
- N. K. Alham, M. Li, Y. Liu and S. Hammoud, Parallelizing Multiclass Support Vector Machines for Scalable Image Annotation, Neurocomputing, Elsevier Science (under review)
- N. K. Alham, M. Li, Y. Liu and S. Hammoud, A Resource Aware Parallel SVM for Scalable Image Annotation, Parallel Computing, Elsevier Science (under review)
- G. Caruana, M. Li, N.K. Alham, and Y. Liu, A Parallel Support Vector Machine for Large Scale Spam Filtering, information Sciences, Elsevier Science (under review)

**Conference papers:**

- Y. Liu, M. Li, S. Hammoud, N.K. Alham, M. Ponraj, Distributed LSI for Information Retrieval, Proc. of IEEE FSKD'10, pp. 2978-2982.
- N. K. Alham, M. Li, S. Hammoud, Y. Liu, M. Ponraj, MapReduce-based Distributed SMO for Support Vector Machines, Proc. of IEEE FSKD'10, pp. 2983-2987.
- S. Hammoud, M. Li, Y. Liu, N. K. Alham, Z. Liu, MRSim: A Discrete Event based MapReduce Simulator, Proc. of IEEE FSKD'10, pp. 2993-2997.

# Chapter 1

# Introduction

This chapter briefly describes the background to the problems investigated in this thesis, motivation of work, major contributions and the structure of the thesis.

## 1.1 Background

In recent years, the amount of information resources is greatly increasing which results in generating mass volume of electronically stored data. The volume and scope of data are increasing dramatically which forms huge document corpus or databases. The current search engines are normally represented by World Wide Web. For instance the largest search engine Google claims that it processes over eight billion [38] pages and more than twenty PB (PetaByte) data processed per day while other search engines such as Yahoo, Bing also deal with enormous volume of data. The Information Retrieval (IR) technologies are not only needed by the larger search engines but also required by other organizations such as companies, universities and hospitals. However the current information retrieval approaches are in most cases inefficient to access the information required by the users [106]. Therefore, to retrieve the information efficiently, several improved information retrieval technology have been developed.

Information retrieval systems are mainly based on the matching of users' queries [39] and the relevant information stored in the database. The traditional IR technologies employed by most search approaches are mainly based on the keyword matching. Matching with keyword, data is usually indexed by attributes such as author, date, abstract and keywords. To perform a search process, firstly a user inputs a number of keywords which represent the required information then the algorithm compares the input keywords to the indexed attributes of the data. The system response is based on matching the user's words and the stored indexed information. However, there are

mainly two issues which impact hugely the performances of current keyword matching based IR technologies. Firstly the performance of keyword matching greatly deteriorates with the increase of volume of data. Secondly keyword cannot describe the semantic relationships exist in the data accurately [110]. Therefore the accuracy of retrieved results is affected significantly due to the lack of accurate of representation the semantic content of the information. To solve the above issues, researchers combine ontology technologies [107] [108] [109] with keyword matching. Their studies show that based on the knowledge expansion, the issue of lack of semantic relationships can be solved. However, considering the efficiency and accuracy aspects, the approach cannot supply a satisfied performance [112] [113]. And also as more and more ontologies available online, it becomes more difficult to find a suitable ontology that meets a user's needs [111]. Latent Semantic Indexing (LSI) [6] [7] [8] [11] [22] [24] [29] [36] [94] has been developed to perform intelligent IR searches [95] based on statistics and algebra to overcome the deficiencies associated with keyword matching retrieval techniques.

## 1.2 The Issues Solved by LSI

LSI has been widely used in information retrieval [9] [10] [12] such as image processing [40] [98], audio and video retrieval systems [41] [42], and multi-language retrieval [43]. LSI is based on the concept that latent structures exist among a number of documents. Building on Vector Space Model (VSM) [45] [89], LSI generates a Term-Document (T-D) matrix after removing all punctuations and stop words which has no distinctive semantic meaning from a document. LSI employs a truncated Singular Value Decomposition (SVD) [4] [17] [92] [93] [97] to convert the keywords domain of the original document corpus to a conceptual domain by decomposing the higher dimensional sparse [90] [96] matrix to a lower dimensional approximate matrix [91] so that the latent semantic relationships among the words and documents are highlighted and the problems of polysemy and synonymy [44] are solved. The results retrieved by LSI are based on the similarities between query and documents.

The similarity is normally measured by calculating the cosine value of the two vectors which represent a query and a document.

LSI is regarded as a good replacement to the traditional keyword matching IR technologies. Traditionally the most fundamental issues in IR are the problems of synonymy and polysemy [44]. Synonymy is where several different words may express one concept and the words of query may not match those in the relevant documents. For instance the word *van* and *vehicle* have the similar meaning. However, when a user input query with word *van* to search using keyword matching, the document with word *vehicle* may not be returned as the field of *vehicle* is not covered in the query. Even though a document with world *van* has been returned, the content of the document may not belong to a topic describing automobiles however some other content which may just simply involve a word *van*. Polysemy is where words may have different meanings and the words of query may match those in the irrelevant documents. For instance the word *bank* has different interpretations in the fields of finance and nature. It can be used to describe a financial intermediary or can be used to describe the land alongside or sloping down to a river or lake. Therefore when a user searches the word *bank* with a financial meaning, the traditional keyword matching may return incorrect results. Therefore, the precision and recall is significantly affected by the synonymy and polysemy. Here the recall is defined as the ratio of the number of the relevant documents retrieved to the number of relevant documents in the database. The precision is defined as the ratio of the number of relevant documents retrieved in the total number of documents retrieved with a query of a user [44].

LSI addresses the problems of synonymy and polysemy by analyzing the semantic relationships among terms and documents. LSI assumes that there must be certain kinds of latent semantic structures, which are hidden in the context due to the existing polysemy and synonymy within the documents and corpus. Therefore, LSI does not use simpler keyword matching however it is based on statistics and algebraic

calculation to discover the latent semantic relationships and underlying semantic structures in the documents. Comparing terms used across documents, it has been recognized that certain groups of terms frequently appear among a number of documents. However, the groups of terms are barely appeared in the other documents. Thus from the semantic phase, the terms and documents with the terms can be regarded as semantically close enough to each other while the terms and documents without the terms are considered as semantically distant [6]. Practically, LSI returns documents which have similar meaning, even though the keywords input by users may not appear in the target documents.

## 1.3 Problem Statements

LSI suffers from scalability problems especially in processing massive document collections due to SVD which is considered to be computationally intensive. Therefore, several techniques have been proposed to enhance the performance of LSI. Gao [14] and Bassu [3] combined the clustering algorithm *k-means* [30] [31] and LSI to reduce the overhead (large executing time consumed) of typical LSI. These approaches show enhancement in performances however the overhead of *k-means* with large document collection are not considered. An alternative approach is to distribute the computation of LSI among nodes in a cluster environment using the Message Passing Interface (MPI). Seshadri and Iyer [28] proposed a parallel SVD clustering algorithm using MPI. Documents are split into a number of subsets. Each subset of the documents is clustered by a participating node in the cluster. However, The MPI approaches mainly target on homogeneous computing environments with limited support for fault tolerance and incur large inter-node communication overhead when shipping large date across the cluster. Currently heterogeneous computing environments are increasingly being used as platforms for resource intensive distributed applications. One major challenge in using a heterogeneous environment is to balance the computation loads across a cluster of participating computer nodes.

This thesis presents MR-LSI (MapReduce based LSI), a distributed LSI for high performance and scalable information retrieval. MR-LSI improves current approaches by focusing on three aspects. Firstly, MR-LSI employs *k-means* to cluster documents into a number of subsets of documents to reduce the complexity of SVD in computation [18] [20] [37]. Second, MR-LSI builds on MapReduce [2] [5] [23] [33] [35] [76] to distribute the computation of LSI among a number of computers of which each computer only processes a subset of documents. MapReduce has become a major enabling technology in support of data intensive applications. MapReduce has built-in fault tolerance [88] and handles I/O operations effectively which reduces communication overhead significantly. Finally, two types of resource aware load balancing schemes based on both static and dynamic factors are designed to optimize the performance of the MR-LSI algorithm in heterogeneous computing environments. In order to evaluate the effectiveness of the resource aware MR-LSI algorithm in large scale MapReduce environments and the performance of load balancing strategies [62] [68], a Hadoop framework simulator HSim has been developed. It can accurately simulate the behaviors of the framework so that several studies have been done based on the simulator.

## 1.4 Motivation of Work

It has been widely recognized that LSI suffers from scalability problems in processing massive document collections due to SVD which is considered to be computationally intensive which can be represented by $O(m \times r^2)$ where $m$ is the number of documents and $r$ is the rank of T-D matrix [13] [26] [99]. A combination of clustering algorithm *k-means* with LSI is proposed in [3] [14] to reduce the overhead of typical LSI. However the overhead of *k-means* dealing with large document collection is not considered which affects the performance of the algorithm hugely. Thus, an approach should be considered to solve the large overhead issue by involving the *k-means* algorithm combining with LSI. Current literature shows a number of approaches have been proposed in speeding up LSI process in computation

[3] [14] [16] [19] [21] [25], the scalability of these approaches still remains a challenging issue due to the lack of an effective load balancing scheme in utilization of heterogeneous computing resources. The unbalanced load issue can deteriorate the performance of algorithms with LSI. Therefore combining both speeding up the computation of LSI and load balancing, an efficient distributed LSI algorithm should be designed.

This thesis presents a distributed LSI algorithm based on the MapReduce model. One of the most popular implementations of MapReduce model, Hadoop framework becomes popular due to its remarkable characteristics. However, the large number of configuration parameters of Hadoop brings a number of challenges to users to decide on a set of parameters that are crucial for achieving high performances. It is impractical to build up a Hadoop cluster which contains a large number of nodes to evaluate performance of a MapReduce based algorithm. These challenges motivate the desire to have a Hadoop environment simulator which can be used to tune the performance of a Hadoop cluster and analyze the behaviors of Hadoop applications.

Hadoop framework based on MapReduce has become a major enabling technology in support of data intensive applications, which facilitate to process data in a distributed computing environment. Hadoop framework has a number of processing units called Map instances (*mappers*) and Reduce instances (*reducers*) [35]. As *mappers* and *reducers* are controlled by TaskTracker, therefore they work independently without communicating with each other, which is different from traditional distributed computing systems such as MPI. Therefore a notable feature of the Hadoop implementation of MapReduce framework is the ability to support heterogeneous environments. However, in the current version of Hadoop framework lacks of an effective load balancing scheme for utilizing resources with varied computing capabilities. This challenge motivates this work to balance the loads among *mappers* in a dynamic computing environment with considering the interactions of a number of factors including Hadoop parameters, load of IO system and load of processors.

## 1.5 Major Contributions

The main contribution of the thesis is speeding up LSI in Hadoop distributed computing environments by combining the clustering algorithm *k-means* to improve the performance of the typical standard LSI algorithm. Load balancing strategies are deployed to significantly enhance the performances of the algorithm. The following descriptions are the detailed contributions presented in this thesis:

1. To facilitate the analysis of Hadoop framework behaviors, HSim, a Hadoop environment simulator is designed and implemented. HSim aims to accurately simulate the behaviors of Hadoop framework. The current version of HSim modeled and simulated Hadoop framework from four phases. The first phase is node phase which contains parameters of processor, memory, hard disk, network interface, Map instances and Reduce instances. The second phase is the cluster specifications including parameters of number of nodes, configuration of nodes, routers, job queue, and job schedulers. The third phase has the parameters to control the behaviors of above components, in which is including the size of data chunks, JVM reuse, sort factor, virtual memory, the number of copying threads, data spilled threshold. The last phase is the functions and parameters of the simulator itself including simulating speed, system clock, accuracy levels and system reporter. HSim supports to simulate both homogeneous and heterogeneous Hadoop computing environment. Additionally HSim can be adopted to create static and dynamic environments based on the interactions among 'HDD' component, 'CPU' component and 'LoadGenerator' component. Based on the above characteristics, HSim can simulate types of MapReduce jobs. To validate the accuracy, reliability and performance of HSim, several published MapReduce applications based on Hadoop framework are simulated. The validation of HSim follows a two step process. In the first step, HSim is validated against published benchmark results. In the second step, a physical Hadoop environment is set up to evaluate the performance of HSim using our Hadoop applications. The

comparative results show high accuracy and stability of HSim in simulating Hadoop applications.

2. The combination of *k-means* and LSI to speed up the performance of typical LSI has been implemented. To solve the overhead issue brought by *k-means*, which remain a research issue, MR-LSI distributes *k-means* and LSI using Hadoop framework.
   a) MR-LSI distributes the *k-means* algorithm and LSI using Hadoop framework, which enhances the performance of typical LSI significantly when processing large document collection.
   b) The scalability of MR-LSI has been studied using HSim. The impacts of tuning the cluster parameters for MR-LSI are analyzed in details.
   c) A static load balancing strategy based on genetic algorithm has been proposed which considers the heterogeneous environment with various computing resources.
3. The work also considers the load balancing issue in dynamic Hadoop distributed computing environment. A dynamic load balancing strategy for Hadoop framework has been put forward.
   a) The dynamic load balancing strategy designed for Hadoop framework has been proposed. Comparing to the other load balancing solutions such as computing ratio based scheduler strategy, it modeled the characteristics of Hadoop framework in dynamic environment which the load of processors and hard disks are following certain distributions. The evaluation shows it outperforms the other schedulers and enhances the performance of the cluster significantly.
   b) Comparing to a number of established dynamic load balancing strategies with fixed window size (the time interval of executing load balancing algorithm), the algorithm has dynamically changed window sizes which is fully determined by the algorithm itself.
   c) The traditional genetic algorithm has large overhead due to its iterations. Though a number of works claim that controlling the iterations within a small number can still gain optimized solutions, it is not suitable in a complex

dynamic environment in a Hadoop cluster. The work generates a way to reduce the number of iterations of genetic algorithm with considering the characteristics of Hadoop framework, which can significantly enhance the performance of the dynamic load balancing algorithm.

d) Dynamic load balancing strategy is evaluated with both fixed window size and dynamic window size, computing ratio based strategy. The results show that the work significantly improves the performance of the cluster.

## 1.6 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 is a literature review. Section 2.1 and Section 2.2 introduce the basics of LSI algorithm based on vector space model and *k-means* algorithm. This is essential to understand the knowledge which this thesis is based. Section 2.3 introduces the Hadoop framework based on MapReduce model. Section 2.4 discusses the related work of the thesis. Section 2.5 concludes the chapter.

Chapter 3 is dedicated to the design and implementation of the Hadoop simulator HSim. Section 3.1 describes the modeling of parameters in Hadoop framework from four aspects. Section 3.2 gives the details of the design of the simulator. In section 3.3 a number of validations have been down using the computing environments and results of published benchmarks and customized experiments. Section 3.4 concludes the chapter that HSim is provided to be suitable for simulating Hadoop framework.

Chapter 4 presents the MR-LSI algorithm which aims for scalable information retrieval. Section 4.1 describes the design and implementation of MR-LSI in detail. Section 4.2 describes the design and implementation of the static load balancing algorithm in details. Section 4.3 and 4.4 gives the experimental and simulation results of MR-LSI. Section 4.5 concludes the chapter.

Chapter 5 proposes a dynamic load balancing strategy for Hadoop framework to enhance the performance the cluster. Section 5.1 reviews the current load balancing in Hadoop framework. Section 5.2 gives the details of the design of the dynamic load balancing algorithm. Section 5.3 presents the simulation environment. Section 5.4 simulated and evaluated the algorithm and shows the performance compared to some other load balancing strategies. Section 5.5 concludes the chapter.

Finally, Chapter 6 summarizes the contributions of the thesis and proposes directions for future work.

# Chapter 2
# Literature Review

## 2.1 LSI

In the Vector Space Model (VSM) [45], if a document corpus has a number of n keywords, then an n-dimension vector can be built up of which each dimension represents combination of keywords for one document. The documents of the corpus and the queries can be represented by the vectors by the concept of VSM based on the weight of the key words. It is quite obvious that the greater the weight is, the more important the word is. Therefore, in one vector if the weight equals to or greater than 1, it means the word appears in a document. Otherwise if the weight is 0, it means the word does not appear in a document. Based on these vectors, by calculating the cosine values of query and document, the similarities can be measured and useful document collections can be retrieved. VSM abstracts the documents to be vectors and does information retrieval by mathematical computation. Thus VSM does not do any simpler traditional keywords matching.

Though VSM opened up a new way of text mining technologies, there are still some drawbacks existing in the model. The first point is that the key words are assumed to be independent without any relationship. However, there might be certain kinds of relationships among the keywords of documents, which means that VSM is not suitable enough to deal with the associated keywords. The second point is if the scale of the vector space is too large, the processing speed will become highly considerable. As an improvement of VSM, LSI uses the terms processed by statistics to index the documents. Therefore the semantic relationships among the term-term and document-document are highlighted. It also reduces the impacts caused by polysemy and synonymy.

LSI processes the relationships among the terms and documents based on the concept of VSM. Additionally LSI assumes that there must be certain kinds of latent semantic structures, which are hidden in the context because of polysemy and synonymy existing within the documents and corpus. As a result, LSI does not use simpler keyword matching but uses a way of statistics computation to discover the latent semantic relationships. The core computation of LSI is to do the SVD (Singular Value Decomposition) operation on the formed Terms-Documents matrix. And then LSI keeps a pre-given number of largest singular values and corresponding $U$ and $V$ matrices to form a new approximate matrix, which can represent the original matrix approximately. Thus the original terms-documents matrix removes the unnecessary noises and reduces the density of the original matrix, which can reduce the computing complexity of the future computation. The detail of the LSI algorithm is given below.

The terms and documents of a document collection could form a T-D (Terms and Documents) matrix $A_{m\times n}$ where *m* is the number of the terms and *n* is the number of documents.

$$A=[a_{ij}] \quad 1\le i\le m, \quad 1\le j\le n$$

The original matrix $A_{m\times n}$ can be factored into the product of three sub matrices using SVD (Singular Value Decomposition):

$$A=U\sum V^T$$

$\sum$ is a diagonal matrix. The diagonal elements in the matrix are the singular values of matrix $A_{m\times n}$ in descending order. The matrices $U$ and $V$ are orthogonal and normalized which satisfy the equation:

$$U^TU=V^TV=I.$$

LSI computes a low rank approximation to $A_{m\times n}$ using a truncated SVD. The first $k$ elements which are larger than a certain value $\tau$ will be kept and the values of the rests of the diagonal elements $(r-k)$ will be set to zero in the matrix $\sum$. Simultaneously matrices $U$ and $V$ will be truncated to be $U_k$ (First $k$ columns

are kept.) and $V_k$ (First $k$ rows are kept). Thus the original T-D matrix $A_{m\times n}$ will be presented by an approximate matrix $A_k$ with $\sum_k$, $U_k$ and $V_k$:

$$A_k = U_k \sum_k V_k^T$$

The submitted query $q$ will be processed by equation to gain $q_v$:

$$q_v = q^T U_k \sum_k^{-1}$$

Thus to compare the similarities of query and documents can be measured by calculating the cosine values of vector $q_v$ and document $D_j$.

$$\cos\theta_j = \frac{q_v \bullet D_j}{\| q_v \|_2 \| D_j \|_2}$$

Thus if the value of $\cos\theta_j$ is greater than certain given threshold $\sigma$, the document $D_j$ is the target document. Thus the set of $D_j$ can be represented by:

$$D_k = \{d_j \mid \cos\theta_j = \cos(q_v, D_j) \geq \sigma\}$$

LSI does not do keywords matching simply compare to the other traditional text mining technologies, in which due to the polysemy and synonymy, the semantic relationships of terms and documents are hidden deeply in the context. However, based on the SVD computation, LSI can form an approximate matrix from the original terms-documents matrix. The new matrix reduces the so called 'noise' and highlights the semantic relationships of terms and documents.

## 2.2 K-means

*K-means* [30] [31] is a clustering algorithm based on calculating distances between centroids and points (vectors). It calculates the Euclid Distance between vectors as the criterion function of clustering. The following steps represent how the algorithm works.

1. At the beginning, the algorithm selects several (number of *k*) points randomly from the input vectors to be the initial centroids.

2. It calculates the distances between the points of input vectors and the initial centroids. And then each point is clustered to the sub-cluster of which the centroid is closest to the point.
3. Acquire the new centroids of newly formed sub-clusters by calculating the average value of points which are in the same sub-cluster.
4. Execute 2 and 3 repeatedly. After several iterations if the centroids of clusters are not changed any more, then the algorithm can be regarded as finished.

The workflow of *k-means* algorithm is described below:

Input: Number of clusters to be clustered (*k*)

Data set including *n* vectors

Methods:

1. For the input data set, choose number of *k* vectors randomly as the initial centroids.
2. Calculate the distances between the vectors of input data set and the initial centroids.
3. According to the distances, assign each vector to the cluster with the shortest distance from it.
4. Calculate the average value of the vectors in the sub cluster as the new centroid.
5. Using the new centroids, re-cluster the vectors.
6. Repeat 3, 4, and 5.
7. Until the centroids of sub-clusters are stable. Algorithm is finished.

During the computation, the distances among vectors can be measured by Euclid Distance which is expressed as:

$$d(x,y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

where $x_i$ is the coordinates of the points and $y_i$ is the coordinates of the centroids.

From the above descriptions we know that *k-means* has strong abilities to process vector based clustering jobs. It offers convenient and flexible ways to achieve satisfied results. As *k-means* shows strong self-adaptabilities, thus during the whole computation of *k-means* algorithm the only factor should be noticed is the number of chosen centroids *k*.

## 2.3 Hadoop Framework Based on MapReduce

MapReduce [2] [5] [23] [33] [35] [48] is a distributed programming model for data intensive tasks which has become an enabling technology in support of Cloud Computing. Programmatically inspired from functional programming, at its core there are two primary features, namely a map and a reduce operation. From a logical perspective, all data is treated as a Key (K), Value (V) pair. Multiple mappers and reducers can be employed. At an atomic level however a map operation takes a {K1, V1} pair and emits an intermediate list {K2, V2} pairs. A reduce operation takes all values represented by the same key in the intermediate list and processes them accordingly, emitting a final new list {V2}. Whilst the execution of reduce operations cannot start before the respective map counterparts are finished, all map and reduce operations run independently in parallel. Each map function executes in parallel emitting respective values from associated input. Similarly, each reducer processes keys independently and concurrently. Figure 2.1 shows the structure of the MapReduce model. Popular implementations of the MapReduce model include Mars [46], Phoenix [47], Hadoop [2] [5] [33] [35] and Google's implementation [48]. Among them, Hadoop has become the most popular one due to its open source feature.

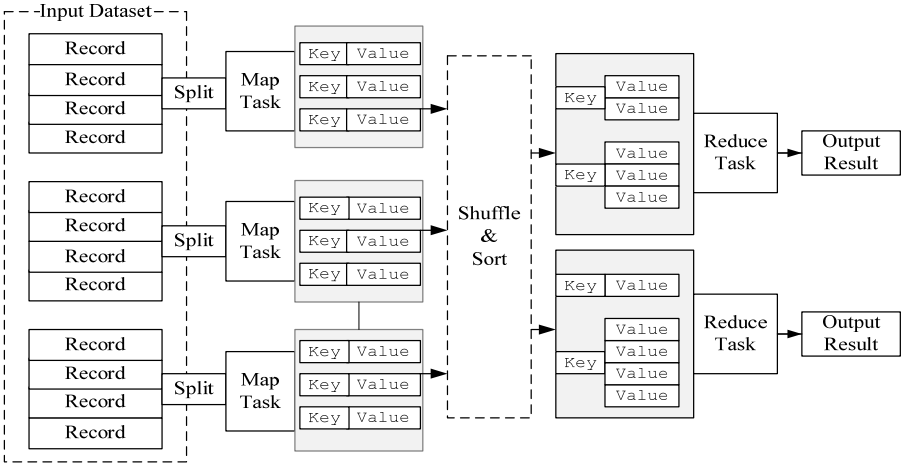


Figure 2.1: The MapReduce model.

## 2.4 Optimization Techniques

Optimization techniques can help to enhance the performance of solutions. The algorithms of optimization have been considered and developed during a long period. Optimization consists in trying variations on an initial concept and using the information gained to improve on the idea. Many optimization problems from the industrial engineering world, in particular the manufacturing systems, are very complex in nature and quite hard to solve by conventional optimization techniques [114]. At present, several optimization algorithms have been widely used in a number of fields. Neural network [116] [120] is one of the algorithms. It is a complicated network system that can realize parallel disposing and nonlinear transformation for information by simulating the way of human cerebral nerves to dispose information. It has a favourable ability to learn itself, adapt itself, associate and recollect, process parallel etc [115]. However, neural network has several drawbacks which may reduce the performance. It can be easily to fall into the local optimum. And also the convergence speed of neural network is quite slow [116]. Moreover, in neural network several important factors such as the structure of network, momentum factor and the training ratio are frequently based on experiences of researchers. These factors highly affect the performances of neural network in terms of the training speed and the disposing ability [117]. Ant Colony Algorithm [121] is another optimization algorithm

which is based on new heuristic biological modeling method. It imitates the behaviors of real ant colony. In the animate nature ants have the ability to find out the food from the nest in the shortest path without any visible reminder. The core of the algorithm is to find the optimal path based on processing of the pheromone left by ants. Ants release the pheromone in the path. The other ants can perceive the pheromone in certain range and their behaviors will be affected. The pheromone will accumulate along with the number of ants passing through the path. As a result, the following ants have higher chance to select the paths with more pheromone [118]. The ant colony algorithm has the ability of processing in parallel and searching in global. However, it has two issues which may affect the performance of the algorithm. The first one is the ant colony algorithm can be easily to fall to local optimum. The second one is that the convergence speed is slow.

Compared to the above discussed algorithms, Genetic Algorithm has several characteristic which can help to avoid the issues of local optimum and slow convergence speed. Genetic algorithm is an adaptive, heuristic and stochastic searching algorithm which is based on the idea of evolutions in natural selection and inheritance during biology circles. The algorithm is widely used on solving complex problems such as function optimization, image processing, classification, machine learning and so on. And also it is proved that the genetic algorithm has strong robustness and global parallel searching [119]. Genetic algorithm is mainly consisted by the following parts.

1. Coding: It models a problem with mathematical model. Thus the computer can parse the coded data and process it further. Binary coding is frequently used in most of the cases of genetic algorithm. However, considering different conditions of the problem to be solved, the other coding approaches such as decimal coding can be used
2. Initial population: A set of initial solutions are involved as the first population which is to be evolved in the algorithm.
3. Genetic operators: it has a series of components which are selection, fitness,

crossover and mutation. Selection selects the individuals which will be evolved. The most popular approach for selection is Russian roulette wheel method which is based on probability. Fitness is involved to evaluate the quality of an individual. In this thesis, mean square error (MSE) is the fitness function. Thus the lower the fitness is, the better the individual is. Crossover recomposes the homologous chromosomes via mating to generate new chromosomes. The generated offspring inherit the basic characteristics of their parents. Some of them may adapt to the fitness function better than their parents did, so they may be chosen as parents in next generation. Thus the diversity of the chromosomes could be maintained which results in avoiding local optimum. Mutation could mutate genes in a chromosome based on smaller probabilities so that the searching space can be expanded. As a result, the local optimum can also be avoided.

4. Stop: When certain requirement of the solution is satisfied, the algorithm stops and outputs the best or optimal result.

Thus compared to the neural network and ant colony algorithms, genetic algorithm is able to adapt to the complex problems quite well. And also it can avoid the local optimum issue which exists both in neural network and ant colony algorithms. Thus, this thesis proposes load balancing strategies based on the genetic algorithm due to its remarkable characteristics.

## 2.5 Related Work

### 2.5.1 MapReduce Simulator

Few existing MapReduce simulators are available and MRPerf [49] [50] is a representative one. The MRPerf can serve as a design tool for MapReduce infrastructure, and as a planning tool for making MapReduce deployment far easier via reduction in the number of parameters that currently have to be manually tuned. From the published testing results, MRPerf shows its high accuracy in simulating the impacts of network topologies due to its adoption of NS2 [51] for network simulation.

However it should be pointed out that although MRPerf achieves high accuracy in simulating behaviors related to the underlying networks, it can simulate limited behaviors of Hadoop framework. The behaviors of Hadoop are affected by a large number of parameters.

For example, in the Map phase, the performance of Map instances is highly coupled with the current states of node processors, buffers, hard disk and networks. When thresholds are reached, certain components may be interrupted to guarantee the performance and synchronizations. In the Reduce phase, the performances of Reduce instances are highly depended on the current IO states. The copying, shuffling and sorting procedures are quite dynamic based on the current system states. MRPerf does not simulate these real time interactions accurately due to its heavy dependencies on the estimations of the values of parameters. The major limitations of MRPerf are listed below:

- The processing resources for each user are fixed in MRPerf. However, resources in a Hadoop environment are dynamically changing and are usually shared by a number of users dynamically.
- MRPerf does not simulate the exact behaviors of Map and Reduce phases. In a Map instance, the spilled data will be kept writing into buffer while Map task is running. When the occupied size of the buffer is less than a certain threshold, the in-memory data is also kept spilling into hard disk simultaneously. Due to the highly uncertain real time states of the system, this mechanism significantly affects the number of spilled files which will further affect the IO behaviors. MRPerf simply ignores these procedures and uses a pre-defined data value.
- If the occupied size of the buffer is larger than a certain threshold, the CPU processing will be blocked until the whole content in buffer is flushed. This event can also affect system behaviors but MRPerf does not consider this.
- In the Reduce phase, MRPerf still performs a simple simulation to start reduce tasks simultaneously due to lack of accurate simulations in Map phase.
- Another drawback of MRPerf is that it only supports homogeneous environment,

but Hadoop can be applied to both homogeneous and heterogeneous environments.

These approximations and simplifications in terms of parameterization cannot reflect real world Hadoop applications. MRPerf was validated using TeraSort, Search and Index [49] [50]. All these three algorithms do not involve complex behaviors of Hadoop framework when the tests were carried out in a homogeneous environment. So the estimations and simplifications of MRPerf did not affect much of its accuracy. It would become a problem when using MRPerf to simulate complex behaviors of Hadoop.

The limitations of MRPerf motivated the work on HSim. Our focus in HSim is to accurately simulate the behaviors of Hadoop framework. Using HSim, the performances of Hadoop applications can be studies from a number of angles including the impacts of the parameters on the performance of a Hadoop cluster, the scalability of a Hadoop application in terms of the number of nodes used, and the impact of using heterogeneous environments. HSim complements the design of MRPerf in that HSim focuses on simulating the Map and Reduce behaviors of Hadoop, and MRPerf focuses on the impact of network topologies of Hadoop.

### 2.5.2 Distributed LSI

The current research efforts in speeding up LSI computation generally fall into two approaches. One approach combines LSI with clustering algorithms such as *k-means* [16] [19] to cluster a set of documents into a number of smaller subsets and process each subset of documents individually to reduce the complexity of SVD in computation. One representative work of this approach is presented in [14] in which three clustering schemes are introduced, i.e. non-clustered retrieval (NC), full clustered retrieval (FC) and partial clustered retrieval (PC). The NC scheme employs a truncated SVD to pre-process the original data without any clustering. The FC scheme fully clusters data with a *k-means* algorithm, and then makes use of SVD to

approximate the matrix of the document vectors in each cluster. The PC scheme only works on a few clusters that are closely related to a given query for high efficiency.

Another approach distributes the computation of LSI among a cluster of computers using the Message Passing Interface (MPI). For example, Seshadri and Iyer [28] proposed a parallel SVD clustering algorithm using MPI. Documents are split into a number of subsets of which each subset of the documents is clustered by a participating computer. Experimental results have shown that the overhead in LSI computation is significantly reduced using a number of processors.

Although the two aforementioned approaches are effective in a certain way in speeding up LSI computation, a number of challenges still remain. For example, the *k-means* approach does not consider the overhead incurred in clustering documents which can be high when the size of document collection is large. The MPI approach is restricted to homogeneous computing environments without any support for fault tolerance. It should be noted that modern computing infrastructures are mainly heterogeneous computing environments in which computing nodes have a variety of resources in terms of processor speed, hard disk and network bandwidth. As a result, distributing LSI computation in a heterogeneous computing environment with MPI can cause severe unbalanced workload in computation which leads to poor performance.

### 2.5.3 Dynamic Load Balancing in Heterogeneous Environments

To solve the unbalanced load issue in a dynamic distributed computing environment, a number of studies have been done in enabling load balancing among a number of computing nodes. In the Hadoop framework currently it has no proper scheduler designed for a dynamically heterogeneous environment. Therefore the performance of the framework running on a heterogeneous cluster has chance to be enhanced. At present there are a few researches focusing on studying load balancing for MapReduce. One research contributed by Groot [57] pointed out that due to the

overheads of data copying, network transferring, local hard disk reading and writing, a *mapper* may limit the job execution time. The author also claimed that the slower *mapper* may hold the whole job processing up, which delays the job finishing time. To show the impacts of unbalanced load issue, the author use Jumbo based on Google Distributed File System [58], which is claimed that has the similar performance as Hadoop framework has. In the author's scenario he implemented two algorithms of which one is a single algorithm called 'word count' [2] and the other one is a complex one called 'Parallel FP-Growth frequent item set mining algorithm' [87]. In the evaluations the results show that the slower nodes delay the processing which causes that the faster nodes are not fully utilized. Based on the results, the author claimed that both *mapper* and *reducer* impact the performance of Hadoop framework. However, firstly in this paper only a number of experiments have been done without any solution on solving the unbalanced load issue. Secondly, the impacts brought by *reducer* should be considered. For theoretical algorithm experiments, multiple *reducers* may be involved in terms of efficiency. Contrarily, for a practical algorithm, *reducer* is normally involved to collect the final output which should be regarded as a whole data set without any segmentation. Thus for the data integrity, single *reducer* is better than multiple *reducers* which needs another job to collect parts from different *reducers* to form a whole data set. Therefore it is regarded that in the data processing, the load issues among multiple *mappers* are more critical.

One group of researchers realized the importance of load balancing issue in Hadoop as well. Sadasivam et al. [56] try to optimize the performance of the Hadoop cluster so that they proposed an approach called Parallel Hybrid PSO-GA using MapReduce based on genetic algorithm. In their algorithm they use Hadoop framework itself to deal with the genetic algorithm [81] which aims to solve the unbalanced load issue in Hadoop. Their algorithm mainly aims on achieving an optimized scheduler for multiple users based on the different resource capacities. During the processing, they made the number of iterations maximally 30 times to guarantee the efficiency. Their results show that the PSO-GA algorithm outperforms Max MIPS, typical PSO and

typical GA. However, several points can be criticized from their design and implementation. The first point is that the overhead of Hadoop is quite considerable. When the framework is involved to compute a Hadoop job scheduler for Hadoop itself, though the overhead of following jobs may be reduced, the overhead of the scheduler computation definitely cannot be avoided. The second point in their design is they just simply consider the capacity of a resource in terms of utilization of processor. This simply idea is lack of accuracy to describe the real Hadoop system. As studied in paper [77], there are a number of factors which may impact the performances of the framework including processing features, IO features and Hadoop working mechanisms. Therefore the fitness function based on pure utilizations of processors in Parallel Hybrid PSO-GA cannot compute the scheduler accurately. The third point is just 30 times iterations involved in their algorithm cannot get the optimized solution. The existing errors may differentiate the performance of the scheduler from the actual optimized scheduler. However to increase the number of iterations will increase the overhead of the algorithm, the authors have not done any compensation to calibrate these two issues. The forth point is they considered to balance the load among multiple users but they do not consider the load among *mappers* for one job. Thus the unbalanced load will make certain number of *mapper*s unutilized, which delays one job. Moreover the total number of jobs will be affected.

Another group of researchers aim to assign different volumes of so-called data fragments to different computing nodes on balancing the loads. Xie et al. [55] established a heterogeneous Hadoop cluster and measured the processing speed of each node based on the overhead of processing 1GB data. And then according to different computing ratio, they allocated the nodes with different number of fragments proportionally. They claimed that their strategy enhances the performance of Hadoop framework. However, there are three arguments about their research. The first one is that is it proper to define the computing ratio for one node based on simply testing the overhead of processing 1GB data? It is well recognized that the processing steps of

Hadoop are quite complex. The number of processor, hard disk, memory buffer, network and parameters' operations are involved. For instances, two absolutely same nodes from hardware aspects will definitely give different performances on processing 1GB data with different buffer size configurations. And also, a machine with slower hard disk may outperform a machine with faster hard disk on processing 1GB data with different sort factor configurations. Therefore the simply way of defining the processing speed of a node cannot represent the real processing capacity of the node. The second argument is that in the current version of Hadoop framework, one job cannot decide how many data chunks to be sent to a node. The only way of Hadoop data chunk allocation is each *mapper* copies one data chunk from HDFS without any interference from users. They may find a particular way to test their strategy in the practical Hadoop cluster. However, they do not mention that. The third argument is in their paper they claims that their computing ratio is based on the response time of each node which is proportional to the processing speed of the processor. For some of the algorithms which have less data output, their computing ratio may perform well. However, when the algorithm has a number of IO operations, which cause the response time is not proportional to the processing speed to the processor, the performance of their algorithm will be definitely deteriorated.

The above studies aim to solve the unbalanced load issue in Hadoop framework based on MapReduce computing model. Though their approaches give ways to enhance the performances, they do not consider multiple factors involved in Hadoop. Therefore their results are not that representative. Moreover, they do not consider any dynamic issues of the algorithm while it is well known that the Hadoop computing environment is dynamic from aspects of loads of processors, speed of IO devices and states of the cluster. Though currently there is little research to study the load balancing strategy in a dynamic Hadoop distributed computing environment, a large number of dynamic load balancing strategies have been published for the other scenarios. These strategies can also give enlightens for designing a dynamic load balancing algorithm especially for Hadoop framework.

One famous way to design a dynamic load balancing algorithm is to implement a static load balancing algorithm repeatedly in a number of time intervals for a dynamically changed environment [61]. Maeng et al. [63] proposed an algorithm named as 'Dynamic Load Balancing of Iterative Data Parallel Problems on a Workstation Clustering' following the above way. The experimental results show that it is a proper way of implementing static load balancing algorithm in a time interval to adapt to a dynamic environment. The approach can enhance the performance of the cluster. Zomaya et al. [64] follow the same way to design their load balancing algorithm. They involve genetic algorithm [82] [83] [85] in each time interval to achieve the optimized job scheduler according to the speeds of processor employed in the cluster. To facilitate the design, they firstly use a fixed 'window size' [60] [63] representing the time interval. Secondly they restrict the iterations of the genetic algorithm to be 10 times. From their experimental results, the performance of cluster is enhanced greatly using their scenario. However, they do not test the impacts of different window sizes. It should be pointed out that the various window sizes may vary the performances according to the changes incurred in the environment. Also they stiffly set the generation of the genetic algorithm to be 10. As the same as we argued in [56], it is quite doubtful that if the solution with only 10-times evolution suffices the optimized solution. However, their studies still can be referenced in the design of dynamic load balancing algorithm for Hadoop framework. Another point should be considered is where the load balancing algorithm can be computed. H. C. Lin and et al. [60] proposed a way to balance load for dynamic environment using a centralized job dispatcher [84]. In their algorithm a number of nodes can be handled by the job dispatcher which uses global state information in making decisions. Based on their evaluations they claim that the policy is most suitable for systems with high-speed communications. And also Bonomi et al. [80] proposed an adaptive optimal load-balancing algorithm in a heterogeneous multiple server system with a central job scheduler. The central job scheduler decides the load in a similar way which [60] [84] do. Their study is quite helpful in designing the dynamic load

balancing algorithm for Hadoop as the framework has two features which are quite similar to their scenario. The first is there is a component called JobTracker [33] [35] which holds the global information of the cluster. The second is though communications exist in the framework, the communication overhead is less than the other distributed computing systems like MPI. Thus based on their idea, a centralized job dispatcher is quite suitable.

In order to facilitate the experiments, to design a dynamic computing environment is necessary. As stated by Dhakal et al. [65], although somewhat restrictive, this is a meaningful assumption in order to obtain an analytically tractable result. So they designed the load of processor following exponential distribution. Based on that dynamic load, they develop their own strategy to balance the load among nodes. Their research gives a way to establish an environment with dynamic factors in which the performances of the load balancing algorithms can be evaluated.

Summarizing, researches on load balancing for Hadoop framework are mainly focus on enhancing the performance without considering the lower layer and dynamic features in detail. In their designs, they only simply consider the processing speed of the nodes. However, it is known that the processing steps in Hadoop involve a number of interactions among hardware and cluster parameters. These detailed mechanisms affects the performances of a Hadoop cluster quite a lot. And also they do not consider how to balance loads among nodes in a dynamic environment, which the situation practically exists in a real varying resources Hadoop cluster caused by operating system loads or the resource sharing. Another point should be mentioned is that due to the differences of processing mechanisms between the Hadoop framework and the other distributing computing systems, the dynamic load balancing algorithms designed for the other systems are not suitable for Hadoop. Therefore these challenges motivate the design of our dynamic load balancing algorithm which targets on balancing load among nodes in a dynamic heterogeneous Hadoop computing environment.

## 2.6 Summary

Due to the scalability issue of the LSI algorithm, several works to speed up the performance in term of overhead have been done. To combine the clustering algorithm such as *k-means* is regarded as a remarkable way. However, the newly involved clustering algorithm brings new overhead so that the new issue should be solved. And also in most of the works, the load balancing issue has never been considered. Thus, a solution for balancing loads among nodes in the distributed computing environment with both static and dynamic factors should be considered. Therefore, to evaluate the performance of MR-LSI in a large Hadoop environment and the performance of load balancing algorithm, a Hadoop simulator which can accurately simulate the Hadoop framework is needed.

# Chapter 3

# HSim: A MapReduce Simulator

MapReduce is an enabling technology in support of Cloud Computing. Hadoop which is a MapReduce implementation has been widely used in developing MapReduce applications. This chapter presents HSim, a MapReduce simulator which builds on top of Hadoop. HSim models a large number of parameters that can affect the behaviors of MapReduce nodes, and thus it can be used to tune the performance of a MapReduce cluster. HSim is validated with both benchmark results and user customized MapReduce applications.

## 3.1 Modeling Hadoop Parameters

The performance of a Hadoop application can be affected by a large number of parameters. In this section, we present the modeling work on these parameters.

### 3.1.1 Node Parameters

- Processor: HSim supports one processor per computer by default design, but the number of processors could be changed. One processor can have one or more cores. The processing speed of a processor core is defined as the volume of data units processed per seconds which can be measured from real experimental tests.
- Hard disk: In hard disk entity, the speeds of IO operations vary from time to time. Several parameters are introduced to build the degressive reading/writing model. Let $x_{max}$ represent the maximum reading/writing speed of hard disk. For example from the experimental results of testing Seagate Barracuda 1 TB hard disk $x_{max}$ is about 120MB/s in reading, and 60MB/s in writing. Let $x_{min}$ represent the minimum reading/writing speed of hard disk, $x_{min}$ is around 55MB/s in reading and 25MB/s in writing. Another parameter which is degressive factor *r* is used to represent in each second the value of lost speed. The value of the factor is around 0.0056 based on experimental tests. Using these parameters we can calculate the real time speed $x$ of hard disk using formula

(3.1).

$$H(t) = \frac{x_{min} \cdot x_{max}}{(x_{min} - x_{max})e^{-rt} + x_{max}} \quad (3.1)$$

- Memory: In each memory entity two parameters are modeled, reading and writing. In our experimental tests, the reading speed of standard DDR2-800 memory with dual-channel could reach up to 6000MB/s and the writing speed is up to 5000 MB/s. It is quite obvious that both the reading and writing speeds would not be the bottlenecks of the system due to their fast speeds.
- Ethernet adaptor: In each Ethernet adaptor entity, two parameters are modeled, upstream bandwidth and downstream bandwidth. The bandwidth can be in the range of 100Mbps and 1000Mbps.

### 3.1.2 Cluster Parameters

The cluster parameters represent the details of a simulated Hadoop cluster. It involves several aspects which include the number of nodes, topology and network facilities.

- Number of nodes: The number of nodes can vary from 1 to a few hundreds.
- Topology: The number of nodes can be organized with a certain network topology. Currently HSim only supports simple racks.
- Network facilities: The speed of a router can be in the range of 100Mbps and 1000Mbps. When the bandwidth of a router is defined, a number of standalone computers must be configured to connect to the router to decide on their network capacities.
- Job queue and job schedulers: A job queue holds the waiting job entities. According to different job schedulers, jobs are waiting for processing resources. HSim supports two job schedulers of Hadoop framework – first come first serve and fair scheduler. These two types of schedulers generate different job processing orders.

### 3.1.3 Hadoop System Parameters

Before a Hadoop application starts processing data, the data should be saved into

Hadoop Distributed File System (HDFS) [33] [35] in advance. The number of files affects the number of Map instances involved. Normally the number of Map instances equals to the number of file chunks. If the number of chunks is larger than the maximum number of Map instances in the cluster, Map instances will be assigned with data chunks via waves. If a whole data set is only saved in one file, the single file could be separated into a number of chunks logically via supplied APIs of the Hadoop framework. When data is being processed, it would go through a number of processing steps such as sorting, merging, combining, copying, reducing. These steps highly affect the performances of the system so that several parameters are modeled to control the behaviors of these steps. As these parameters are configurable and most of them are involved in the actual Hadoop framework so we named these parameters Hadoop system parameters.

- **Job specifications**: In a job specification, a number of parameters are involved to describe the properties of a job. Job ID refers the unique id assigned to each job for tracking. The JobSize is the total size of the input data. No matter how many chunks of the data are submitted, this value should be the total size of the whole data. When the simulation starts, the data will be fetched from the HDFS. The NumberOfRecords parameter is used to represent the number of records in the data so that the size of each record can be calculated by this value and the size of the job. In the simulator this parameter is experimentally used to measure the number of records combined by Combiners, which may affect the performances of the system when certain types of Hadoop applications are executed. The MapOutputRatio parameter represents the volume of intermediate data to be generated by Map instances which has an impact on IO performance. The ReduceOutputRatio parameter is quite similar to MapOutputRatio. In some Hadoop applications the Reduce instances do not only copy data from Map instances but also generate their own intermediate data which affects IO performance. This parameter specifies the size of intermediate data to be generated in the Reduce phase. The ReducingRatio parameter represents the size of final results which will be reduced in HDFS. This parameter can affect the

performance of the underlying network and also the performance of a local hard disk. The NumberOfChunks parameter is used to specify the number of files to be used to carry data. This parameter determines the number of Map instances assigned to the job. If the number of chunks is only one, a number of logically separated files should be specified. The NumberOfReducers parameter represents the number of required Reduce instances for the job. If this parameter is defined, then a number of Reduce instances will be allocated for the current job according to their availabilities.

- **Simulated Hadoop parameters**: This group of parameters is highly related to Hadoop framework. The io.sort.mb parameter represents the size of memory buffer to use while sorting map output. The io.sort.record.percent parameter represents the proportion of io.sort.mb reserved for storing record boundaries of the map output results. The remaining space is used for the map output records themselves. The io.sort.spill.percent parameter is a threshold that determines when the Map instance should start spilling processes writing data into memory. If the threshold is reached, the CPU processing will be suspended and the buffer will be flushed, which means all the data saved in virtual memory will be spilled into hard disk. The io.sort.factor (1) parameter specifies the maximum number of streams to merge when sorting files in the Map phase. It significantly affects the IO performance of the system. The mapred.reduce.parallel.copies parameter refers to the number of threads used to copy map outputs to the reducer. Using a proper number of copying threads according to hardware resources, the performances of the system would be enhanced. The io.sort.factor (2) parameter represents the maximum number of streams to merge when sorting files is carrying out in the reduce phase. The mapred.job.shuffle.input.buffer.percent parameter is the proportion of total heap size to be allocated to the map outputs buffer during the copy phase of the shuffle. The mapred.inmem.merge.threshold parameter represents the threshold number of map outputs for starting the process of merging the outputs and spilling to hard disk. Using this parameter a number of smaller mapper outputs could be operated in memory but not local hard disk.

Therefore the sorting and merging involve less overhead generated by hard disk. The JVM Reuse parameter is partially simulated in HSim. Using JVM reuse, the overhead generated by some short-lived tasks will be significantly reduced.

### 3.1.4 HSim Parameters

HSim itself needs several parameters to control its own behaviors. Five important parameters are introduced in HSim:

- System Clock: The System Clock parameter is an absolutely and continuously timing component. In each change of the system clock, its current value will be added by one second. It is used to record the current system time, and to measure the performances of Hadoop applications in different cluster configurations.
- Executing Speed: This parameter controls the execution speeds of all the components in HSim.
- Accuracy Level: For normal Hadoop applications, it is enough to set this parameter to the level of seconds. To maintain high accuracy in simulation, milliseconds can be set for the applications as well.
- Shared Parameters: These parameters can control the rates of the shared resources include hard disk and bandwidth. The ratio is defined by

$$r = AssignedResource / TotalResource.$$

- Reporter: This parameter records several important system states for analysis.

Table 3.1: Summarizes the parameters modeled in HSim.

| Category | Specification |
|---|---|
| Node Parameters | processor, hard disk, memory, Ethernet card, Map instance, Reduce instance |
| Cluster Parameters | number of nodes, topology, network facilities, job queue, job scheduler |
| Hadoop System Parameters | job specifications, Hadoop parameters |
| HSim Parameters | system clock, execution speed, accuracy level, shared parameters, reporter |

## 3.2 The Design of HSim

This section presents the design of HSim in detail. The prototype is based on Hadoop framework.

### 3.2.1 HSim Architecture

Figure 3.1 shows the data flow of HSim. To perform a simulation, the Cluster Reader component reads the cluster parameters from the Cluster Spec to create a simulated Hadoop cluster environment. A specified number of nodes are initialized and arranged using a certain type of topology. After the cluster is configured, the node parameters will be processed by the Cluster Reader as well to specify the types of nodes including processors, hard disk, memory, Master node, Slave nodes, Map instances and Reduce instances. This initialization process can create both homogeneous nodes and heterogeneous nodes. Then the simulated cluster is ready for incoming jobs retrieved from the job queue using different job schedulers. The Job Spec will be processed by the Job Reader component and jobs will be submitted to HSim for simulation.

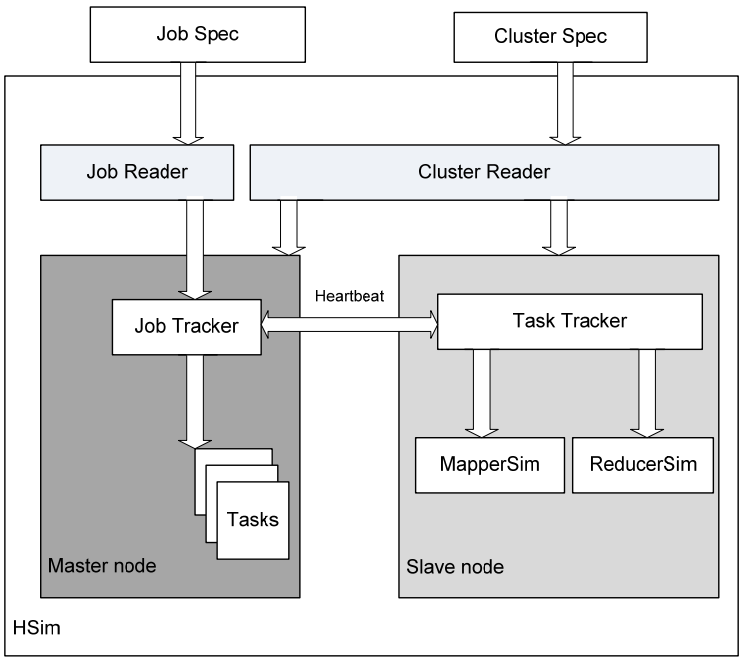


Figure 3.1: HSim components.

HSim follows a master-slave mode. The simulated Map instances (MapperSim), Reduce instances (ReducerSim), JobTracker and TaskTrackers are located on these

nodes. The Master node is the Namenode of Hadoop framework which contains JobTracker to correspond and schedule the tasks. The Slave nodes are the Datanodes of Hadoop framework which contains TaskTrackers. On Slave nodes Map instances and Reduce instances perform data processing tasks. From Figure 3.1 it can be observed that when a job is submitted to a simulated Hadoop cluster, the JobTracker splits the job into several tasks. Then TaskTracker and JobTracker will communicate with each other via messaging based on heartbeats. One thing should be pointed out that in Hadoop framework, the communications among JobTracker and TaskTrackers are based on HTTP. However in the simulator simplicity has been done. The HTTP communications are not simulated but using the times consumed by the communications to measure the overhead generated by HTTP communications. If the JobTracker finds that all the Map tasks have been finished, and then the Reduce instances will be notified to be ready for merging phase. Moreover if the JobTracker finds all Reduce tasks have been finished, then the job will be considered as finished. If the Map tasks have not been finished yet, the TaskTrackers will be notified to choose a Map task or a Reduce Task based on their availabilities.

### 3.2.2 MapperSim

When a Hadoop application is submitted to HSim, the input data will be split into a number of data chunks and each chunk is associated with a Map instance. During the processing, each task will be assigned to a Map instance for execution. The operations of a Map instance are simulated by the MapperSim component. MapperSim simulates the operations of a Map instance (mapper) on each node. It copies data which is saved in HDFS to its own local hard disk. Commonly each MapperSim processes one file chunk but if only one file chunk is saved in HDFS, then a logically separated number of chunks can control the number of MapperSim instances involved in the job. When the data is copied and saved in the local hard disk, MapperSim starts processing the data based on the job spec of the simulated Hadoop application. During the processing steps, intermediate data will be generated. To improve the IO performance, the intermediate data will be written into a memory buffer. In the buffer, the data can be

pre-sorted to gain high efficiency. As long as data is writing into the buffer, if a threshold is reached, a background thread will start spilling the data to hard disk. The intermediate data will be kept writing into the buffer while the spilling takes place. If the buffer is full during this time, the CPU processing will be blocked until the spill procedure is complete. This step means that the processor involved in MapperSim does not simply keep processing, it may be interrupted by the current states of memory buffer. For each spilled chunk of the output, before it is written to the hard disk the background thread will divide the chunk into partitions which are associated with the Reduce instances. During this step, the in-memory pre-sorting is occurred. And if a Combine function is needed, combiner will be involved in this step after sorting. After the task is finished, the partitions will be merged into a single file which contains sorted data to be copied to the Reduce instances. Figure 3.2 shows the working mechanism of MapperSim.

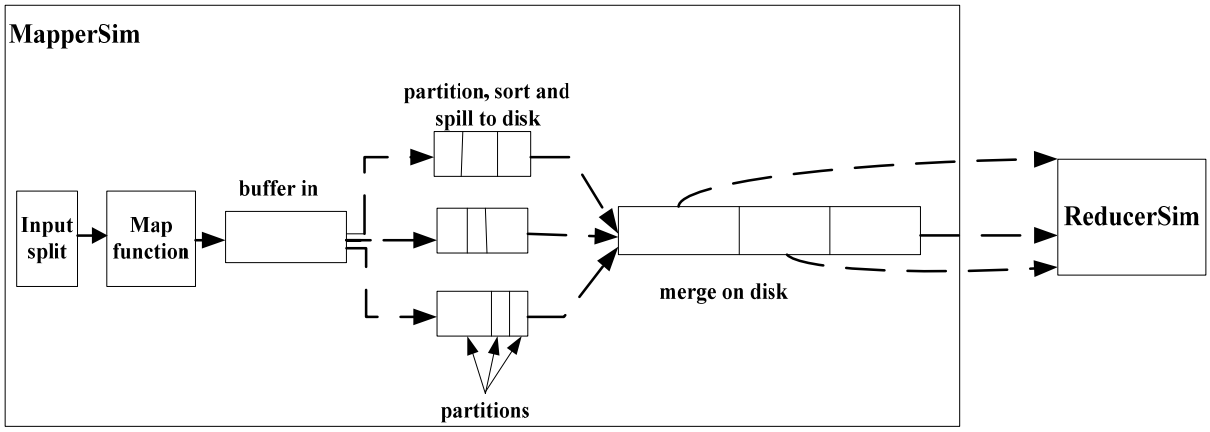


Figure 3.2: Data flows in the MapperSim component.

Figure 3.3 shows a sequential diagram shows the interactions of MapperSim with other components in HSim.

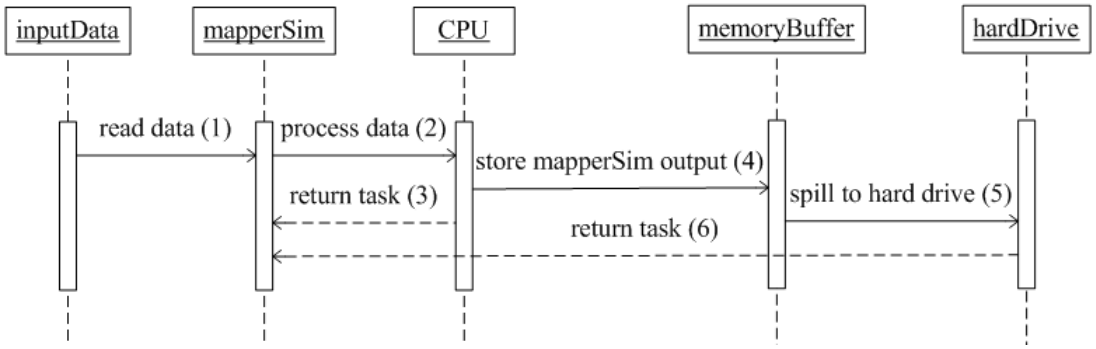


Figure 3.3: MapperSim sequence diagram.

### 3.2.3 ReducerSim

The ReducerSim component simulates the Reduce instances in Hadoop framework. It is used to collect the outputs from MapperSim and reduce the final outputs into HDFS. Figure 3.4 shows the data flows in ReducerSim.

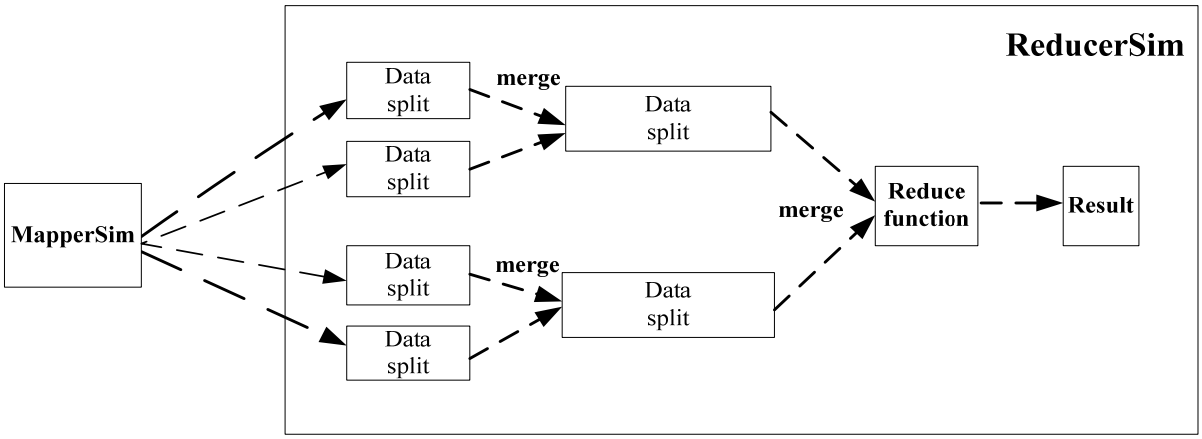


Figure 3.4: Data flows in the ReducerSim component.

The output files of the MapperSim component are saved in the local hard disk. The ReducerSim component needs the output from several MapperSim components for its particular partition. The ReducerSim starts copying data when an output is ready. Each ReducerSim has a number of copying threads so that it can copy the output results from a number of MapperSim components in parallel. If the size of the output is small, it will be copied into a memory buffer otherwise it will be copied into the hard disk directly. If the output results are copied into memory, when a certain threshold is reached, e.g. a percentage of buffer used or a number of file copied, these

outputs will be merged and spilled into hard disk. As the number of files increases, a background thread merges them into larger and sorted files. When all the output results from the MapperSim components have been copied, the sorting step will start. This step merges the map outputs and maintains sorting orders of outputs. After the files have been sorted, they will be reduced into HDFS as one final output. For some Hadoop applications, the Reduce instances may need to process data involving processors but without IO operations. The ReducerSim in HSim supports this feature. Figure 3.5 shows its sequence diagram.

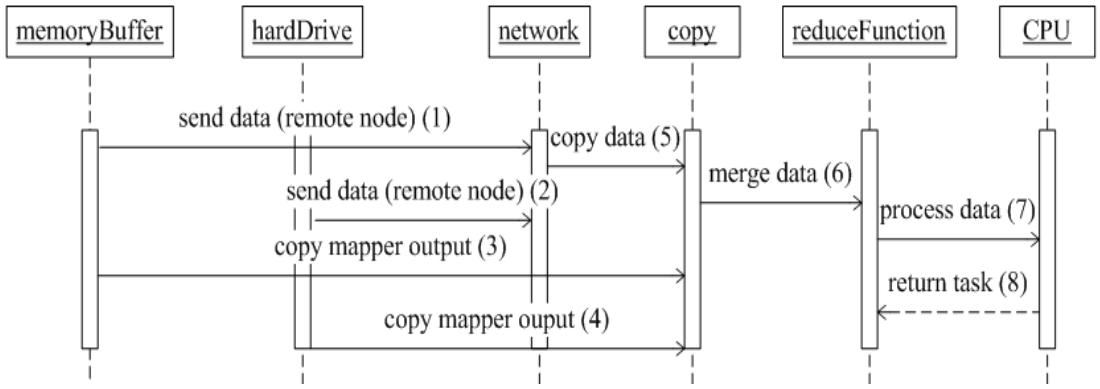


Figure 3.5: Hardware interactions in ReducerSim.

### 3.2.4 JobTracker and TaskTracker

JobTracker is mainly used to track a simulated job and TaskTracker is used to run individual tasks. When a job is submitted, the job ID will be sent to JobTracker for tracking. The JobTracker starts computing the input splits for the job. Then it creates one map task for each split. TaskTrackers periodically send messages to the JobTracker via heartbeats which tell the JobTracker that a TaskTracker is working. As part of the heartbeat, a TaskTracker will tell that if the current task is finished and ready to run a new task. Figure 3.6 shows the work flows of the components in HSim

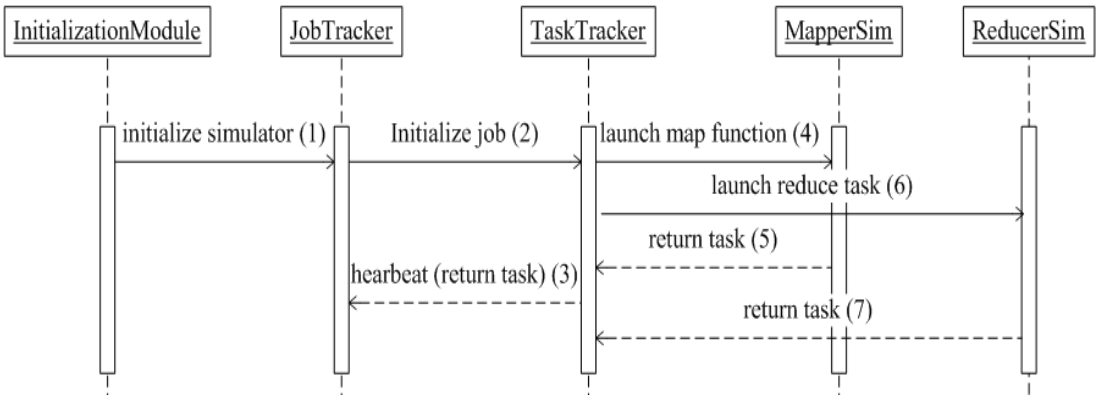


Figure 3.6: The workflow of HSim.

## 3.3 Validations of HSim

To validate HSim, a number of tests have been conducted. The performances of HSim against published benchmark results have been compared. And also an experimental environment of a Hadoop cluster has been set up and the simulator HSim is evaluated with our Hadoop applications.

### 3.3.1 Validating HSim with Benchmarks

HSim is validated firstly with 3 benchmark results presented in [27] [54] - Grep Task, Selection Task and UDF Aggregation Task.

*3.3.1.1 Grep Task*

This task simulated exactly what [27] [54] did in their benchmarking work. HSim simulated the cluster using 1 node, 10 nodes, 25 nodes, 50 nodes and 100 nodes respectively. Two different scenarios have been tested, one is that each node is assigned 535MB data to process, and the other is that 1TB data is submitted to the cluster. Each scenario was evaluated 5 times. The simulation results are plotted in Figure 3.7 and Figure 3.8 respectively which are close to the benchmark results. Both the simulation results and benchmark results are in the same scale. Regarding the complex physical environments, the simulation results can supply acceptable accuracy. The gaps between simulation results and benchmark results can be ignored. The confidence intervals of the results are small in both scenarios (in the range of 0 and 2.6 seconds in the first scenario and in the range of 4.1 and 7.6seconds in the second scenario) showing a stable performance of HSim.

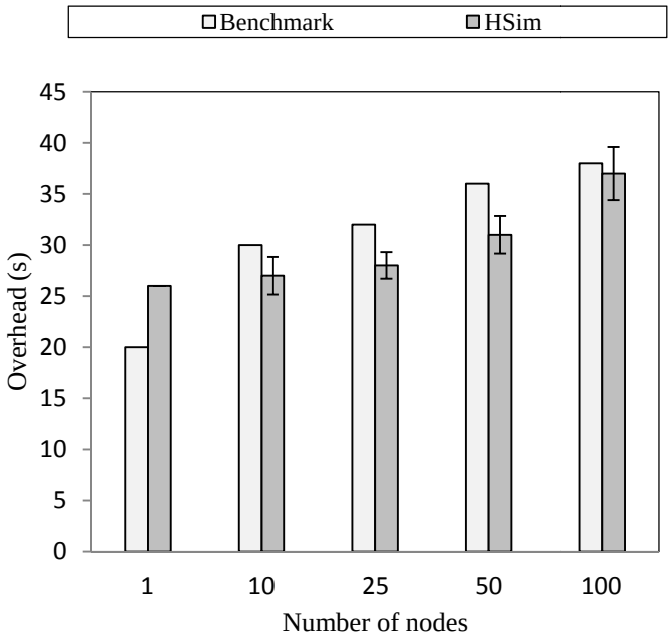


Figure 3.7: Grep Task evaluation (535MB/node).

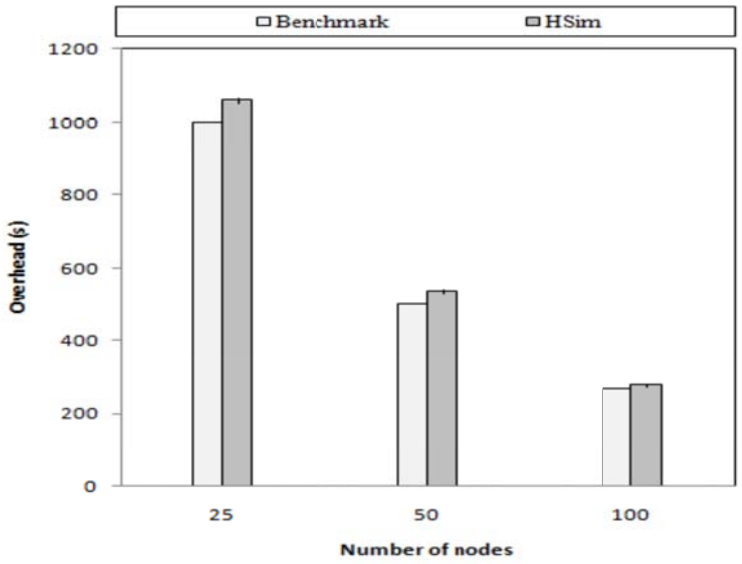


Figure 3.8: Grep Task evaluation (1TB/cluster).

### *3.3.1.2 Selection Task*

The Selection Task was designed to observe the performances of Hadoop framework dealing with complex tasks. Each node processes 1GB ranking table to retrieve the target pageURLs with a user defined threshold. This task is simulated and the results are shown in Figure 3.9. Regarding the multiple factors and complexity of the physical benchmark environments, the simulation results are highly close to the

benchmark result. The simulation results show that considering the complex working mechanisms and parameters of Hadoop framework, the simulator HSim can supply sufficiently close results compared to the benchmark results.

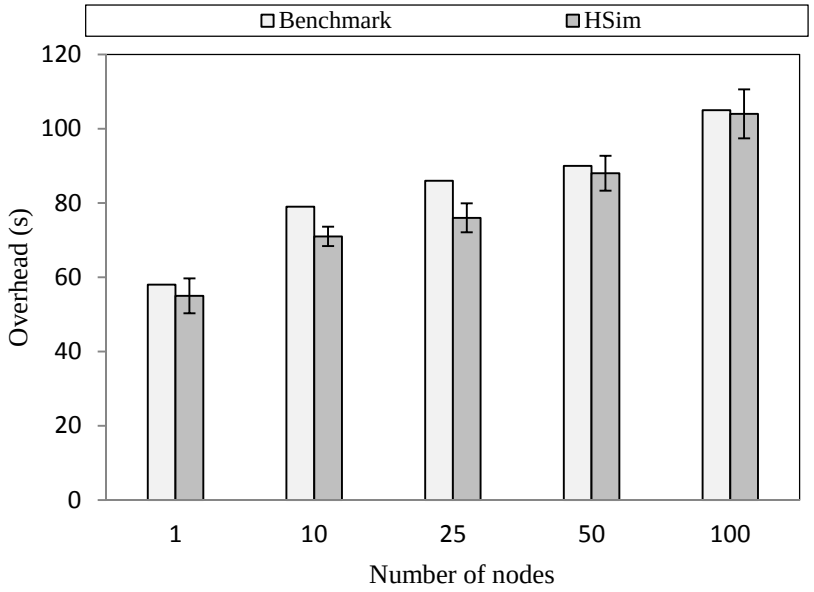


Figure 3.9: Selection task evaluation.

From Figure 3.9 it can be clearly observed that the simulated results are close to the benchmark results, and the confidence intervals are small, in the range of 2.6 and 6.6 seconds.

*3.3.1.3 UDF Aggregation Task*

The UDF Aggregation Task reads the generated document files and searches for all the URLs appeared in the contents. And then for each unique URL, HSim counts the number of unique pages that refers to that particular URL across the entire set of files. The simulation results are shown in Figure 3.10 which again are close to the benchmark results considering the complexities of the simulations. The simulation results show a high stability of HSim for the task.

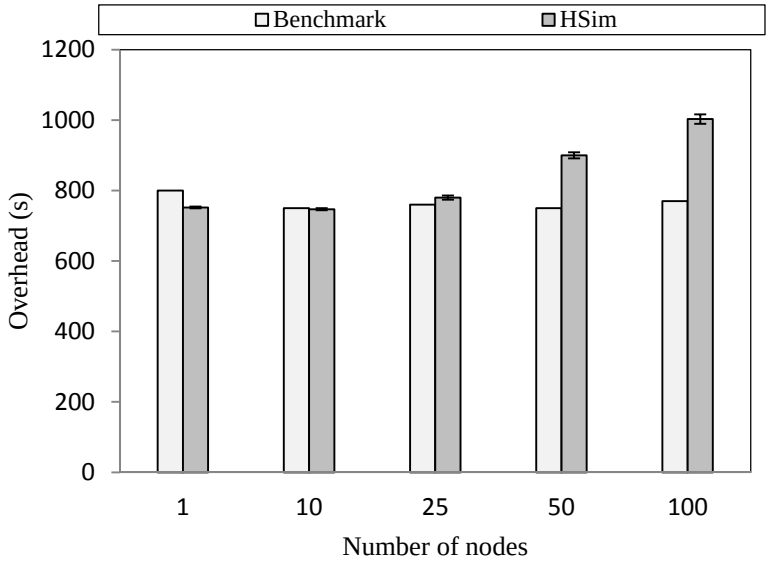


Figure 3.10: Aggregation task evaluation.

### 3.3.2 Evaluating HSim with Customized Hadoop Applications

Two customized Hadoop applications are involved for validation secondly - one is for information retrieval and the other one is for content based image annotation. The two applications were evaluated in both a Hadoop experimental cluster and HSim. This section presents the evaluation results.

*3.3.2.1 The Experimental and Simulated Environments*

The Hadoop experimental cluster consists of 4 nodes. Three nodes were used as Datanodes with CPU Q6600@2.4G, RAM 3GB, 120GB Seagate Hard Disk, and running OS Fedora 12. One node is used Namenode with CPU C2D7750@2.26G, 2GB RAM and running OS Fedora 12. Each Datanode employed 4 mappers and 1 reducer with default cluster configurations. The network bandwidth is 1Gbps. We used HSim to simulate a Hadoop cluster with the same configurations as those of the experimental cluster.

*3.3.2.2 MR-LSI*

MR-LSI [52] is a MapReduce based distributed LSI algorithm for information retrieval. The details will be described in the next chapter. MR-LSI is designed and implemented using the Hadoop framework. It involves both Map and Reduce

functions, and contains a number of IO operations. MR-LSI was evaluated in both an experimental environment and HSim, and plotted the results in Figure 3.11.

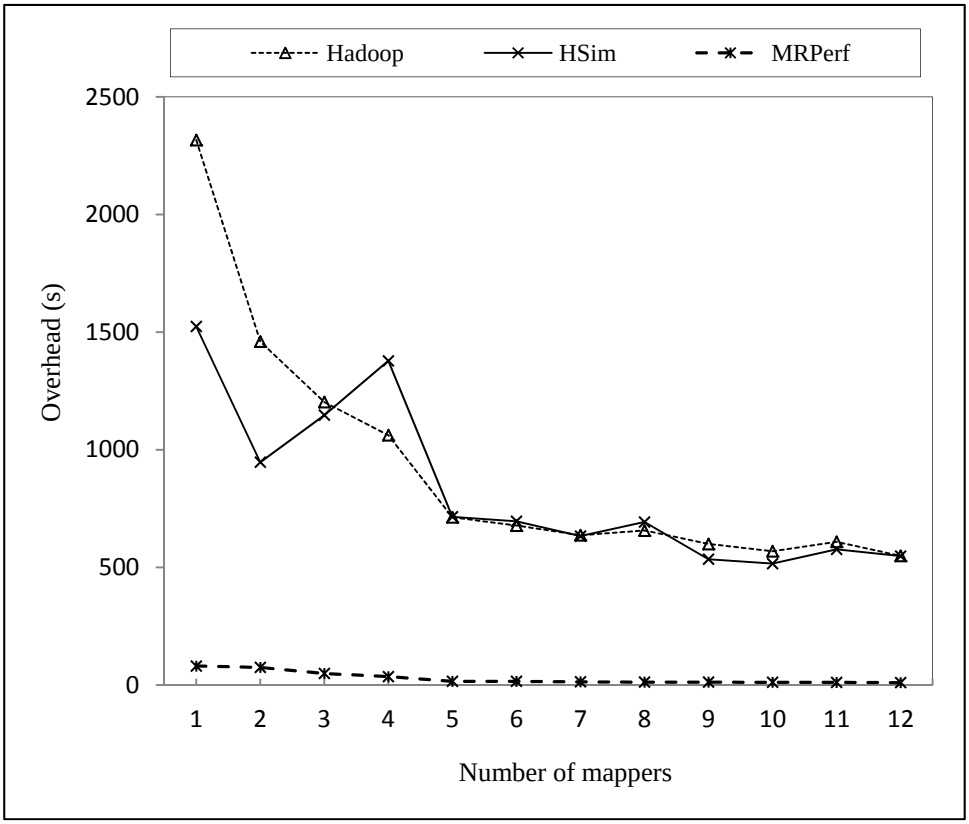


Figure 3.11: Evaluating HSim with MR-LSI.

It can be observed that the overall performance of HSim is substantially close to that of the real Hadoop cluster, especially for scenarios dealing with MapReduce jobs with larger sizes of datasets and involving an increased number of mappers. One thing should be pointed out that HSim is designed to simulate a large scale Hadoop cluster so that if only one node is in the cluster the errors may occur due to inaccuracy of simulating a cluster consisted by a single node (In this case one machine employs four mappers so when the number of mappers is less than 5, there is only one node in the cluster.). For comparison purpose, the performance of the MRPerf simulator is also tested using the same configurations as that of HSim. From the results presented in Figure 12 it can be seen that HSim significantly outperforms MRPerf in comparison with the performance of the real Hadoop cluster. As discussed in Section 2, using too much estimation on the values of Hadoop parameters limits MRPerf in simulating

MapReduce behaviors accurately.

*3.3.2.3 MR-SMO*

MR-SMO [53] is a MapReduce based distributed SMO algorithm for content based image annotation. MR-SMO is built on Hadoop framework, and also involves both Map and Reduce functions. MR-SMO was evaluated in the experimental Hadoop cluster as well as in HSim. The MRPerf simulator is also employed to evaluate the performance of MR-SMO. From the results presented in Figure 3.12 it can be observed that the performance of the simulated cluster using HSim is considerably close to that of the real Hadoop cluster. Again MRPerf does not produce accurate simulation results.

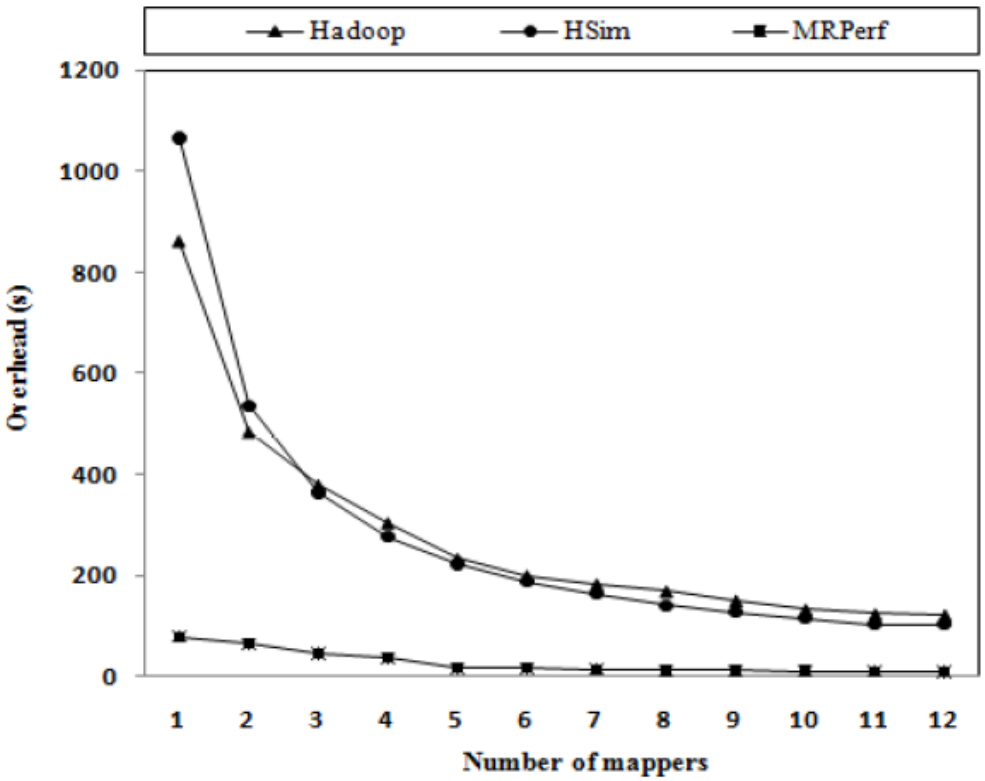


Figure 3.12: Evaluating HSim with MR-SMO.

### 3.3.3 Discussions

The Hadoop framework is a complex system involving a number of components. HSim is designed and implemented to simulate such components and interactions. It

works similarly like the way of the Hadoop framework works. However we cannot simply conclude that HSim can accurately simulate Hadoop without any limitations. The accuracy of HSim can be affected by a number of factors such as the time of job propagations, cold starts of Map instances, key distributions, system communications, shared hardware resources and dynamic IO loads. These dynamic factors may affect the performance of both experimental and simulated results depending on user applications. Enabling the Combiner feature of Hadoop also can affect the accuracy of HSim. However, the combiner instance has not been fully implemented in HSim. A combiner can be considered as an in-memory sort process. The output of mappers will be combined and written into an intermediate file by a combiner. And then the file will be sent to a reducer. So when the number of mappers is small, the benefits gained from using combiners are not significant. However when the number of mappers gets large, system IO operations includes hard disk reading, writing and network utilities will benefit a lot from combiners. Though HSim does not work that well with simulated combiners in large clusters, it still performs well in a simulated cluster with up to 100 nodes.

## 3.4 Summary

This chapter presents HSim, a Hadoop simulator for simulating data intensive MapReduce applications. HSim was validated with established benchmark results and also with experimental environments which have shown that HSim can accurately simulator MapReduce behaviors. HSim can be used to investigate the impacts of the large number of Hadoop parameters by tuning their values. It can also be used to study the scalability of MapReduce applications which might involve hundreds of nodes.

# Chapter 4

## Parallelizing LSI for Scalable Information Retrieval

Latent Semantic Indexing (LSI) has been widely used in information retrieval due to its efficiency in solving the problems of polysemy and synonymy. However, three drawbacks affect the performance of LSI. The first disadvantage is that LSI is notably a computationally intensive process because of the computing complexities of singular value decomposition and filtering operations involved in the process. The second disadvantage is several studies show that the truncated SVD can be lack of efficiency in processing large inhomogeneous text collections [14] [17]. The third disadvantage is for large datasets the SVD computation may be too expensive to be carried out on conventional computers. Also, the dense data structure of the truncated SVD matrices poses a huge challenge for both disk and memory spaces of conventional computers [13] [14]. Thus, a number of researchers proposed algorithms based on clustering technologies [3] [13] [14] [92] [93] to solve the issues stated of LSI. One of the clustering algorithm *k-means* has been involved by [3] [13] [14]. Combining with *k-means*, the original dataset of documents can be clustered into several sub-clusters according to the similarities of topics of the documents. As a result, the dimension of the original T-D matrix formed from the inhomogeneous text collections is reduced. Also, the computing complexity and cost are reduced. However, it should be noted that the combined clustering algorithm *k-means* can also generate large overhead when it is dealing with large dataset. Thus to distribute the k-means combining with LSI is an efficient way to solve the above issue.

This chapter presents a MapReduce based distributed LSI algorithm (MR-LSI) for high performance and scalable information retrieval. MR-LSI distributes *k-means* using Hadoop framework based on MapReduce computing model. Each *mapper* processes a data chunk which is separated from the original dataset by running *k-means* algorithm. After the dataset is clustered, a number of sub-clusters are output

by *reducer*. And then, a number of *mappers* are started to do truncated SVD computation in each sub-cluster. Finally, *reducer* outputs the final results into HDFS. The performance of MR-LSI is first evaluated in a small scale experimental environment. Subsequently, HSim is involved for further evaluation of MR-LSI in large scale simulation environments. By partitioning the dataset into smaller subsets and optimizing the partitioned subsets across a cluster of computing nodes, the overhead of the MR-LSI algorithm is reduced significantly while maintaining a high level of accuracy in retrieving documents of user interest. A genetic algorithm based load balancing scheme is also designed to optimize the performance of MR-LSI in heterogeneous computing environments in which the computing nodes have varied resources.

## 4.1 The Design and Implementation of MR-LSI

MR-LSI employs *k-means* to group documents into a number of clusters of documents. To minimize the overhead of *k-means* in clustering documents, MR-LSI partitions the set of documents into a number of subsets of documents and distributes these subsets of documents among a number of processors in a MapReduce Hadoop environment. Each processor only clusters a portion of the documents and subsequently performs a truncated SVD operation on the generated document cluster. The details on the design of MR-LSI are given below.

Let

- $D$ represent the set of $p$ documents, $D = \{d_1, d_2, d_3, \dots, d_p\}$.
- $P$ represent the set of $m$ processors in a Hadoop cluster, $P = \{p_1, p_2, p_3, \dots, p_m\}$. Each processor runs one *map* instance called *mapper*.
- $M$ represent the set of $m$ *mappers* running in the Hadoop cluster, $M = \{map_1, map_2, map_3, \dots, map_m\}$.

In LSI, the set of $D$ documents can be represented by a set of vectors denoted by $V$,

$V = \{v_1, v_2, v_3, \dots, v_p\}$. Each vector $v_i$ represents the frequencies of keywords that appear in document $d_i$. The input of each *mapper* includes two parts. The first part is a centroid set of $C$ with $k$ initial centroids which are randomly selected from the vector set $V$, $C = \{c_i \in V | c_1, c_2, c_3, \dots, c_k\}$. The second part of the input of a *mapper* is a portion of $V$ denoted by $V_i$. The vector set $V$ is equally divided into $m$ portions according to the number of *mappers*. Thus $V_i$ satisfies $\bigcup_{i=1}^{m} V_i = V$.

Each *mapper* $m_i$ runs on one processor $p_i$ calculating the Euclid distances between $v_{ij} \in V_i$ and $C$ which is denoted by $d_{ij}$, then

$$d_{ij} = \|v_{ij} - c_q\|,\ j = 1, 2, \dots, \frac{p}{m},\ q = 1, 2, \dots, k$$

Let $d_{min}$ represent the shortest distance between $v_{ij}$ and $C$, then $d_{min} = min(d_{i1}, d_{i2}, d_{i3}, \dots, d_{i\frac{p}{m}})$.

Based on the shortest distance, the *mapper* selects the corresponding $c_i$ and $v_{ij}$ to generate a key-value pair as one output record. The output pairs of all the *mappers* are fed into the *reduce* instance (called *reducer*). The *reducer* groups the values with the same key $c_i$ into a set of clusters denoted by $Cluster_i$, $Cluster_i = \{v_1', v_2', v_3', \dots, v_{ai}'\}$, where $i = 1, 2, 3, \dots, k$ and $\sum_{i=1}^{k} a_i = p$.

For each $Cluster_i$, the *reducer* calculates a new centroid denoted by $c_i'$, $c_i' = \frac{\sum_{j=1}^{a_i} v_j'}{a_i}$

The *reducer* outputs a set of centroids denoted by $C'$, $C' = \{c_1', c_2', c_3', \dots, c_k'\}$ which will be fed into the *mappers* for computing another set of centroids $C''$ until the values of the centroids in set $C'$ are the same as those in $C''$, then the *reducer* outputs the $Cluster_i$. Each of the $k$ jobs runs a *mapper* performing a truncated SVD operation in $Cluster_i$. In each $Cluster_i$, the vectors $v_{ai}'$ form a T-D matrix $A$, where $A = U\Sigma V^T$. After performing a truncated SVD operation, the matrix $A$ can be represented by an approximate matrix $A_k$, where $A_k = U_k \Sigma_k V_k$, $k$ is the rank of the matrix.

In LSI, for a submitted query $q$, it is processed using equation (4.1).

$$q_v = q^T U_k \Sigma_k^{-1} \quad (4.1)$$

The similarities of the query to the documents can be measured by calculating the cosine values of vector $q$ and the vectors of matrix $V_k$ using equation (4.2).

$$cos\,\theta_j = \frac{q_v \cdot D_j}{\|q_v\|_2 \|D_j\|_2} \quad (4.2)$$

where *j* represents the $j^{th}$ document in the clustered document set.

If the value of $cos\,\theta_j$ is larger than a given threshold $\tau$, then the document $D_j$ will be a target document. Therefore the set of target documents $D$ can be represented as $D = \{d_j | cos\,\theta_j = cos(q_v, D_j) \geq \tau\}$. Finally, the *reducer* generates $k$ clusters of documents. For each cluster of documents, a truncated SVD operation is performed and targeted documents are retrieved.

## 4.2 Static Load Balancing Strategy for MR-LSI

A remarkable characteristic of the MapReduce Hadoop framework is its support for heterogeneous computing environments. Therefore computing nodes with varied processing capabilities can be utilized to run MapReduce applications in parallel. However, current implementation of Hadoop only employs first-in-first-out (FIFO) and fair scheduling without support for load balancing taking into consideration the varied resources of computers. A genetic algorithm based load balancing scheme is designed to optimize the performance of MR-LSI in heterogeneous computing environments.

### 4.2.1 Algorithm Design

To solve an optimization problem, genetic algorithm solutions need to be represented as chromosomes encoded as a set of strings which are normally binary strings. However, a binary representation is not feasible as the number of *mappers* in a Hadoop cluster environment is normally large which will result in long binary strings.

A decimal string to represent a chromosome in which the data chunk assigned to a *mapper* is represented as a gene is employed.

In Hadoop, the total time ($T$) of a *mapper* in processing a data chunk consists of the following four parts:

- Data copying time ($t_c$) in copying a data chunk from Hadoop distributed file system to local hard disk. It depends on the available network bandwidth and the writing speed of hard disk.
- Processor running time ($t_p$) in processing a data chunk.
- Intermediate data merging time ($t_m$) in combining the output files of the *mapper* into one file for *reduce* operations.
- Buffer spilling time ($t_b$) in emptying filled buffers.

$$T = t_c + t_p + t_m + t_b \qquad (4.3)$$

Let

- $D_m$ be the size of the data chunk.
- $H_d$ be the writing speed of hard disk in MB/second.
- $B_w$ be the network bandwidth in MB/second.
- $P_r$ be the speed of the processor running the *mapper* process in MB/second.
- $B_f$ be the size of the buffer of the *mapper*.
- $R_a$ be the ratio of the size of the intermediate data to the size of the data chunk.
- $N_f$ be the number of frequencies in processing intermediate data.
- $N_b$ be the number of times that buffer is filled up.
- $V_b$ be the volume of data processed by the processor when the buffer is filled up.
- s be the sort factor of Hadoop.

Therefore

$$t_c = \frac{D_m}{\min(H_d, B_w)} \qquad (4.4)$$

Here $t_c$ depends on the available resources of hard disk and network bandwidth. The slower one of the two factors will be the bottleneck in copying data chunks from Hadoop distributed file system to the local hard disk of the *mapper*.

$$t_p = \frac{D_m}{P_r} \qquad (4.5)$$

When a buffer is filling, the processor keeps writing intermediate data into the buffer and in the mean time the spilling process keeps writing the sorted data from the buffer to hard disk. Therefore the filling speed of a buffer can be represented by $P_r \times R_a - H_d$. Thus the time to fill up a buffer can be computed by $\frac{B_f}{P_r \times R_a - H_d}$. As a result, for a buffer to be filled up, the processor will generate a volume of intermediate data with the size of $V_b$ which can be computed using equation (4.6)

$$V_b = P_r \times R_a \times \frac{B_f}{P_r \times R_a - H_d} \qquad (4.6)$$

The total amount of intermediate data generated from the original data chunk with a size of $D_m$ is $D_m \times R_a$. Therefore the number of times for a buffer to be filled up can be computed using equation (4.7).

$$N_b = \frac{D_m \times R_a}{V_b} \qquad (4.7)$$

The time for a buffer to be spilled once is $\frac{B_f}{H_d}$, therefore the time for a buffer to be spilled for $N_b$ times is $\frac{N_b \times B_f}{H_d}$. Then we have

$$t_b = \frac{N_b \times B_f}{H_d} \qquad (4.8)$$

The frequencies in processing intermediate data $N_f$ can be computed using equation (4.9).

$$N_f = \left\lfloor \frac{N_b}{s} \right\rfloor - 1 \qquad (4.9)$$

When the merging occurs once, the whole volume of intermediate data will be written into the hard disk causing an overhead of $\frac{D_m \times R_a}{H_d}$. Thus if the merging occurs $N_f$ times, the time consumed by hard disk IO operations can be represented by $\frac{D_m \times R_a \times N_f}{H_d}$. We have

$$t_m = \frac{D_m \times R_a \times N_f}{H_d} \qquad (4.10)$$

The total time $T_{total}$ to process data chunks in one processing wave in MapReduce Hadoop is the maximum time consumed by $k$ participating *mappers*, where

$$T_{total} = \max(T_1, T_2, T_3, \ldots, T_k) \qquad (4.11)$$

According to divisible load theory [102] [103] [104] [105], to achieve a minimum $T_{total}$, it is expected that all the *mappers* to complete data processing at the same time:

$$T_1 = T_2 = T_3 = \ldots = T_k \qquad (4.12)$$

Let

- $T_i$ be the processing time for the $i^{th}$ *mapper*.
- $\bar{T}$ be the average time of the $k$ *mappers* in data processing, $\bar{T} = \frac{\sum_{i=1}^{k} T_i}{k}$

Based on equations (4.11) and (4.12), the fitness function is to measure the distance between $T_i$ and $\bar{T}$. Therefore, the fitness function can be defined using equation (4.13) which is used by the genetic algorithm in finding an optimal or a near optimal solution in determining the size for a data chunk.

$$f(T) = \sqrt{\sum_{i=1}^{k}(\bar{T} - T_i)^2} \qquad (4.13)$$

### 4.2.2 Crossover

To maintain the diversity of the chromosomes, the algorithm needs functions of crossover. Crossover recomposes the homologous chromosomes via mating to generate new chromosomes or so called offspring. The generated offspring inherit the basic characteristics of their parents. Some of them may adapt to the fitness function better than their parents did, so they may be chosen as parents in next generation. Based on crossover, the algorithm can keep evolving until an optimal offspring has been found. In this algorithm, to gain the effective of design and operations, single-point crossover which refers to set only one crossover point randomly in the chromosome has been employed. The processes of crossover could be regarded as:

1. Randomly select pairs of the chromosomes (schedulers) as parents to mate.
2. In each pair, randomly select a position as crossover point. If the length of the chromosome is $k$ then there will be $k-1$ available points.
3. In each pair, the chromosomes change their parts which are after the crossover point with each other according to crossover probability $p$ .

However in the algorithm simply crossing the chromosome may cause one problem. As each gene is the value of the actual volume of data each Map instance takes, to change the members of genes may differentiate the original total volume of data $\sum_{i=1}^{k} D_i$ .

Assume the original total volume of data is $\sum_{i=1}^{k} D_i$ and the volume of data after crossover is $\sum_{i=1}^{k} d_i$ , then the difference $\Delta D = \left| \sum_{i=1}^{k} D_i - \sum_{i=1}^{k} d_i \right|$ should be considered and processed. In the algorithm $\Delta D$ is divided into $k$ parts. The size of each part is randomly assigned. And then these $k$ parts will be randomly added to or removed

from $k$ genes in the chromosome. Thus the total size of processed data in one wave could be guaranteed.

### 4.2.3 Mutation

To avoid the local optimum of the algorithm, mutation has been introduced into our algorithm. Mutation could mutate genes in a chromosome based on smaller probabilities. Moreover new individuals could be generated. So that combined with crossover the information loss due to the selection could be avoided. Thus the validity of the algorithm could be guaranteed. The mutation contributes in two main aspects in our algorithm.

1. Improving the local search ability of the algorithm. The crossover operation could find a number of chromosomes with better adaptability from a global angle. These chromosomes are close to or helpful to gain the optimal solution. However crossover cannot execute local search in details. So using mutation to tune the values of certain genes from local detailed phase could make the chromosome much closer to the optimal solution. So the search ability is enhanced compare to that of only crossover involved.
2. Maintaining the diversity of the colony moreover preventing the premature convergence of the algorithm. Mutation replaces the original genes with newly mutated genes so that the structure of a chromosome could be significantly affected. The diversity of the colony could be maintained.

The algorithm mutates genes mainly based on simple mutation which refers that to mutate one or several genes in the chromosome based on mutation probability $p$. There are two steps in the simple mutation.

1. Randomly select a gene to be the mutation point. Base on mutation probability $p$ to decide if the chromosome mutates.

2. If the probability decides the gene should mutate, then the value of the gene will be mutated which means a new value replaces the original value. As a result a new individual is generated.

However, it is quite similar to crossover processes that when the value of one gene mutates, the original total volume of data $\sum_{i=1}^{k} D_i$ has been changed. Assume the original volume of the gene is $D_i$ and the volume after mutation is $d_i$, then the difference $\Delta D = |D_i - d_i|$. To solve $\Delta D$ issue, $\Delta D$ is divided into $k$ parts. The size of each part is randomly assigned. And then these $k$ parts will be randomly added to or removed from $k$ genes in the chromosome. Thus the total size of processed data in one wave could be guaranteed. Based on this design, the algorithm has a strong ability to change its searching direction to gain the optimal solution in a large search space.

## 4.3 Experimental Results

To evaluate the performances of MR-LSI a small scale Hadoop cluster consisting four computer nodes has been set up. Table 4.1 shows the configurations of the Hadoop cluster.

Table 4.1: The experimental environment.

| | |
|---|---|
| Number of Hadoop nodes: | 4 |
| Nodes' specifications: | Three Datanodes: CPU Q6600@2.4G, RAM 3GB and running OS Fedora 11.<br>One Namenode: CPU C2D7750@2.26G, RAM 2GB and running OS Fedora 12. |
| Number of *mappers* per node: | 2 |
| Number of reducer: | 1 |
| Network bandwidth: | 1000Gbps |

To evaluate the performances of MR-LSI, 1000 papers were collected from the IEEE XPlore data source. For each paper selected, a T-D matrix will be constructed. In the

tests, also two strategies Closest Distance Searching (CDS) and All Distances Searching (ADS) for clustering documents which are similar to the clustered strategies proposed in [14] have been designed.

Processed by *k-means*, the original dataset is clustered into a number of sub-clusters. Within these sub-clusters, one or a few of them may be close to the query while the others are far away from the query. CDS calculates the distances between a query $q$ and the centroid of each sub-cluster. The closest sub-cluster to the query $q$ will have the highest probability in containing the target documents. A truncated SVD will only be performed on the closest sub-cluster. As CDS just retrieves information in one cluster, the time consumed for executing CDS is least. ADS calculates the distance between a query and the centroid of each sub-cluster, and a truncated SVD will be performed on all the sub-clusters. As ADS retrieves information in all sub-clusters, the misclassified documents may have chance to be retrieved.

### 4.3.1 Evaluating MR-LSI

MR-LSI was evaluated from the aspects of precision and recall in comparison with standalone LSI, standalone LSI combined with *k-means* using the CDS strategy, and standalone LSI combined with *k-means* using the ADS strategy. From the results presented in Figure 4.1 and Figure 4.2 it can be observed that the performance of MR-LSI is close to that of the standalone LSI. It is worth pointing out that the CDS strategy only works on the closest sub-cluster of documents related to a query. Compared with other algorithms, CDS retrieves a smaller number of documents which resulting in lower performance in recall.

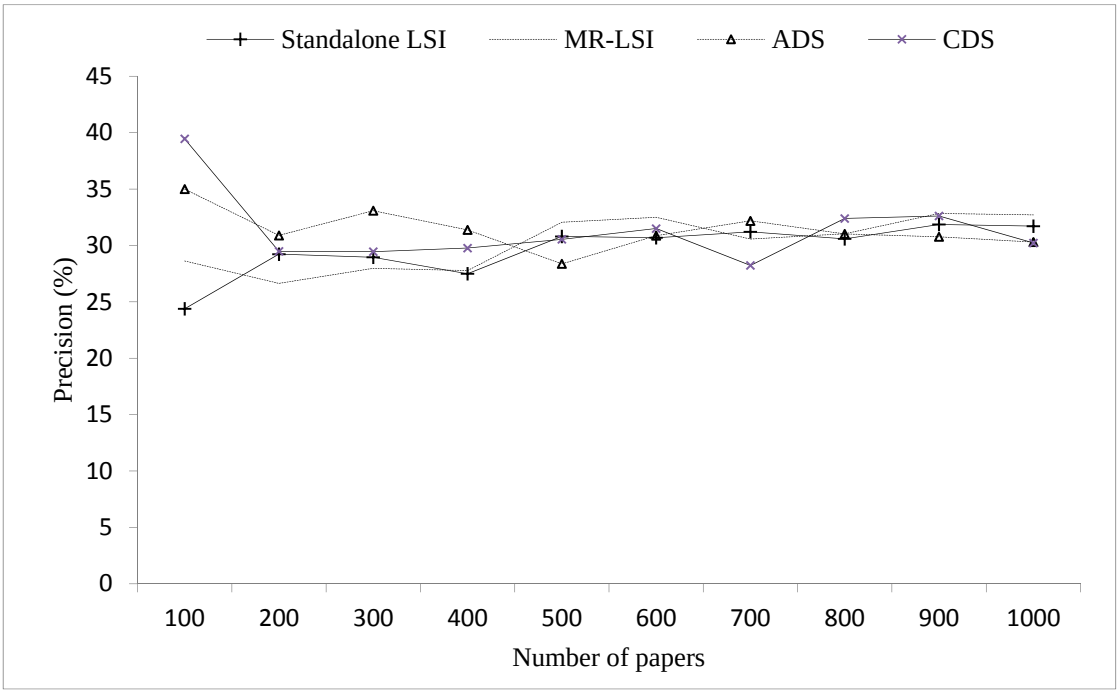


Figure 4.1: The precision of MR-LSI.

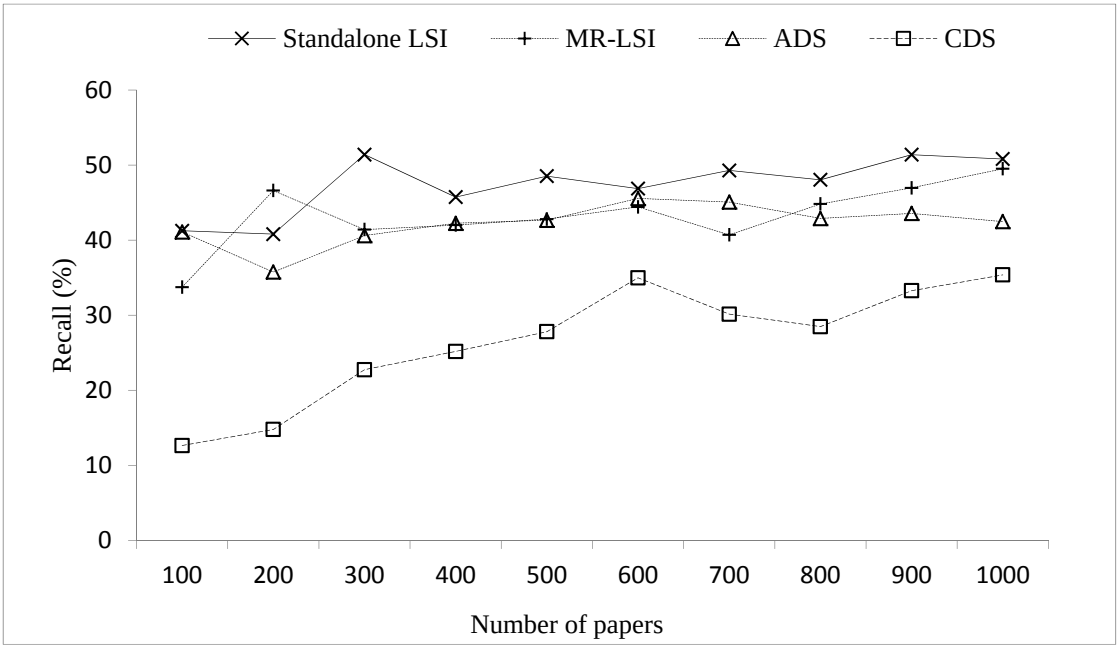


Figure 4.2: The recall of MR-LSI.

There are a number of tests have been conducted to evaluate the overhead of MR-LSI in computation. The number of documents to be retrieved varied from 100 to 1000. However, the size of the dataset was not large. From Figure 4.3 and Figure 4.4 it can be seen that MR-LSI consumed more time than other algorithms in processing the

dataset. This is mainly due to the overhead generated by the Hadoop framework which is effective in processing large scale data. Both the ADS and the CDS strategies perform faster than the standalone LSI indicating the effectiveness of a combination of LSI with *k-means*.

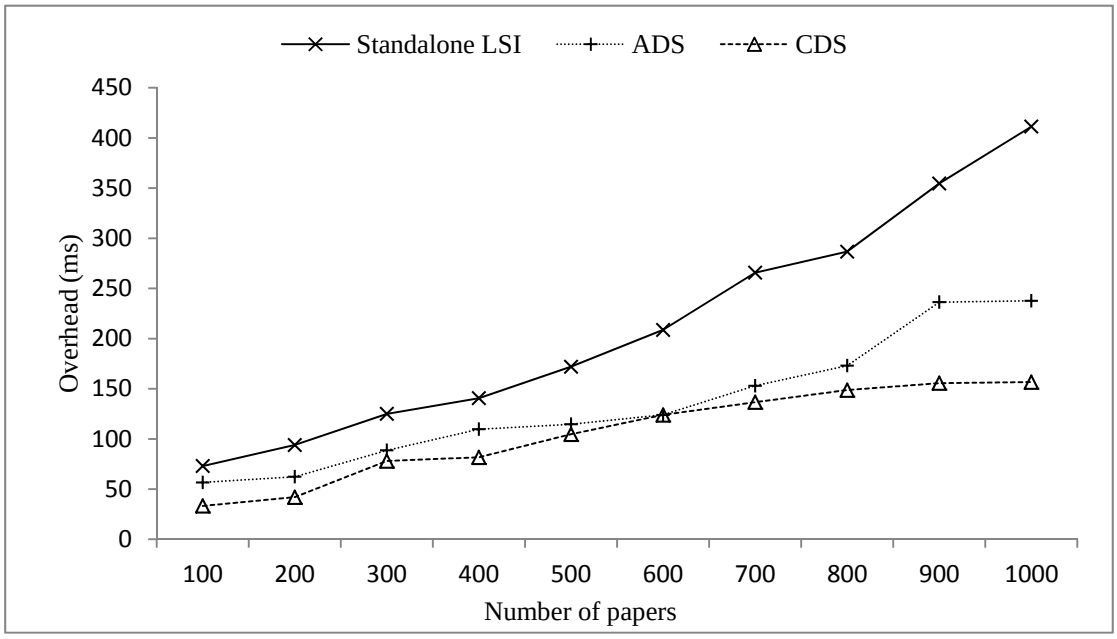


Figure 4.3: The overhead of standalone LSI, ADS and CDS in computation.

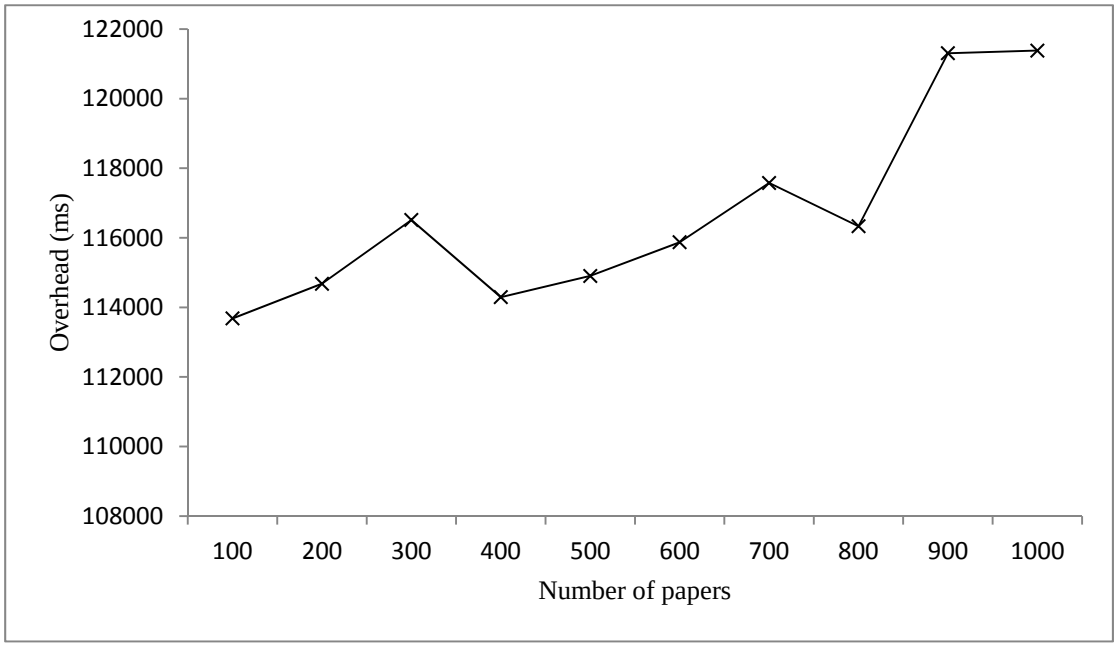


Figure 4.4: The overhead of MR-LSI.

And also a number of additional tests have been as well conducted to further evaluate

the overhead of MR-LSI in processing a large collection of documents. The size of the document collection is increased from 5KB to 20MB and the overhead of MR-LSI with that of the CDS strategy is compared as CDS is faster than both the standalone LSI and the ADS strategy. From the results plotted in Figure 4.5 it can observed that when the data size is less than 1.25MB, the overhead of CDS is stable. However, the overhead of CDS starts growing when the size of dataset is larger than 2.5MB. When the size of data reaches to 10MB, the overhead of CDS increases sharply. Compared with CDS, the overhead of MR-LSI is highly stable with an increasing size of dataset shows its better scalability than the CDS strategy. It also should be mentioned that when the size of data increases higher than 20MB, the heap space exception occurs when CDS processes data due to the memory limitation of applications in a standalone node.

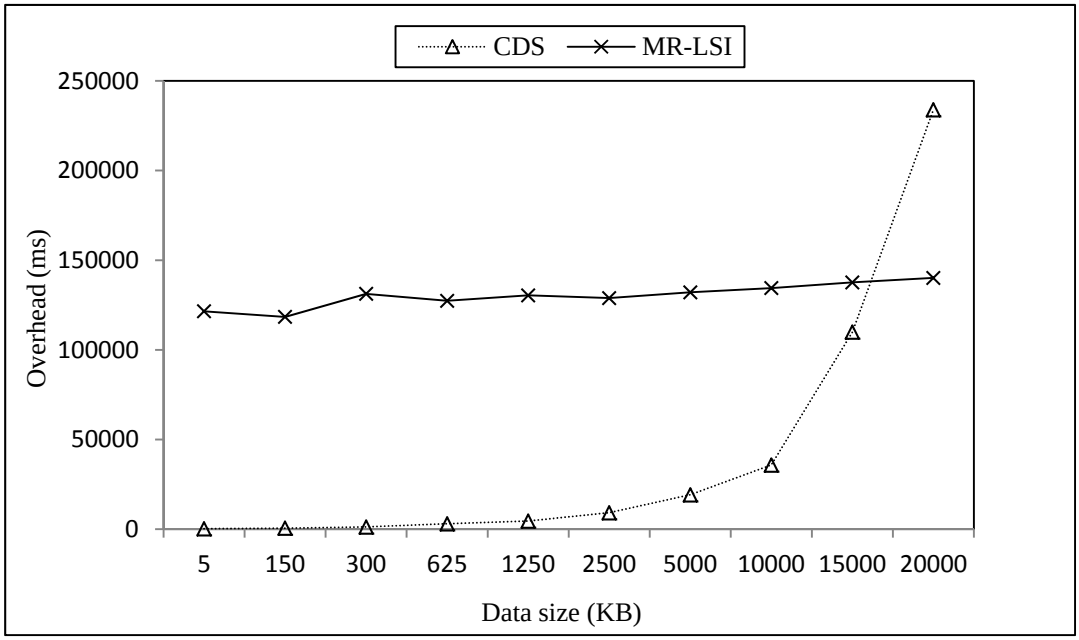


Figure 4.5: Comparing the overhead of MR-LSI with CDS.

## 4.4 MR-LSI Simulation Results

To further evaluate the effectiveness of MR-LSI in large scale MapReduce environments, HSim has been developed using pure JAVA programming language. This chapter accesses the performance of the MR-LSI in simulation environments.

### 4.4.1 Simulation Results

To study the impacts of Hadoop parameters on performance of MR-LSI, a cluster has been simulated with the configurations as shown in Table 4.2. Each node has a processor with 4 cores. The number of *mappers* is equal to the number of processor cores. There are two *mappers* running on a single processor with two cores. The speeds of the processors were simulated in terms of the volume of data in MB processed per second. In the following sections, the impacts have been shown of a number of Hadoop parameters on the performance of MR-LSI.

Table 4.2: The simulated environment.

| | |
|---|---|
| Number of simulated nodes: | 250 |
| Data size: | 100,000MB |
| CPU processing speed: | Up to 0.65MB/s |
| Hard drive reading speed: | 80MB/s |
| Hard drive writing speed: | 40MB/s |
| Memory reading speed: | 6000MB/s |
| Memory writing speed: | 5000MB/s |
| Network bandwidth: | 1Gbps |
| Number of mappers: | 4 per node |
| Number of reducers: | 1 or more |

#### *4.4.1.1 Multiple Reducers in One Node*

From Figure 4.6 it shows that the number of *reducers* does not affect the performance of *mappers* greatly. This is because mappers and reducers work almost independently in Hadoop environments.

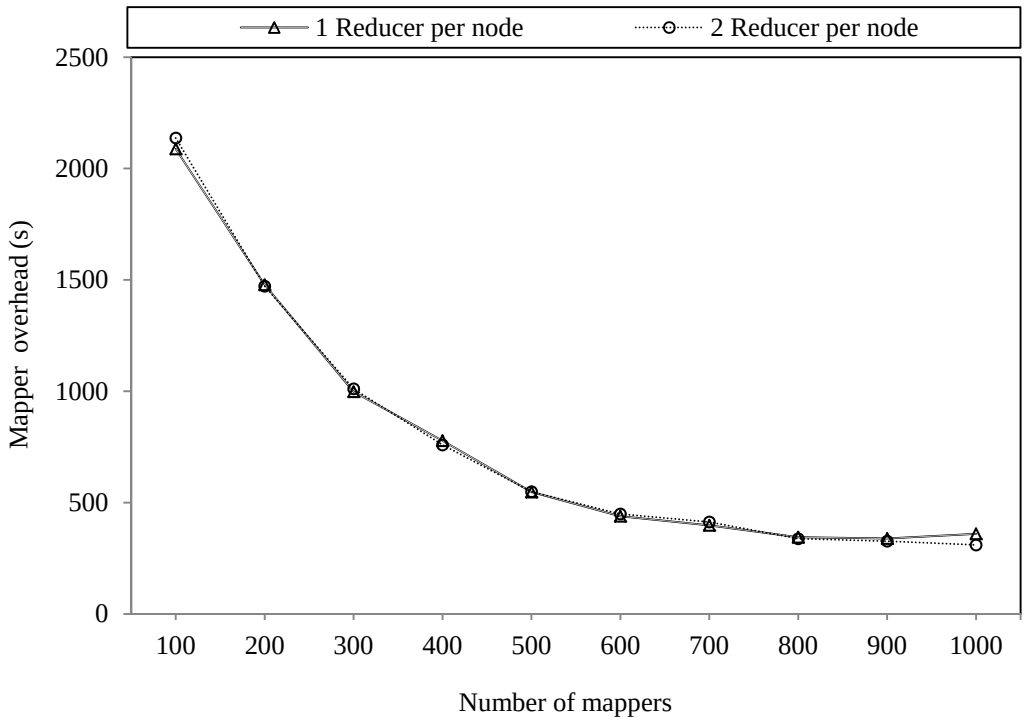


Figure 4.6: The impact of the number of reducers on *mapper* performance.

Figure 4.7 shows the impact of the number of *reducers* on the overall overhead when processing a job. Allocating multiple *reducers* on one node increases results in the shared resources issue. Especially for MR-LSI a number of hard disk operations involved, the shared hard disk gives worse performance in reducing phase of the *reducers* than that of unshared hard disk.

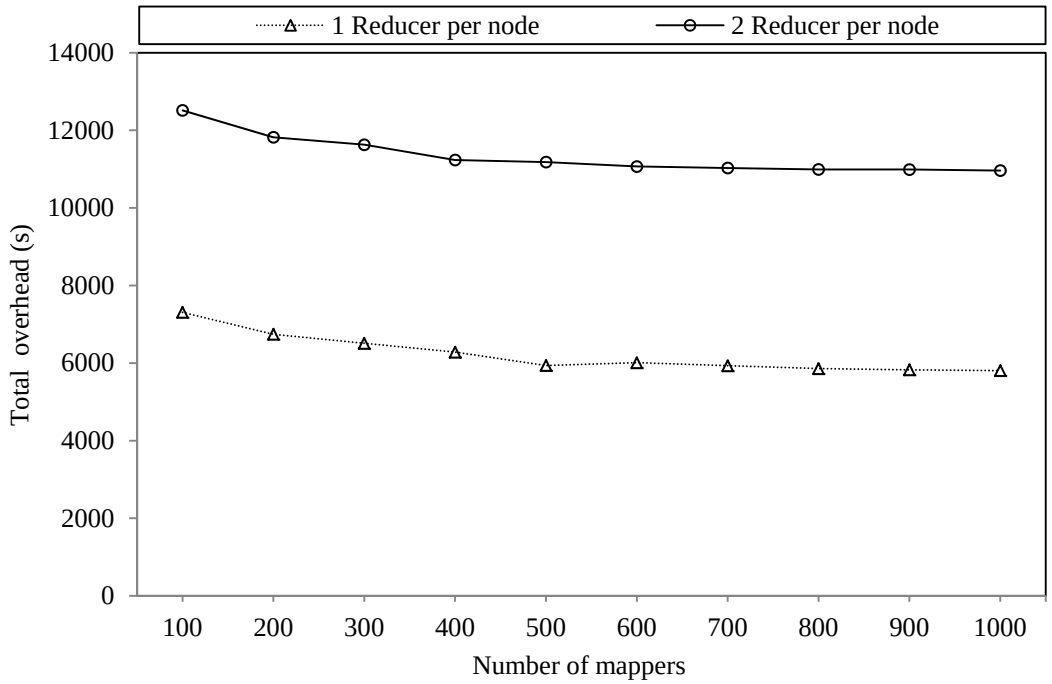


Figure 4.7: The impact of the number of reducers on the total process.

#### *4.4.1.2 Sort Factor*

In Hadoop, The parameter of sort factor controls the maximum number of data streams to be merged in one wave when sorting files. Therefore, the value of sort factor affects the IO performance of MR-LSI. From Figure 4.8 it can be observed that the case of using sort factor 100 gives a better performance than sort factor 10. When the value of sort factor is changed from 10 to 100, the number of spilled files will be increased which reduces the overhead in merging.

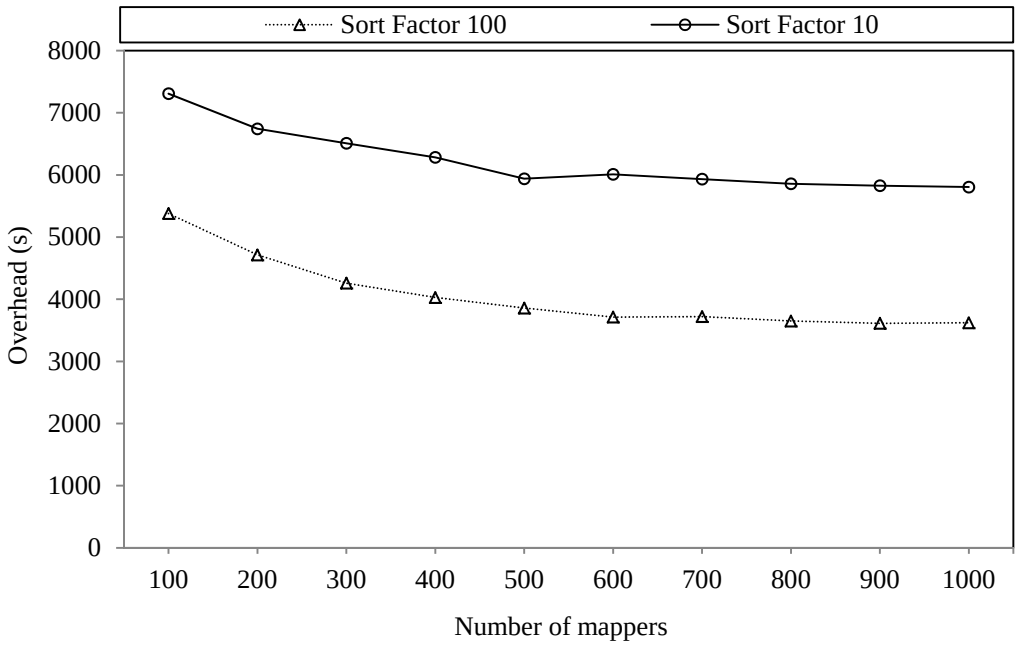


Figure 4.8: The impact of sort factor.

#### *4.4.1.3 Buffer Size*

The buffer size in Hadoop contributes to IO performance, and it affects the performance of a processor. The default value of a buffer size is 100MB. The performance of MR-LSI with a data size of 1000MB is tested. As shown in Figure 4.9, the *mappers* generate a small number of spilled files when using a large size buffer which reduces the overhead in merging. Furthermore, a large buffer size can keep the processor working without any blocking for a long period of time.

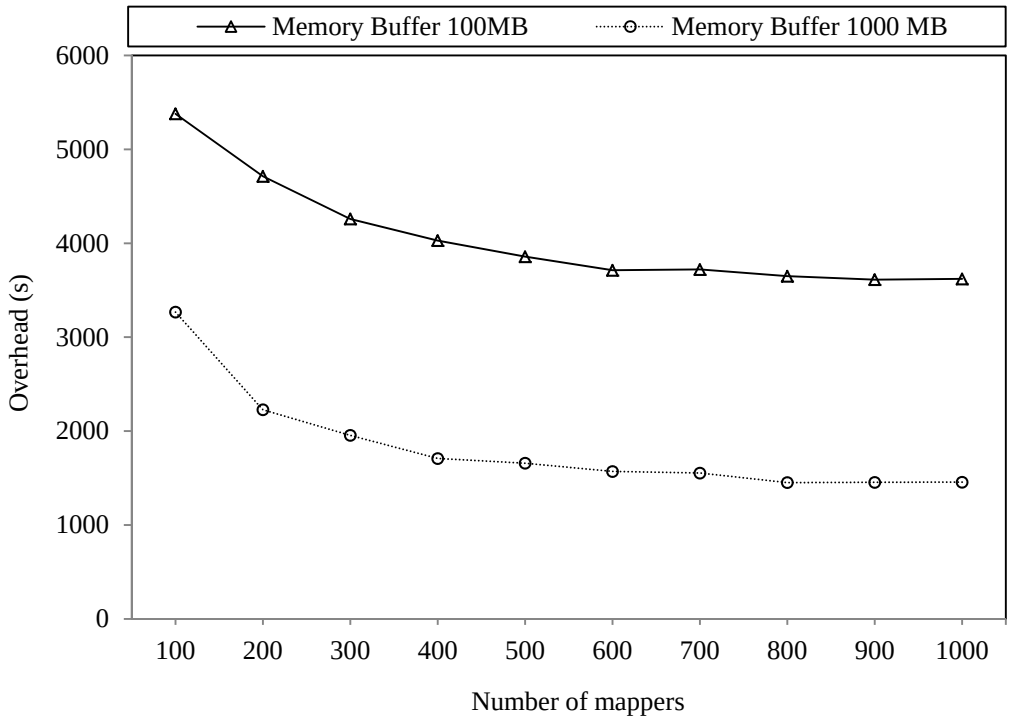


Figure 4.9: The impact of buffer size.

*4.4.1.4 Chunk Size*

Each *mapper* processes a data chunk at a time. Thus the size of data chunks highly affects the number of processing waves of *mappers*. From Figure 4.10 it can be observed that using a large size for data chunks reduces the overhead of mappers in processing, and also reduces the total overhead of the process as shown in Figure 4.11. However, both of the two chunk sizes produce the same performance when the number of *mappers* increases to 800 and 900 respectively. In the case of chunk size 64MB, to process 100,000MB data, using 800 *mappers* needs $\left\lceil \frac{100{,}000MB}{800\times 64MB} \right\rceil = 2$ waves to finish the job. In the case of chunk size 100MB, using 800 *mappers* needs $\left\lceil \frac{100{,}000MB}{800\times 100MB} \right\rceil = 2$ waves to finish the job. Similarly, using 900 *mappers* needs 2 waves to process the 100,000MB data in both cases. When the number of *mappers* reaches 1000, the performance of the two cases with different data sizes varies.

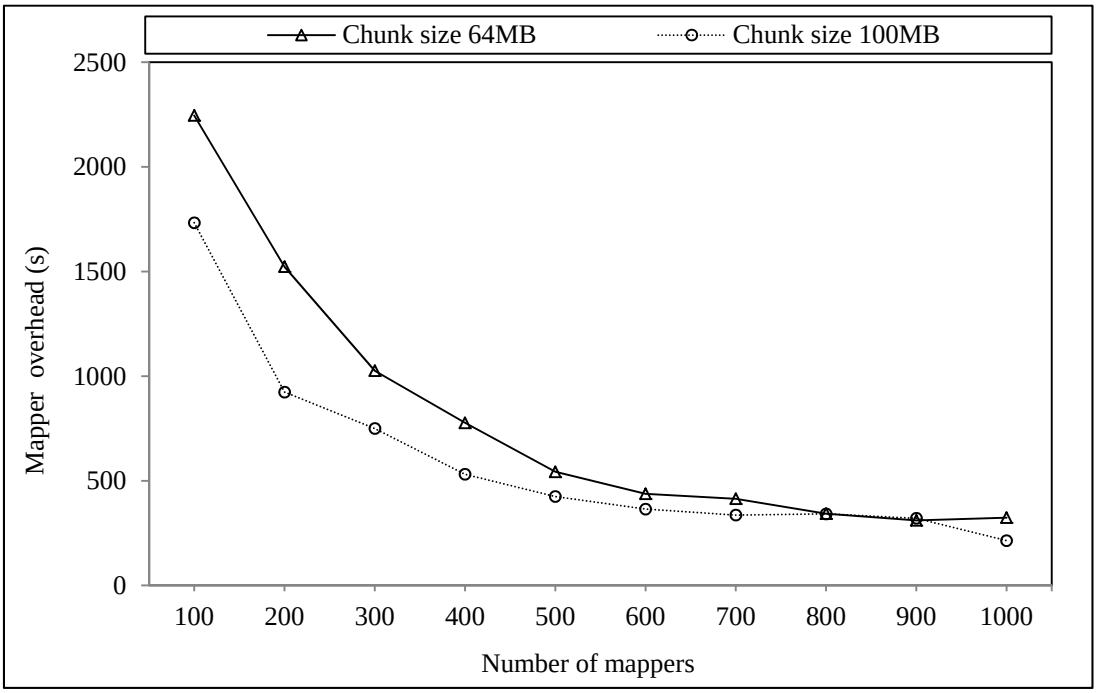


Figure 4.10: The impact of data chunk size on the *mappers* in MR-LSI.

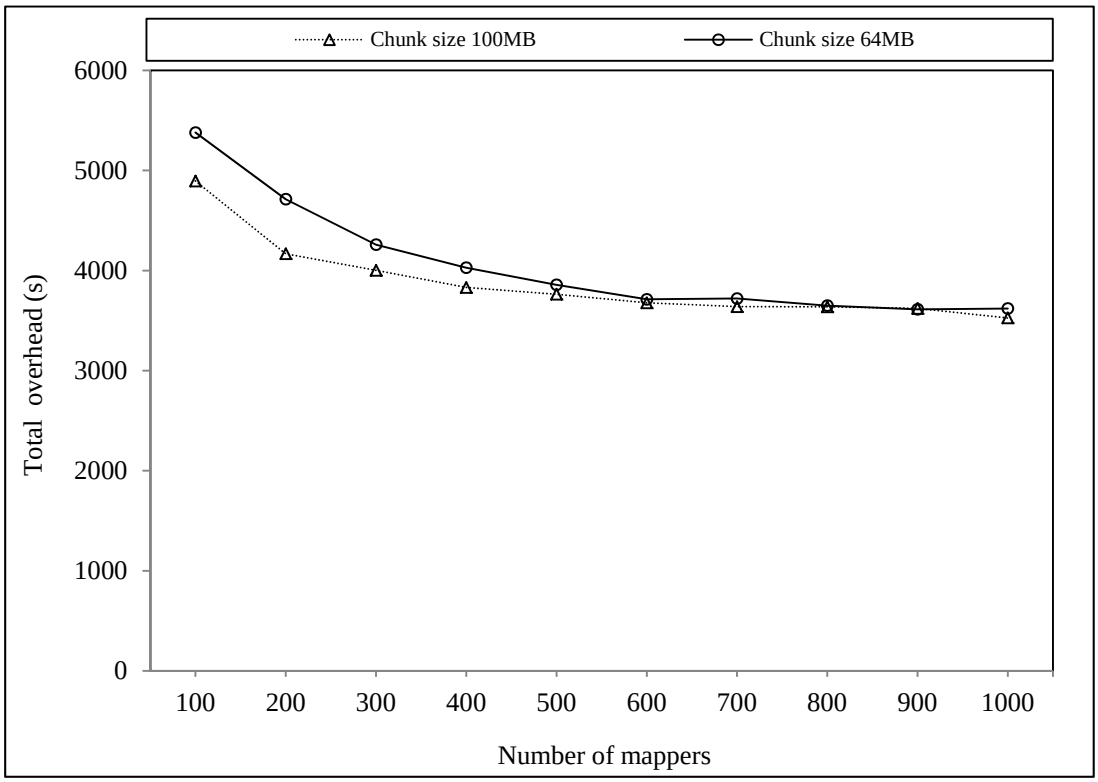


Figure 4.11: The impact of data chunk size on MR-LSI.

### *4.4.1.5 CPU Processing Speed*

Figure 4.12 shows the impacts caused by different processing speed of processors.

From the figure we can observe clearly that a faster processor can gain better performance compared to that of a slower processor.

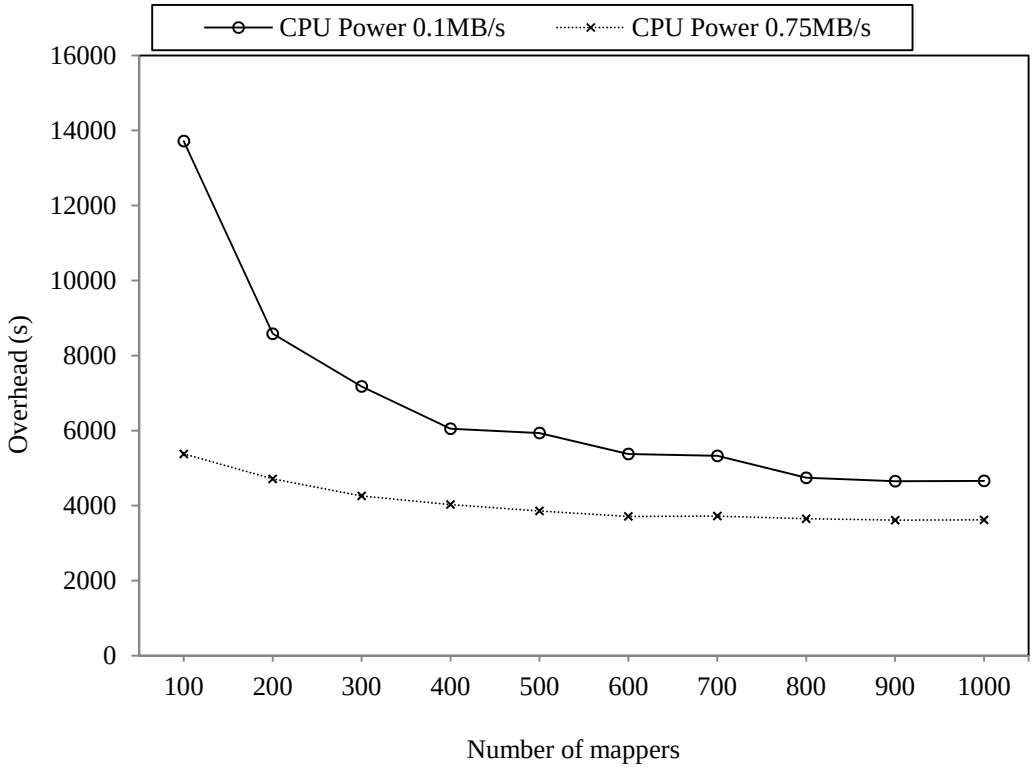


Figure 4.12: The impact of different CPU processing speeds

*4.4.1.6 Number of Reducers*

Figure 4.13 shows that increasing the number of *reducers* enhances the performance of MR-LSI when the number of *reducers* is small. More reducers are used more resources will need to be consumed due to Hadoop's management work on the *reducers*. In some cases multiple *reducers* need an additional job to collect and merge the results of each *reducer* to form a final result. This can also cause larger overhead.

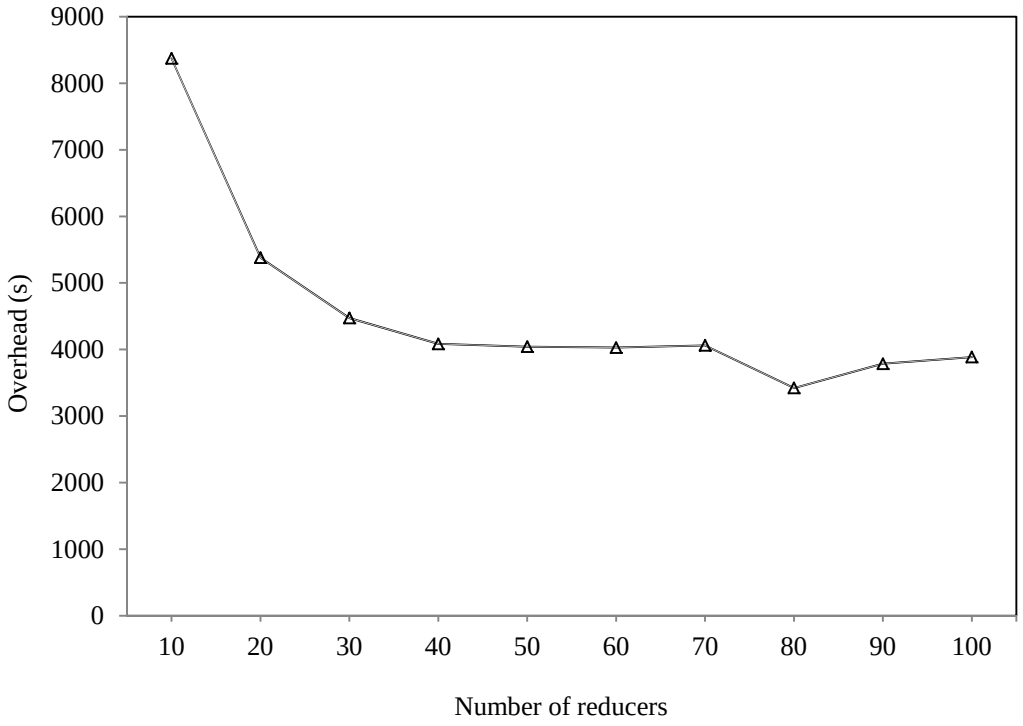


Figure 4.13: The impact of reducers.

### 4.4.2 Load Balancing Simulation Results

Table 4.3 shows the configurations of the simulated Hadoop environments in evaluating the effectiveness of the load balancing scheme of MR-LSI.

Table 4.3: Hadoop simulation configuration.

| | |
|---|---|
| Number of simulated nodes: | 20 |
| Number of processors in each node: | 1 |
| Number of cores in each processor: | 2 |
| Size of data: | Test 1: 10GB<br>Test 2: 10GB to 100GB |
| The processing speeds of processors: | Depending on heterogeneities |
| Heterogeneities: | From 0 to 2.28 |
| Number of hard disk in each node: | 1 |
| Reading speed of hard disk: | 80MB/s |
| Writing speed of hard disk: | 40MB/s |
| Number of Map instances: | Each node contributes 2 Map instances. |
| Number of Reduce instances: | 1 |
| Sort factor: | 100 |

To evaluate the load balancing algorithm, a cluster with 20 computers has been

simulated. Each computer has one processor with two cores. The number of *mappers* is equals to the number of processor cores. Therefore two *mappers* are running on a single processor with two cores. The speeds of the processors are generated based on the heterogeneities of the Hadoop cluster. In the simulation environments the total processing power of the cluster was $P = \sum_{i=1}^{n} p_i$ where *n* represents the number of the processors employed in the cluster and $p_i$ represents the processing speed of the $i^{th}$ processor. For a Hadoop cluster with a total computing capacity $P$, the levels of heterogeneity of the Hadoop cluster can be defined using equation (4.14).

$$Heterogeneity = \sqrt{\sum_{i=1}^{k} (\bar{p} - p_i)^2} \qquad (4.14)$$

In the simulation, the value of heterogeneity was in the range of 0 and 2.28. The reading and writing speeds of hard disk were generated based on the real measurements from the experiments conducted.

Firstly 10GB data has been tested in the simulated cluster with different levels of heterogeneity. From Figure 4.14 it can be observed that when the level of heterogeneity is less than 1.08 which indicates a nearly homogeneous environment, the load balancing scheme does not make any difference to the performance of MR-LSI. However, the load balancing scheme reduces the overhead of MR-LSI significantly with an increasing level of heterogeneity.

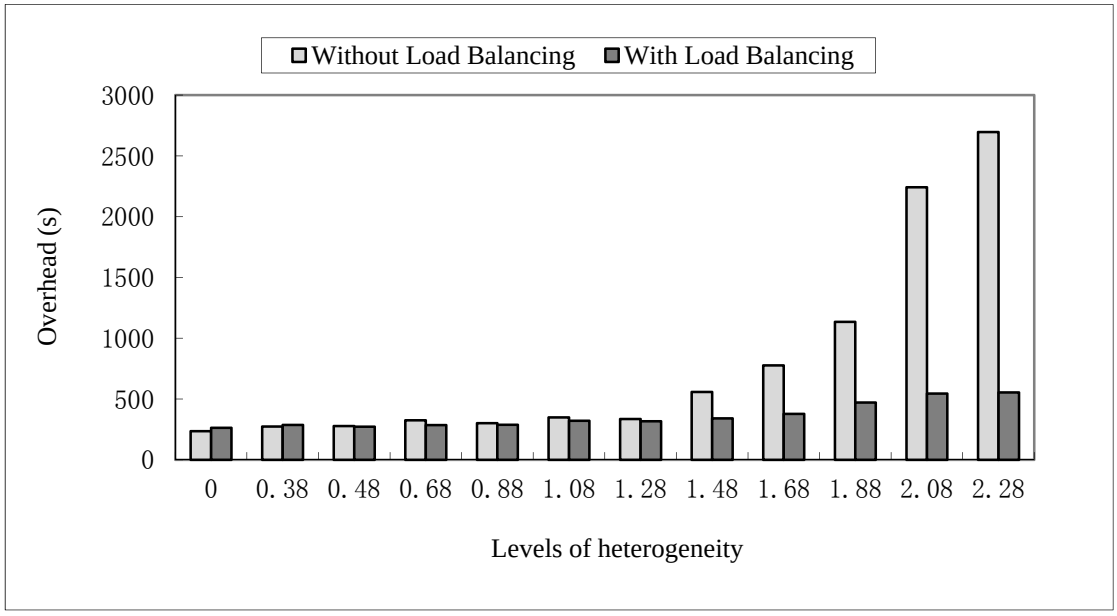


Figure 4.14: The performance of the load balancing scheme.

The levels of heterogeneity are keeping the same in the tests but varied the size of data from 1GB to 10GB. This set of tests was used to evaluate how the load balancing scheme performs with different sizes of datasets. Figure 4.15 shows that the load balancing scheme can always reduce the overhead of MR-LSI.

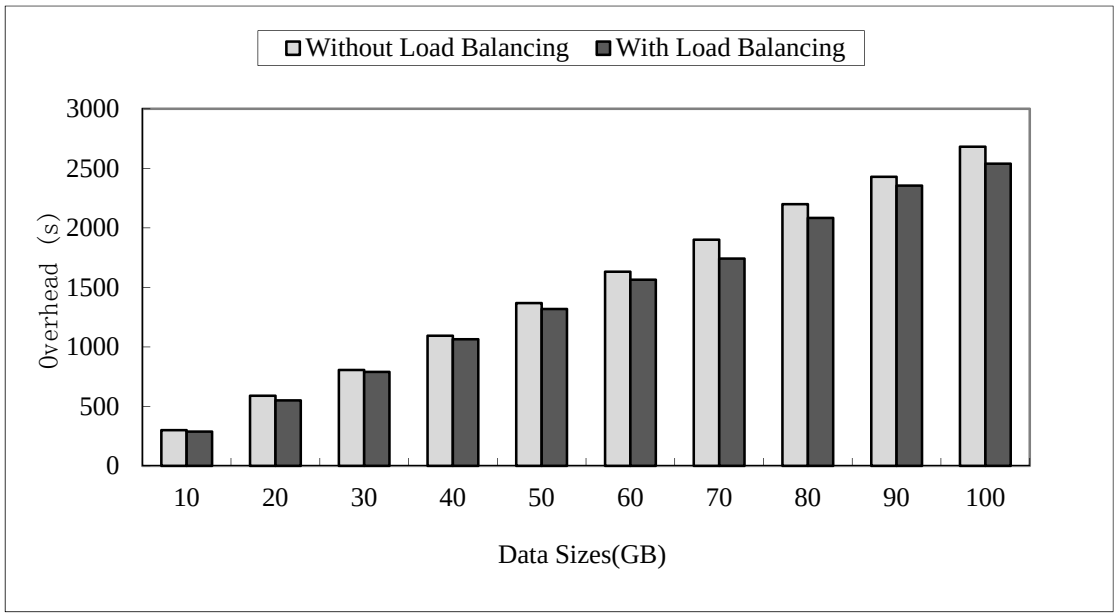


Figure 4.15: The performance of the MR-LSI with difference sizes of data.

The load balancing scheme builds on a genetic algorithm whose convergence affects the efficiency of MR-LSI. To analyze the convergence of the genetic algorithm, the

number of generations is varied and the overhead of MR-LSI in processing a 10GB dataset in the simulated Hadoop environment is measured. Figure 4.16 shows that MR-LSI reaches a stable performance when the number of generations in the genetic algorithm reaches 300.

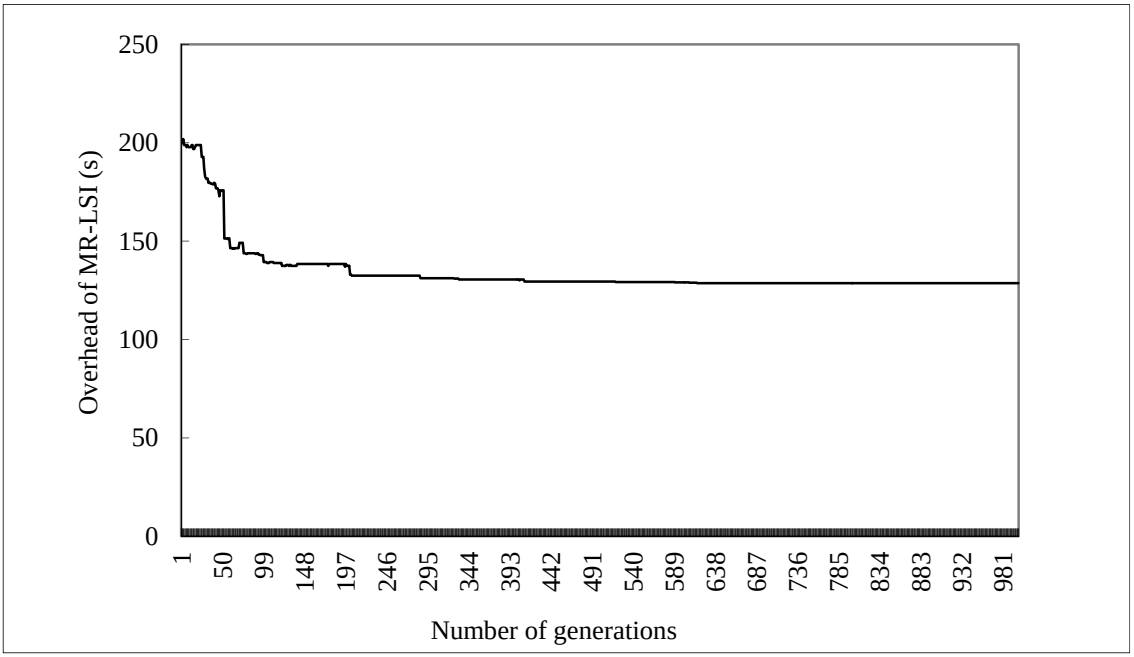


Figure 4.16: The convergence of the load balancing scheme.

The load balancing scheme also produces some overhead during execution. Figure 4.17 shows an increased overhead of the load balancing scheme when the number of *mappers* increases together with an increasing size of data. However the MR-LSI algorithm can still achieve benefit from load balancing algorithm. For example, for heterogeneity 2.08, the overhead of load balancing algorithm is 331s. The time consumed for one processing wave of *mappers* is 363s with load balancing. The time consumed for one processing wave of *mappers* is 2256s without load balancing. Thus the performance is enhanced 69.2%. As in the static computing environment, the scheduler only needs to be computed once, thus it can be claimed that for a long-time processing job with proper heterogeneities, the load balancing algorithm can enhance performances greatly.

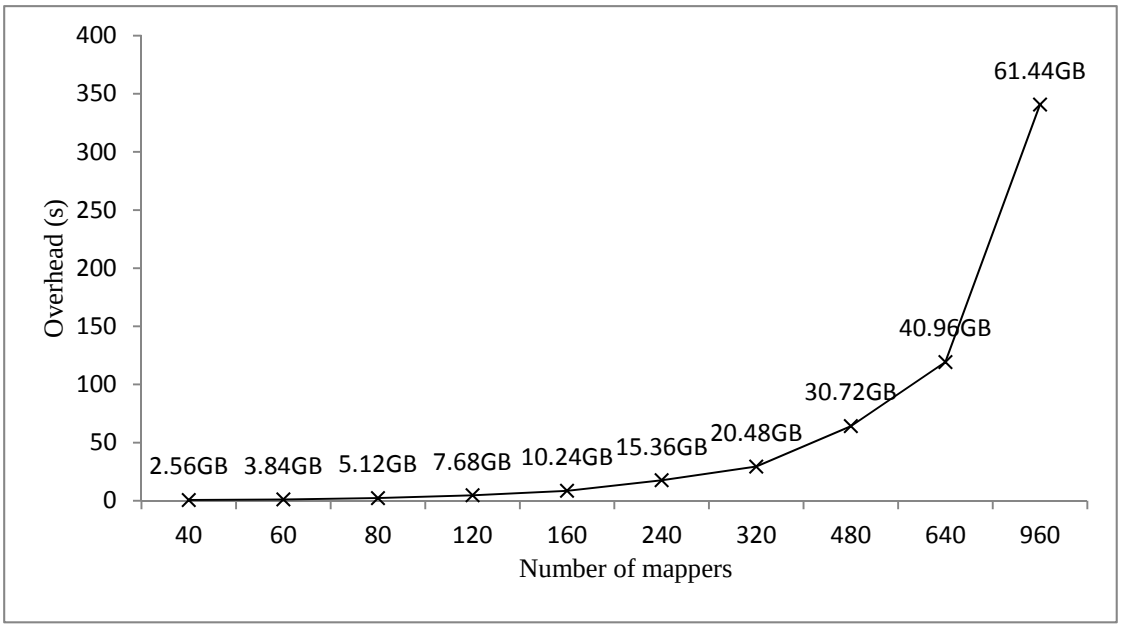


Figure 4.17: The overhead of the load balancing scheme with different sizes of data.

## 4.5 Summary

This chapter presents MR-LSI for scalable information retrieval. MR-LSI is effective when processing a large dataset due to high scalability of MapReduce in support of data intensive applications. Both experimental and simulation results have shown that the MR-LSI algorithm speeds up the computation process of SVD while maintaining a high level of accuracy in information retrieval. The simulating results also indicate that the load balancing strategy can enhance the performance of the Hadoop cluster when it is running a Hadoop application.

# Chapter 5

# Dynamic Load Balancing in Heterogeneous MapReduce Environments

The distributed computations are widely used in the modern world for processing large scale jobs. Hadoop framework which is based on Google MapReduce model becomes popular due to its great processing power and ease to use. However as the lack of load management, in a dynamic heterogeneous computing environment, the performance of Hadoop framework may be deteriorated. Therefore this chapter presents a dynamic load balancing algorithm which aims to balance the load among heterogeneous nodes. Due to the complexity of changing code of the Hadoop, the Hadoop simulator HSim is involved to evaluate the performance of the dynamic load balancing algorithm. The results indicate that the performances have been significantly enhanced due to the balanced load gained from the load balancing algorithm.

## 5.1 Load Balancing in Hadoop Framework

This section will be consisted by two sub sections. This first one states the reasons that why Hadoop framework needs load balancing and the second sub section states the current job schedulers employed by Hadoop framework.

### 5.1.1 Dynamic Load Balancing

It’s quite obvious that a large job can achieve performance enhancement when it’s processed in a distributed computing environment. At present the distributed computing systems have become more and more popular in data processing due to the reduced hardware costs and advanced computer network technologies. A standard distributed computing environment is normally consisted by a number of nodes with same or different dynamic computing capacities. The nodes are connected by different types of networks. Via the network, resources of the nodes can be share by a number of users or tasks. The distributed computing environment contains huge computing

capacity due to its mass amount of resources. Compare to an expensive standalone machine, no matter how much resource the machine has, it definitely will go over due to the occupations of large number of running applications. It has been pointed out that even with a proper scale of cheaper hardware, the overall computing power of the standalone nodes can easily exceed that of a supercomputer. However though a distributed computing environment can supply a huge adaptability to deal with mass data, it is obviously observed that if nodes have different deployments of hardware, software or networks, certain number of nodes may be overloaded and some others may be idle simultaneously. Thus the performance enhancement for distributed computing systems has become a key issue. The simplest way to enhance the capacity of a cluster is to add more nodes in which may result in high cost. However, it is easy to see that if people use advanced hardware and optimized software, the performance of the cluster will definitely be improved. This solution can solve the case that all nodes are overloaded, which is no matter some of them are faster and some of them are slower. However if the deteriorated performance is caused by the reason of unbalanced load among nodes due to their own individual and dynamic processing capacities, just simply enlarging the number of nodes or upgrading the hardware and software can hardly gain the expected efficiencies. Thus to improve the performance of the system in the way of redistributing load from the currently heavily loaded nodes to lightly loaded nodes in a heterogeneous and dynamic computing environment should be considered.

### 5.1.2 Unbalanced Load Issue in Hadoop Framework

As described earlier, Hadoop framework is designed to process large scale data in a distributed computing environment. As being claimed by Hadoop, the framework facilitates the developments of distributed computing based applications. These kinds of facilities are based on the interactions among three important components mainly which are named HDFS, Map instances (*mappers*) and Reduce instances (*reducers*). Though the overall structure of the Hadoop framework simplifies the processing, the components hide a lot of complex low-layer details including hardware and software

aspects at the background. At present, a number of well known Hadoop clusters are running in highly homogenous environments. For example Hadoop at Yahoo [100] employs a homogenous cluster with 4000 processors, 3T RAM and 1.5PB storage capacity. A large number of benchmarks and sorting competitions have been tested based on the environment. The results have been published to show the powerful of Hadoop framework. In these distributed computing tasks, people focus on the extreme performances using homogenous environment which can ideally avoid the unbalanced load issues. Therefore, behind these highlighted results, the load balancing issue is quite considerable of which has been hidden deeply by the homogeneous environments. Normally, it is extremely hard to build up a homogenous cluster with a number of nodes up to several thousands. As a result, a number of Hadoop clusters with heterogeneous nodes are quite common. The architecture of Hadoop framework has been designed quite flexible to adapt to heterogeneous resources. Thus, it can be seen clearly that the heterogeneities of the resources will affect the performance of the cluster. For instance various processing powers of processors, different writing and reading speeds of hard disks, different accessing times and seeking times of magnetic heads, different writing and reading speed of memory, different speeds of networks, and even different software deployments may vary the overall performance of a Hadoop cluster. A simple test has been done to evaluate the performance of the framework in a small heterogeneous environment which contained three machines and two out of the three were actual processing nodes. The faster machine has quad-core processor AMD Phenom II x4 940 BE@3.0GHz and a RAID0 storage system. The slower machine has a single core with hyper-threading processor Pentium 4@2.66GHz. Based on this heterogeneous cluster the MR-LSI algorithm has been executed in terms of evaluating the differences between two nodes. The result shows a huge gap between these two nodes: the time which faster machine spent on finishing its own map tasks is nearly five times faster than those of the slower machine spent on. This huge difference results in delay of finishing the job. It indicates clearly that the heterogeneity deteriorates the performance of the cluster due to the unbalanced workload. Devaraj Das [59], the engineering manager of Yahoo Bangalore Grid

Computing Group concludes the load issue from four aspects:

1. Imbalance in input splits
2. Imbalance in computations
3. Imbalance in partition sizes
4. Imbalance in heterogeneous hardware

So it can be clearly observed that there is a huge opportunity to enhance the performance of Hadoop framework in a way of balancing workload in a heterogeneous environment. To solve the unbalanced load issue, Hadoop framework employs a job scheduler which aims to balance the load among the nodes.

### 5.1.3 Current Load Balancing Policies in Hadoop Framework

Hadoop framework is designed to serve multiple jobs which are located in the job pool. In Hadoop, the framework supplies a simple job scheduler FIFO (First In First Out). The scheduler serves the jobs in order of their submissions. The sequential scheduler could ease the management of job to some extent and sometimes it is efficient when the framework deals with the job queue. The simpler scheduling algorithm only generates little overhead compared to those of complex balancing algorithms. So the job scheduler may response the jobs as quick as possible. However the drawback of the scheduler policy is quite obvious. As Hadoop framework prescribes, with the FIFO scheduler when a job is processed the job will occupy the whole computing resources across the cluster. The other jobs will never have chance to be processed when the job is running. Until the last job has been finished and then the next job in the queue will be served, which would use the whole cluster again. As we know different jobs need different amount of computing resources. Some light jobs may just need only little resources to deal with and in contrast some heavy jobs may need more resources. It is clearly shows that if a light weight job has been processed using the whole cluster, the hardware abuse may occur. In this case the performance of the cluster may be worse than using less computing resources. Simultaneously a heavy weight job which really needs lots of resources yet has to wait until the occupied resources are released. So the drawback of using scheduler

FIFO can be mainly summarized as: although a shared cluster offers great potential for offering large resources to multiple users, the problem of sharing resources fairly between users requires a better scheduling. Otherwise the performances of the cluster may be worse than expected due to the unfairly allocated resources. Therefore to avoid the above issue, the Hadoop framework offers new functions to control the priorities of different jobs. Now Via the mapred.job.priority property or the setJobPriority() method on JobClient five different hierarchies include: VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW. These four properties can control the selecting behaviors of the job scheduler. When the job scheduler finishes the current processed job and is ready to choose next to run, it will select a job with the highest property from the job pool based on the values of the priorities. Thus a number of jobs with higher priorities would be processed before those jobs with lower priorities. But one weakness is pointed out by [35]. With the FIFO scheduler in Hadoop, priorities do not support preemption. As a result, a high-priority job may has chance to be blocked by a low-priority job which starts before the high-priority job is scheduled.

According to the current job scheduling policy involved in Hadoop Framework, it is recognized that the scheduler in the framework can only schedule the jobs simply. Some important heterogeneous factors have not been considered by the Hadoop framework yet. Actually the basic heterogeneous factor is the processing capacity of *mappers*. And also, the computing capacity of a Hadoop cluster may be varying according to the utilities of the nodes. Thus, considering dynamic features, an advanced dynamic job scheduling algorithm which can balance the load among the most basic processing unit *mappers* is proposed in the later section.

## 5.2 Algorithm Design

### 5.2.1 Data Selection

In a dynamic distributed computing environment, the computing capacities of different of nodes are dynamically changing. Therefore, in a certain time interval the

total amount of computing capacity across the cluster may have a chance to stay at a higher level. Contrarily in another time interval, the total amount of computing resources across the whole cluster may be in a lower level. To use the higher computing capacity to process data can enhance the utilization of the cluster. Thus to find a time interval with higher computing capacity for a processing wave and the volume of data should be processed in this time interval are significant. In this chapter an approach that can approximately predict the computing capacity of the cluster [72], and the amount of data will be assigned to the cluster during the time interval has been proposed. Therefore, for each processing wave, the *mappers* have bigger chance to process data using higher computing capacities.

Let $f_i(t)$ represent the processing speed of $i^{\text{th}}$ mapper. Thus the total computing speed at the time point $t$ of the cluster employed a number of $n$ mappers $p$ can be represented by

$$p(t) = \sum_{i=1}^{n} f_i(t) \quad (5.1)$$

Let $D$ represent the total amount of data for a Hadoop job. Thus, considering only processing of processors, the overhead $t$ to complete processing the total amount of data can be represented by

$$\int_a^b p(t)dt = D \quad (5.2)$$

$$t = b - a \quad (5.3)$$

where $b$ is the finishing time and $a$ is the starting time.

Let $w$ represent the number of waves of mappers involved to process the total amount of data $D$. In the time interval $t$, there are a number of $n$ two types of trends of the processing speed $p(t)$. The first one is 'increasing trend'. We define it as below: from time point $t_a^{'}$ to $t_b^{'}$, the average processing speed during this time interval

$$\overline{p'_{ab}} = \frac{\int_{t'_a}^{t'_b} p(t)dt}{t'_b - t'_a} \quad (5.4)$$

keeps increasing until it becomes decrease. The second trend is 'decreasing trend'. It is defined as below: from time point $t'_a$ to $t'_b$, the average processing speed $\overline{p'_{ab}}$ during this time interval keeps decreasing until it becomes increase. Therefore the algorithm selects a number of $w$ greatest $\overline{p'_{abı}}$.

$$P_g = \{\overline{p'_{a1b1}}, \overline{p'_{a2b2}}, \overline{p'_{a3b3}}, \dots, \overline{p'_{awbw}}\} \quad (5.5)$$

After the greatest values selected, the algorithm starts merging the other two trends which are on the left and right sides of greatest $\overline{p'_{abı}}$ based on

$$\overline{p'_{a_{ı-1}b_{ı+1}}} = \frac{\int_{a_{i-1}}^{b_{i-1}} p(t)dt + \int_{a_i}^{b_i} p(t)dt + \int_{a_{i+1}}^{b_{i+1}} p(t)dt}{b_{i+1} - a_{i-1}} \quad (5.6)$$

Thus a number of $w$ new average values $\overline{p'_{a_{ı-1}b_{ı+1}}}$ are generated. Simultaneously the number of $n$ trends reduces to the number of $n - 2w$. Then in these $n - 2w$ trends, the algorithm selects a number of $w$ greatest $\overline{p'_{abı}}$ again and merges the other two trends which are on the left and right sides of them. Until there are a number of $w$ trends left. Thus the time intervals

$$T = \{(a_1b_1), (a_2b_2), (a_3b_3), \dots, (a_wb_w)\} \quad (5.7)$$

may have the higher average processing capacities.

Therefore, amount of data which is processed in the time interval $t = b_1 - a_1$ can be fed to the cluster to be actually processed. It can be expected to be processed within a higher computing capacity interval of the cluster. The amount of data $D_1$ can be represented by

$$D_1 = \int_{a1}^{b1} p(t)dt \quad (5.8)$$

However, due to the uncertainties and IO operations of the actual processing in the cluster, when the amount of data $D_1$ for the first wave is finished, the deviations of $b_1$ is inevitable. Thus the prediction for next wave should be corrected. The correction strategy is designed as following. In next wave, the algorithm re-executes the time interval computation as described above with changes of values of the parameters.

$$w = w - 1 \quad (5.9)$$

$$D = D - D_1 \quad (5.10)$$

Therefore with new values of $w$ and $D$, a new $D_1$ can be calculated and assign the amount of data to the second wave in processing. Until $w = 1$, the rest of the data is assigned to the last processing wave to be processed.

The following examples help to show how the algorithm works.

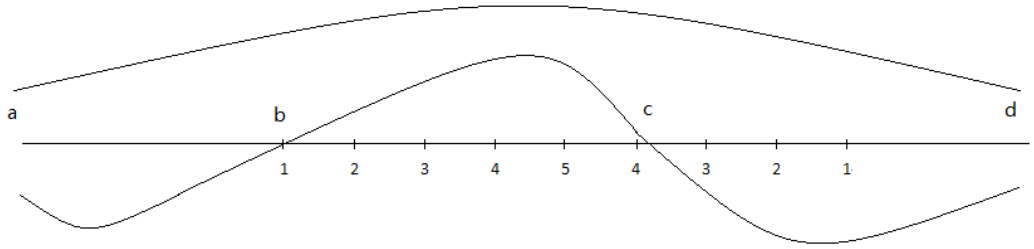


Figure 5.1: Example of computing time interval

In Figure 5.1 when the processing occurs in time point b to c, the average processing capacity is keeping increasing until 3.17. Thus the time interval $t_{bc}$ is regarded as increasing trend. Then from time point c to d, the processing capacity is keeping decreasing until next increasing trend appears. Thus the time interval $t_{cd}$ is regarded as decreasing trend. Similarly, the trend $t_{ab}$ which is before $t_{bc}$, is a decreasing trend as well. When the algorithm selects $t_{bc}$ as one of the most efficient time interval among a number of increasing trends, it merges $t_{bc}$ with $t_{ab}$ and $t_{cd}$ to expand the time interval according equation 5.6. And then the algorithm computes the average processing capacity of newly generated time interval $t_{ad}$ and marks it as efficient interval. Afterwards it selects a number of most efficient time intervals

among the rest increasing trends and newly generated efficient intervals. The algorithm keeps selecting and merging until a number of $w$ time intervals generated. And then the algorithm chooses the first interval as the approximately predicted efficient time interval and computes the volume of data fed to the cluster according to equation 5.8. As this prediction is based on pure processing capacities of processors, thus when the cluster is dealing with data, errors occur due to the delays caused by data IO. Figure 5.2 shows the deviation caused by data IO.

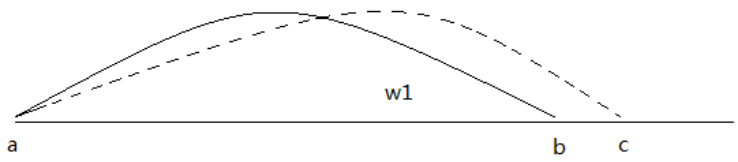


Figure 5.2: Example of deviation caused by IO.

From the figure, it can be observed that due to the IO operations, the actual processing time $t_{ac}$ is different from the predicted time $t_{ab}$. Thus for the next processing wave, the computed volume of data fed to the cluster will be less of accuracy. To solve this issue, the algorithm will select and merge time intervals again based on equation 5.6. However, as the first wave of processing is already finished, in this time the algorithm starts computing at time point c but not a according to equations 5.9 and 5.10.

The following pseudo code summarizes the steps of the algorithm in the Table 5.1.

Table 5.1: The pseudo code of the data selection

| *Algorithm: Data Selection* |
|---|
| 1. Compute the overall load of the cluster $p(t) = \sum_{i=1}^{n} f_i(t)$. |
| 2. Calculate time $t$ for processing the whole data. |
| 3. Calculate average value $\overline{p'_{ab}}$ for each trend. |
| 4. Select a number of $w$ greatest values of $\overline{p'_{ab}}$. |
| 5. Merge the two trends at both sides of the greatest $\overline{p'_{ab}}$. |
| 6. Calculate new $\overline{p'_{ab}}$ in the new time interval $b_{i+1} - a_{i-1}$. |
| 7. Re-select the number of $w$ greatest values of $\overline{p'_{ab}}$. |
| 8. Repeat 4, 5, 6, 7 |
| 9. Until the number of $w$ greatest values left. |
| 10. In the first time interval $(a_1, b_1)$ calculate the amount of data can be processed $D_1$. $D_1$ is assigned to the cluster to be actually |

processed.
11. In next wave of processing $w = w - 1$ and $D = D - D_1$.
12. Re-execute the algorithm until $w = 1$.

It should be pointed out that the algorithm using to feed amount of data into the cluster should has one premise. It is that $p(t)$ cannot be monotonically increasing function or monotonically decreasing function. Otherwise the data will be equally separate to a number of $w$ portions which will be fed to a number of $w$ processing waves.

### 5.2.2 The Design of Load Balancing Functions

The embedded FIFO scheduler in Hadoop aims to serve a job queue. However for one Hadoop job, the framework cannot deal with it well, which means the Hadoop cluster cannot manage the heterogeneous resources well for the job. In the reason of four aspects stated by [59], for a specified Hadoop job, to balance its work load we need to consider proper partition sizes, computations and heterogeneous hardware in a dynamic computing environment which contains dynamic CPU processing ability and dynamic IO ability. Before we introduce the way of the algorithm design we should point out one thing in advance that in the practical Hadoop cluster, the size of data chunks should be the same according to the configuration. However to implement the algorithm, the simulator has been expanded to support different sizes of data chunks. When the system starts computing the optimized scheduler, a central job dispatcher located in the JobTracker will execute the load balancing algorithm based on the volume of data $D_1$ assign to the current processing wave.

In a processing unit which is called Map instance (*mapper*), for processing one data chunk of a Hadoop job, the total processing time could be considered by:

$$T = t_1 + t_2 + t_3 + t_4 \quad (5.11)$$

In the equation, $t_1$ represents the copying time; $t_2$ represents the processor processing time; $t_3$ represents the emptying time when the buffer of the Map instance is filled up; $t_4$ represents the merging time.

For the copying time $t_1$:

Let

- $k$ represent the number of Map instances employed in the cluster.
- $D_m$ represent the corresponding volume of data assigned to m$^{th}$ Map instance.
- $H_w(t)$ represent the writing speed of the hard disk.
- $B$ represent the bandwidth of the network.

$$\sum_{m=1}^{k} D_m = D_1 \quad (5.12)$$

$$t_1 = \frac{D_m}{\min(H_w(t), B)} \quad (5.13)$$

For the processing time $t_2$:

Let

- $p = f(x)$ represent the dynamic CPU processing power of the Map instance where $x$ is time points.
- $a$ represent the start of the copying time of the Map instance.
- $t_{si}$ represent the time of filling up of the buffer.
- $t_{ei}$ represent the time of finishing spilling operation of the buffer.
- $t_{2i}$ represent the processing time of the processor during two blocking intervals.
- $D_{m_i}$ represent the volume of data which the processor can process during two blocking intervals.
- $n$ represent the number of spilled files during the processing.
- $B_f$ represent the size of the buffer.

The Map instance starts copying data at time point $a$ then after time $t_1$ the

processor starts processing the copied data. For processing the whole volume of data $D_m$, several equations should be satisfied.

$$\begin{cases} D_{m_1} = \int_{a+t_1}^{ts1} f(t)dt \\ D_{m_2} = \int_{te1}^{ts2} f(t)dt \\ D_{m_3} = \int_{te2}^{ts3} f(t)dt \\ \quad . \\ \quad . \\ \quad . \\ D_{m_n} = \int_{te(n-1)}^{tsn} f(t)dt \end{cases}$$

$$\sum_{i=1}^{n} D_{m_i} = D_m \quad (5.14)$$

$$\begin{cases} t_{21} = t_{s1} - (a + t_1) \\ t_{22} = t_{s2} - t_{e1} \\ t_{23} = t_{s3} - t_{e2} \\ \quad . \\ \quad . \\ \quad . \\ t_{2n} = t_{sn} - t_{e(n-1)} \end{cases}$$

$$t_2 = \sum_{i=1}^{n} t_{2i} \quad (5.15)$$

For the blocking time $t_3$:

Let

- $r$ represent the output-input ratio of the Map instances.

It is noted that the processing power of the processor can be represented as $p = f(x)$.

Therefore the buffer filling speed while both the processor is processing and the buffer is spilling simultaneously can be represented by:

$$f(x) \times r - H_w(x) \quad (5.16)$$

Therefore, the first processing interval $t_{21} = t_{s1} - (a + t_1)$ to fill up the buffer can be represented by

$$B_f = \int_{a+t_1}^{ts1} [f(t) \times r - H_w(t)]dt \quad (5.17)$$

The time $\Delta t_1$ of the buffer emptying while the processor is blocked can be represented by

$$\Delta t_1 = t_{e1} - t_{s1} \quad (5.18)$$

$t_{e1}$ and $t_{s1}$ satisfy

$$B_f = \int_{t_{s1}}^{t_{e1}} H_w(t)\ dt \quad (5.19)$$

As the same as the first processing interval, the second processing interval $t_{21} = t_{s2} - t_{e1}$ to fill up the buffer can be represented by

$$B_f = \int_{te1}^{ts2} [f(t) \times r - H_w(t)]dt \quad (5.20)$$

The time $\Delta t_2$ of the buffer emptying while the processor is blocked can be represented by

$$\Delta t_2 = t_{e2} - t_{s2} \quad (5.21)$$

$t_{e2}$ and $t_{s2}$ satisfy

$$B_f = \int_{t_{s2}}^{t_{e2}} H_w(t)\ dt \quad (5.22)$$

Until the $n^{th}$ the first processing interval $t_{2n} = t_{sn} - t_{e(n-1)}$ to fill up the buffer can be represented by

$$B_f = \int_{te(n-1)}^{tsn} [f(t) \times r - H_w(t)]dt \quad (5.23)$$

The time $\Delta t_n$ of the buffer emptying while the processor is blocked can be represented by

$$\Delta t_n = t_{en} - t_{sn} \quad (5.24)$$

Finally the emptying time $t_3$ when the buffer of the *mapper* is filled up can be represented by

$$t_3 = \Delta t_1 + \Delta t_2 + \cdots + \Delta t_n \quad (5.25)$$

For the merging time $t_4$:

Let

- $D_{I_i}$ represent in each processing-spilling step, the intermediate data one *mapper* can generate.
- $N$ represent the number of the merging times.
- $s$ represent the value of the sortfactor.
- $t_{mf}$ represent the merging finishing time.

In the first processing-spilling step, one *mapper* can generate a volume of intermediate data with a size of

$$D_{I_1} = B + \int_{a+t_1}^{t_{s1}} H_w(t)dt \quad (5.26)$$

In the following processing-spilling steps, the intermediate data one *mapper* can generate can be represented by

$$D_{I_i} = B + \int_{t_{e(i-1)}}^{t_{si}} H_w(t)dt \quad (5.27)$$

From the $1^{th}$ to $n^{th}$ intermediate data chunks they satisfy the equation

$$\sum_{i=1}^{n} D_{I_i} = D_{m_i} \times r \quad (5.28)$$

Therefore the number of the merging times $N$ can be represented by

$$N = \left\lfloor \frac{n}{s} \right\rfloor - 1 \quad (5.29)$$

Finally the merging time $t_4$ can be computed by

$$D_m \times r \times N = \int_{t_{en}}^{t_{mf}} H_w(t)dt \quad (5.30)$$

$$t_4 = t_{mf} - t_{en} \quad (5.31)$$

Based on the equations, the relationships related to the data chunks $D_m$ and the overall processing time $T$ are established. Therefore the total time $T_a$ to process data in one processing wave in Hadoop cluster is the maximal time consumed by the number of $k$ Map instances that are involved in the file processing:

$$T_a = \max\{T_1, T_2, T_3, \dots, T_k\} \quad (5.32)$$

According to divisible load theory, to achieve a minimal processing time $T$, it is expected that all the Map instances involved to complete the data processing at the same time:

$$T_1 = T_2 = T_3 \dots = T_k \quad (5.33)$$

However, from the equations above it can be observed that it is difficult to get the solutions of $D_{m_i}$ which can represent $T_i$ in a straight way so that we introduce genetic algorithm to help to achieve the solutions.

### 5.2.3 The Design of GA

Due to the complexity of the equations, it is difficult to achieve the solutions so that genetic algorithm is involved to solve the equations. As the target of the algorithm is to balance the processing time $T_i$ among $k$ Map instances, so a proper set of $D_{m_i}$ should be found out as solutions to satisfy the equation $T_1 = T_2 = T_3 \dots = T_k$. Thus the number of chunks with size of $D_{m_i}$ assigned to number of $k$ Map instances could be regarded as genes to form the chromosome. Figure 5.3 shows a chromosome sample with 6 genes.

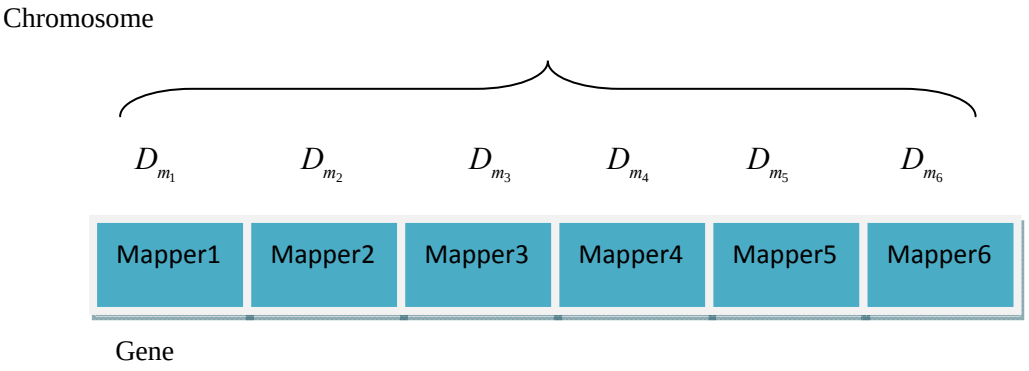


Figure 5.3: A chromosome example.

Thus, a fitness function should be established to evaluate the fitness of the chromosomes. As the processing time $T_i$ of mappers should be close enough, so Mean Square Error (MSE) is employed to assess the fluctuations among $T_i$. Therefore, the fitness function can be defined as follows:

$$f(T) = \sqrt{\sum_{i=1}^{k}(\bar{T} - T_i)^2},\ \bar{T} = \frac{\sum_{i=1}^{k} T_i}{k} \quad (5.34)$$

Where $T_i$ represents the processing time for the $i^{th}$ Map instance.

$k$ represents the number of map instances employed in the Hadoop cluster.

$\bar{T}$ represents the average time in processing of map instances.

In the algorithm the single point crossover is used. However, one issue should be pointed out that just simply crossing the chromosomes may cause one problem. To cross the genes may differentiate the original total volume of data $D_1 = \sum_{m=1}^{k} D_m$. Consider the original total volume of the data is $\sum_{m=1}^{k} D_m$ and the volume of data after crossover is $\sum_{m=1}^{k} d_m$. Then the difference $\Delta D = \left|\sum_{m=1}^{k} D_m - \sum_{m=1}^{k} d_m\right|$ should be polished. In the algorithm $\Delta D$ is divided into $k$ parts randomly. And then these $k$ parts will be randomly added to or removed from the number of $k$ genes accordingly. Thus the total size of processed data in one wave can be guaranteed.

To avoid the local optimum of the genetic algorithm, the standard mutation is also

used in the algorithm. Due to the mutations, the original total volume of data $D_1 = \sum_{m=1}^{k} D_m$ may be changed. Thus consider the original volume of the data is $D_1 = \sum_{m=1}^{k} D_m$ and the volume after mutation is $\sum_{m=1}^{k} d_m$. Then the difference $\Delta D = \left|\sum_{m=1}^{k} D_m - \sum_{m=1}^{k} d_m\right|$ should be polished. In the algorithm $\Delta D$ is divided into $k$ parts randomly. And then these $k$ parts will be randomly added to or removed from the number of $k$ genes accordingly. Thus the total size of processed data in one wave can be guaranteed.

### 5.2.4 The Improvement of the Load Balancing Algorithm

To get the solutions of the equations based on genetic algorithm and algebraic methods, a number of iterations are involved for instance the iterations brought by Newton Tangent Method (NTM). Consider that in one generation of the genetic algorithm, $n_i$ iterations with consumed time $t_i$ for each iteration are involved, thus after a number of $g$ generations with a number of $c$ chromosomes, the overhead of NTM can be roughly represented by $T_{NTM} = \sum_{j=1}^{g} c \times n_{i_j} \times t_{i_j}$. Consider the rest overhead in one generation of the genetic algorithm is $T_i$, therefore the overall overhead of GA could be represented by

$$T = \sum_{j=1}^{g} c \times n_{i_j} \times t_{i_j} + T_i \times g \quad (5.35)$$

From the equation it can be observed that along with increasing the number of generations $g$, the overhead of solving solutions of the load balancing algorithm will be enlarged approximately proportionally. So to control the number of generations can enhance the performance of the load balancing algorithm. However, reducing the number of generations will definitely bring one issue. The accuracy of the solution may be lost which may unbalance the load among *mappers*. Therefore, for gaining a lower overhead with less number of iterations, the genetic algorithm needs an efficient and reliable correctness to make up the loss of accuracy. We reduce the

number of iterations to a smaller value $g_1$ . Thus the solutions of $D_m\{D_1, D_2, D_3, \dots, D_k\}$ for the optimized scheduler contain unbalanced issue which will cause the processing time $T_m\{T_1, T_2, T_3, \dots, T_k\}$ unbalanced. It is known that if $\{T_1 \neq T_2 \neq T_3 \dots \neq T_k\}$, the *mapper* which has the longest processing time will deteriorate the performance of the load balancing algorithm. Especially if a large number of waves have been involved in the processing, the error in each wave will enlarge the overall overhead of the job processing time. If the overhead of computing the scheduler increases to a quite considerable level, the performance of the algorithm will be reduced. Thus after the scheduler is computed with errors, a strategy which can do a compensation is in the following way.

Select the slowest processing time $T_s$ of a *mapper*

$$T_s = \max\{T_1, T_2, T_3, \dots, T_k\} \quad (5.36)$$

The errors $\Delta t$ between the slowest *mapper* and the $i^{th}$ *mapper* can be represented by

$$\Delta t_i = T_i - T_s \quad (5.37)$$

As the genetic algorithm with less number of generations can still find a solution which is close to the optimized solution, so the errors $\Delta t_i$ are normally not large. Thus it can be considered that if the faster *mappers* finish processing, they can take amount of data to be processed in time $\Delta t_i$. Since $\Delta t_i$ is not large so that the buffer is hardly filled up, which means no complex IO operations involved. Finally the compensation can be done. The idle faster *mappers* can start processing smaller amounts of data while the slowest *mapper* is still in processing.

Let

- $a$ represent the finishing time of the $i^{th}$ *mapper*.
- $\tau$ represent a given threshold so that if $\Delta t_i$ is smaller than $\tau$, the algorithm does not compute the following equation.
- $D_e$ represent the volume of data will be processed in $\Delta t_i$.

$$\Delta t_i = T_i - T_s \geq \tau \quad (5.38)$$

$$D_e = \int_a^{a+\Delta t_i} f(t)dt \quad (5.39)$$

The algorithm supports balance loads among *mappers* with different loads of processors. If the loads are not computable, they can be known by statistical computing based on historical data of loads.

## 5.3 Simulation

Due to the limitations of the experimental environment, it is difficult to implement the load balancing algorithm. So HSim is involved to help to evaluate the algorithm, which can facilitate to observe the load balancing behaviors of Hadoop framework.

### 5.3.1 The Dynamic Factors in HSim

To evaluate the dynamic load balancing algorithm, HSim needs to be designed with abilities to supply simulated dynamic Hadoop computing environment. In paper [77] they claim that the most influential factors of a dynamic computing environment is the processor and hard disk. As the accuracies on simulating Hadoop working mechanisms have been validated in chapter 3, therefore based on these accuracies the CPU model has been modified, which can supply different kinds of loads. The details of the load are presented later in this chapter. The hard disk model in HSim is designed to be dynamically changed as time passes. In [86], R. Sharykin reports that a hard disk can be modeled following an exponential distribution due to the loss of speed of a traditional mechanical hard disk. However, the simple exponential distribution cannot describe the performance of a hard disk accurately enough so that in HSim, based on experimental hard disk tests three extra parameters have been involved: max speed $x_{max}$, min speed $x_{min}$ and speed reducing ratio $r$ (The speeds and ratio include both reading and writing speeds of a standard mechanical Winchester architecture based hard disk). Therefore, when the speed of hard disk is reduced from $x_{max}$ to $x_{min}$ while the speed is at $x_{min}$, $r$ is 0. Thus, the equation can be represented by

$$r(x) = r - sx, r(x_{min}) = 0 \quad (5.40)$$

Where $s$ is the relational reducing coefficient between instantaneous speed $x$ and $r$. Therefore $s$ can be represented by

$$s = \frac{-r}{x_{min}} \quad (5.41)$$

Thus

$$r(x) = r\left(1 + \frac{x}{x_{min}}\right) \quad (5.42)$$

Therefore a differential equation can be established to describe the relationships among $x_{max}$, $x_{min}$ and $r$.

$$\frac{dx}{dt} = r\left(1 + \frac{x}{x_{min}}\right)x, x(0) = x_{max} \quad (5.43)$$

Solve the equation the instantaneous speed $H(t)$ related to current time $t$ can be achieved.

$$H(t) = \frac{x_{min}.x_{max}}{(x_{min} - x_{max})e^{-rt} + x_{max}} \quad (5.44)$$

Thus $H(t)$ in HSim is a dynamic factor which can supply a dynamic IO environment.

### 5.3.2 Heterogeneity

HSim supports to simulate heterogeneous computing environment. To describe the differences among nodes, the concept of heterogeneity is introduced. Heterogeneity can define how different the machines in the cluster are and give a quantified indicator to make the differences clear. The most important difference among nodes is the processing speed of processor. Though the other hardware would affect the performances of the execution, the speed of processor is the core factor which decides the overall performance significantly. So the level of heterogeneity of our cluster is defined based on the processing speeds of the processors. The total processing speed of the cluster is kept fixed, which means the value of the total processing speed is a constant value. Then according to different heterogeneities, the speeds of the processors will have different values. The heterogeneity can be represented by

Let

- $P$ represent the total processing speed of the cluster.
- $p_i$ represent the processing speed of the $i^{th}$ processor.

- $\bar{p}$ represent the average processing speed of the cluster.
- $k$ represent the number of processor employed in the cluster.

$$Heterogeneity = \sqrt{\sum_{i=1}^{k}(\bar{p} - p_i)^2} \text{ where } \bar{p} = \frac{\sum_{i=1}^{k} p_i}{k}, \sum_{i=1}^{k} p_i = P \quad (5.45)$$

The equation could represent the varieties of the machines of the cluster. If the value of $Heterogeneity$ is greater than zero, it means the cluster is heterogeneous. The greater the value is, the larger the differences are. If the value of $Heterogeneity$ is zero, it means the cluster is a homogeneous one. However, due to the multi dynamic elements in the computing environment $Heterogeneity$ can not exactly describe the computing difference. The initial processing speed of a processor has been employed to generate the $Heterogeneity$. Therefore the $Heterogeneity$ can show an overall heterogeneous trend but at some points some exceptions may exist.

## 5.4 Simulation Results

### 5.4.1 Exponential Distribution of the Load of Processors

In this simulation the load of the processors follows a distribution which is similar to exponential distribution. Due to the long processing time for the simulated job, the normal exponential distribution cannot satisfy the simulator due to its fast attenuating speed no matter what value of $\lambda$ is. Therefore two parameters are added to the exponential distribution to control its attenuating speed. The newly formed distribution can be represented by

$$f(x) = a \cdot \lambda e^{-\frac{\lambda x}{b}} \quad (5.46)$$

Table 5.2: The simulated environment.

| Simulation environment | |
|---|---|
| Simulated algorithm: | MR-LSI |
| Size of data: | Simulation 1: 40GB<br>Simulation 2: From 10GB to 100GB |

| | |
|---|---|
| Load balancing strategies: | 1. Without load balancing<br>2. Load balancing with dynamic window size<br>3. Computing ratio<br>4. Load balancing with fixed window size |
| Number of simulated nodes: | 20 |
| Number of processors in each node: | 1 |
| Number of cores in each processor: | 2 |
| The processing speeds of processors: | Depending on heterogeneities |
| Heterogeneities: | from 0 to 2.28 |
| Number of hard disk in each node: | 1 |
| Max writing speed of hard disk: | 80MB/s |
| Min Writing speed of hard disk: | 40MB/s |
| Number of Map and Reduce instances: | Each node employs 2 *map* instances. The cluster employs 1 *reduce* instance. |
| Sort factor: | 100 |

Figure 5.4 shows the results of different load balancing strategies in processing 40GB data using the distribution discussed above.

In Figure 5.4 it can be observed that when the heterogeneity is smaller, the scheduler without any load balancing strategy outperforms the other schedulers. However, when the heterogeneity increases larger, it can be seen that both load balancing with dynamic window size and fixed window size perform better. Especially dynamic window size outperforms fixed window size because the fixed window size may cause *mappers* idle to wait for the solution of the scheduler. The computing ratio strategy performs worse due to the affections of the reducing time.

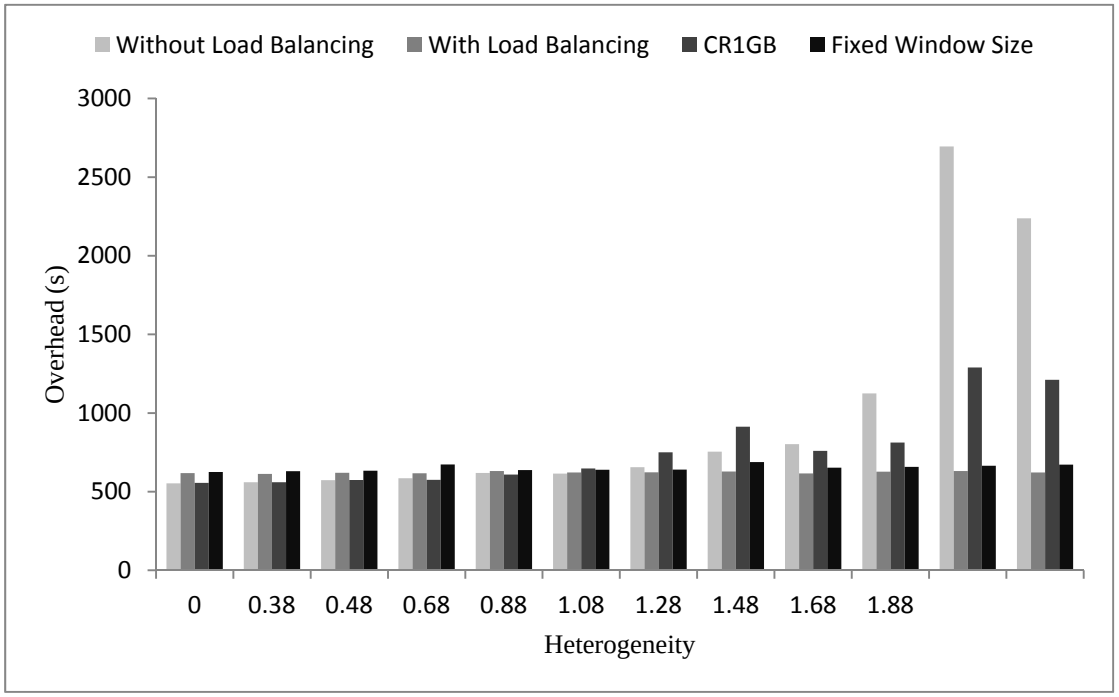


Figure 5.4: Comparison of different load balancing strategies with different heterogeneities.

In Figure 5.4 the window size of one second is used for fixed window size strategy. According to the results of the tests, the smaller window can give better performance. To study the impacts of the window size, the performance of the simulated Hadoop cluster in terms of overhead is tested with window size from one second to one hundred seconds. Figure 5.5 shows the results. From the figure it can be observed that along with the window size increases, the overhead becomes larger.

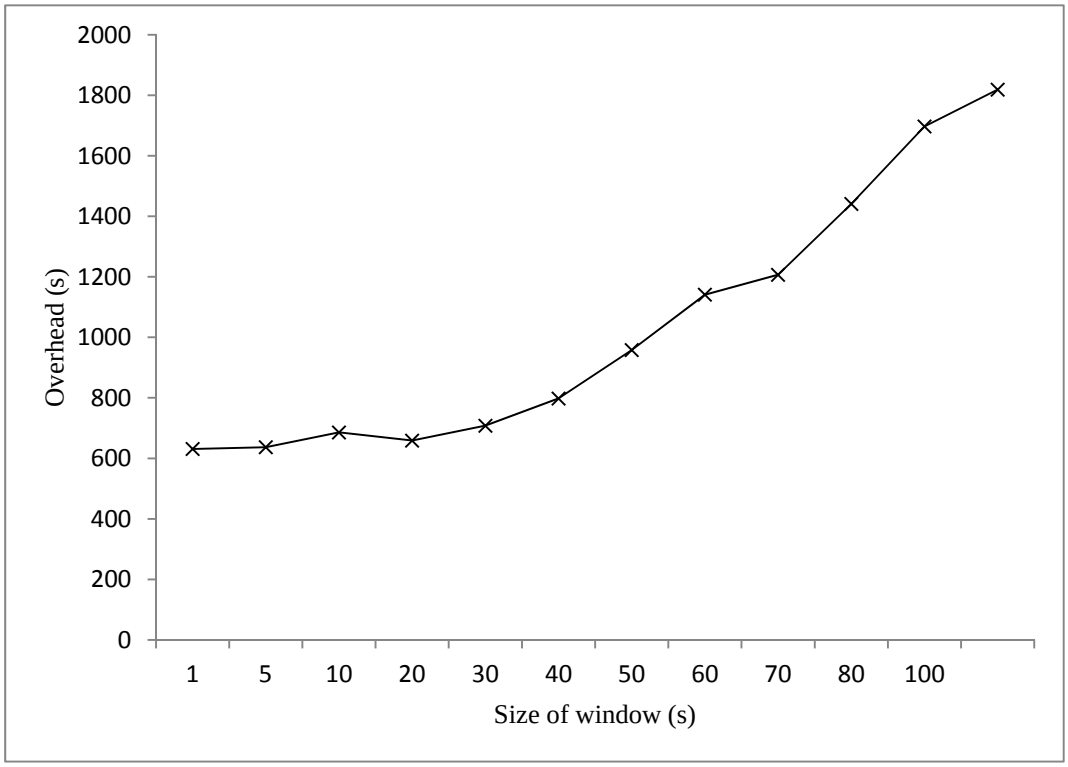


Figure 5.5: The impacts of different window sizes.

For evaluating the load balancing algorithm with different data size, the data size from 10GB to 100GB with heterogeneity 2.28 has been simulated to observe the performances of the algorithm. Figure 5.6 shows the results.

In Figure 5.6 it can be observed that from the data size of 10GB to 100GB, the processing times of the scheduler without load balancing are nearly the same. The reason is the slowest *mapper* becomes the bottleneck. The processing time for the *mapper* is extremely longer which affects the performances hugely. In another simulation with data size of 120GB, the processing time increases to 4489 seconds, which means the slowest *mapper* starts its second wave which causes the other approximate 2240 seconds overhead. From the figure it also can be seen that the load balancing with dynamic window size outperforms the other strategies for any data size. Due to the lack of measuring the computing capability of MR-LSI algorithm, computing ratio gives the worst performance.

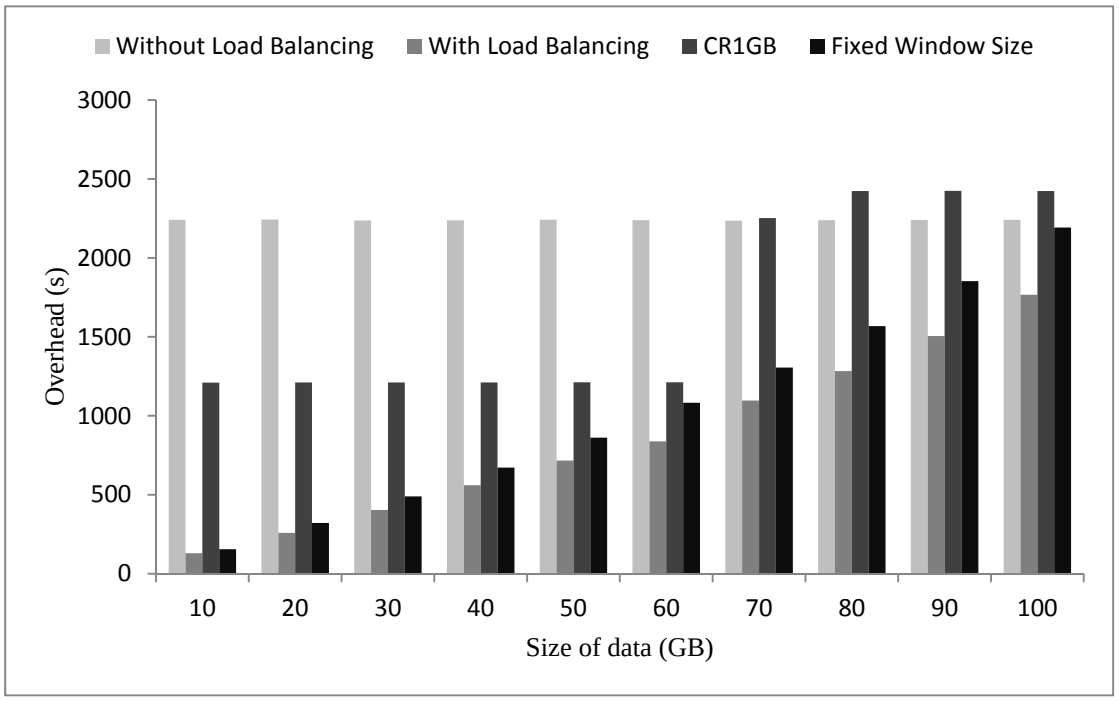


Figure 5.6: Comparison of different load balancing strategies with different data sizes.

As the algorithm is designed with dynamic window sizes to implement the load balancing algorithm, so the size of the window should be various. Figure 5.7 shows the window sizes in different waves for processing 40GB data with heterogeneity 2.28. It also gives that due to the attenuation of the processing power of the processor cross the cluster, the sizes of the window become longer.

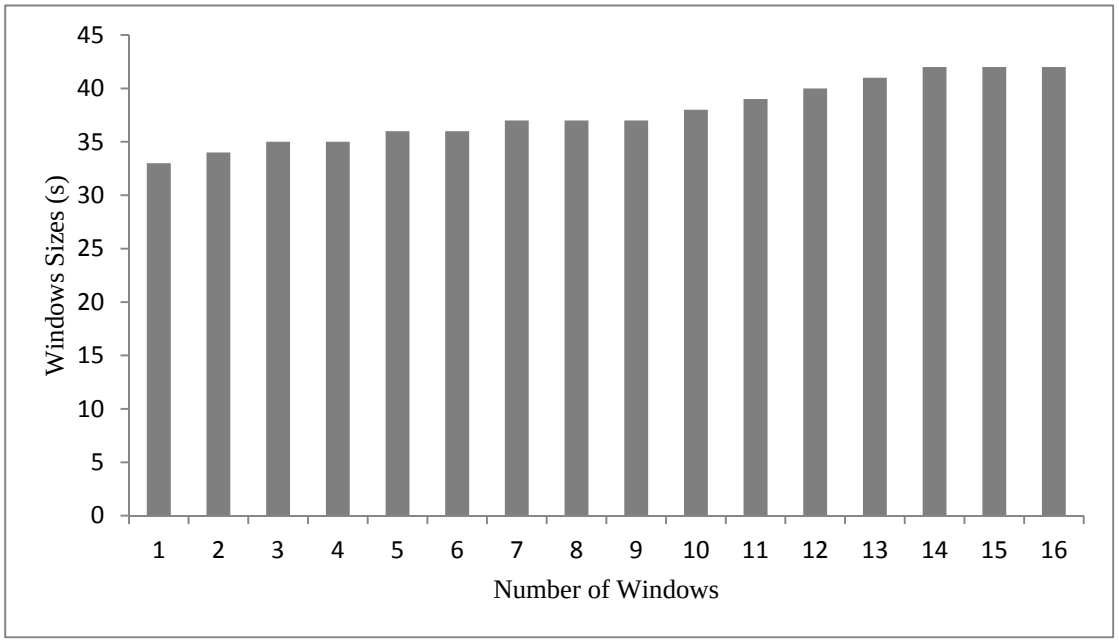


Figure 5.7: The dynamic window sizes.

The load balancing scheme builds on a genetic algorithm whose convergence affects the efficiency of the tested algorithm MR-LSI. To analyze the convergence of the genetic algorithm, the number of generations is varied and the overhead of MR-LSI is measured in the simulated Hadoop environment. Figure 5.8 shows that MR-LSI reaches a stable performance when the number of generations in the genetic algorithm reaches 300. For the simpler load of processors, the solution of 300 generations can supply enough accuracy to perform the load balancing algorithm.

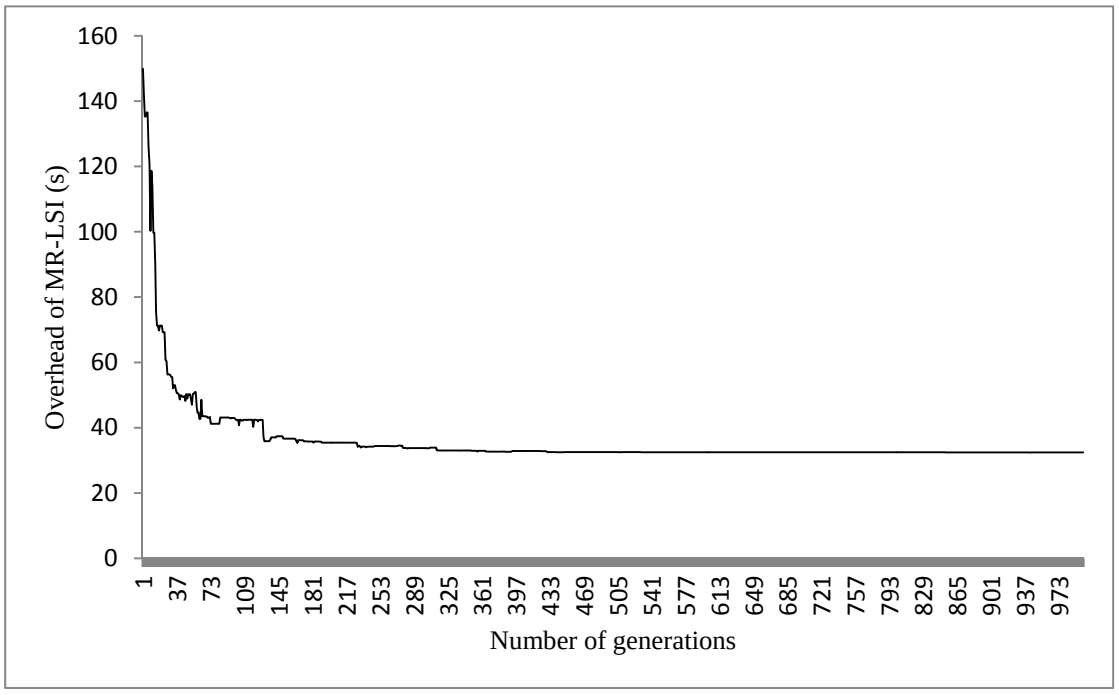


Figure 5.8: Convergence of the algorithm.

The load balancing scheme also produces some overhead during execution. Figure 5.9 shows an increased overhead of the load balancing scheme when the number of *mappers* increases across the cluster. However the load balancing overhead is small compared with the overall processing time of *mappers*.

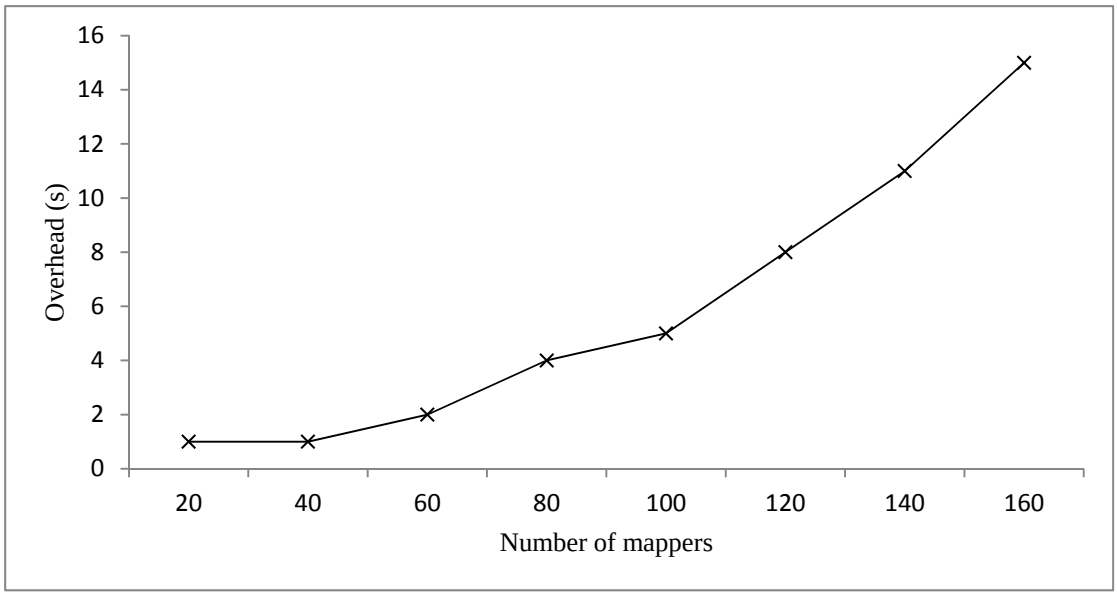


Figure 5.9: The overhead of load balancing with dynamic window size with increasing number of mappers.

### 5.4.2 Cosine Distribution of the Load of Processors

In the previous a distribution which is based on exponential distribution has been used. However the distribution is simple which makes the load of the cluster simply keeping reducing. Thus complex loads for processors should be designed to vibrate the load of the cluster to make a complex computing environment. Based on the fluctuations of the load, the performance of the load balancing algorithm can be evaluated strictly. To design a load which can keep vibrating, the cosine function is employed to build up our load function for each processor. It can be represented by

$$f(x) = acosbx + c \quad (5.47)$$

Thus the load of the cluster $p(t) = \sum_{i=1}^{n} f_i(t)$ is not a simple monotonically increasing or decreasing function. The dynamic environment can be more complex.

Table 5.3: The simulated environment.

| **Simulation environment** | |
|---|---|
| Simulated algorithm | MR-LSI |
| Size of data | Simulation 1: 40GB<br>Simulation 2: From 10GB to 100GB |
| Load balancing strategies | 1. Without load balancing<br>2. Load balancing with dynamic window size |

| | |
|---|---|
| | 3. Computing ratio<br>4. Load balancing with fixed window size |
| Number of simulated nodes | 20 |
| Number of processors in each node | 1 |
| Number of cores in each processor | 2 |
| The processing speeds of processors | Depending on heterogeneities |
| Heterogeneities | from 0 to 2.28 |
| Number of hard disk in each node | 1 |
| Max writing speed of Hard disk | 80MB/s |
| Min Writing speed of Hard disk | 40MB/s |
| Number of Map and Reduce instances | Each node employs 2 *map* instances. The cluster employs 1 *reduce* instance. |
| Sort factor: | 100 |

Figure 5.10 shows the results of different load balancing strategies in processing 40GB data in the complex dynamic computing environment.

From Figure 5.10 it can be observed that the processing overheads without load balancing of different heterogeneities are highly various. The reason is as discussed above that due to the complex dynamic environment, our heterogeneous equation can only show a trend of the heterogeneity but cannot exactly describe the heterogeneity in detail. The figure indicates that when the heterogeneity is lower, the strategy without load balancing performs better. However, when the heterogeneity is larger, the other three load balancing strategies outperform the strategy without load balancing. It is also quite clearly that with larger heterogeneities, the performances of load balancing with dynamic window size are the best.

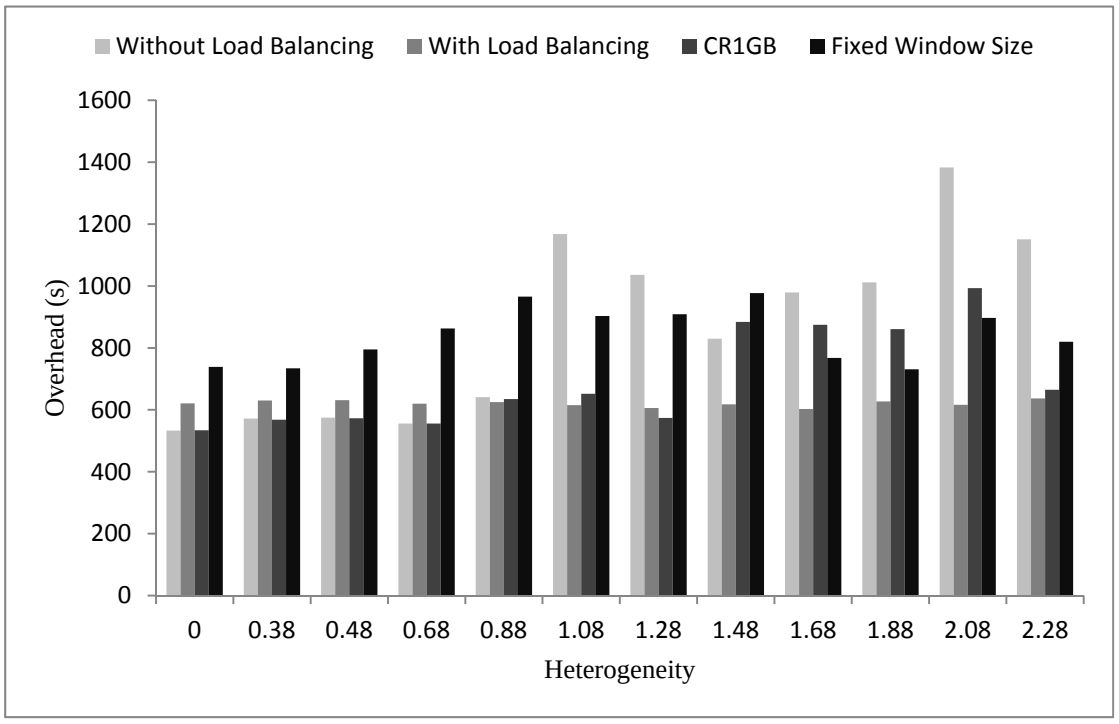


Figure 5.10: Comparison of different load balancing strategies with different heterogeneities.

In Figure 5.10 the window size of one second is used for fixed window size strategy. According to the results of the tests, the smaller window can give better performance. To study the impacts of the window size, the performance of the simulated Hadoop cluster is tested in terms of overhead with window size from one second to one hundred second. Figure 5.11 shows the results. From the figure it can be observed that along with the window size increases, the overhead becomes larger.

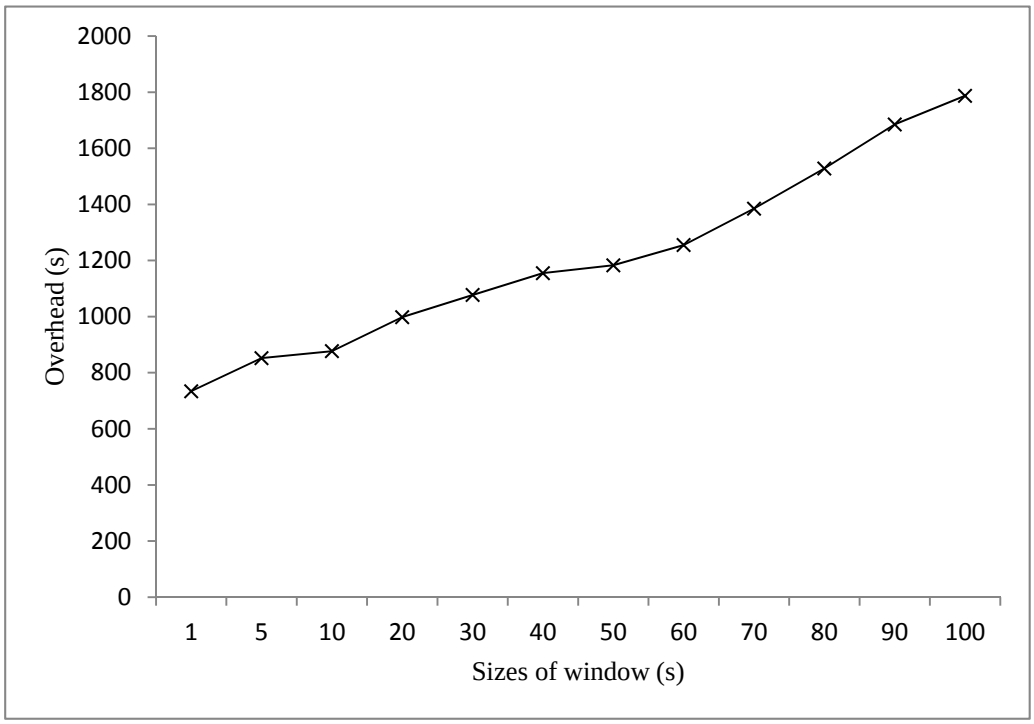


Figure 5.11: The impacts of different window sizes.

For evaluating the load balancing algorithm with different data sizes, the data size from 10GB to 100GB is simulated to feed to the cluster with heterogeneity 2.28. Figure 5.12 shows the results.

In Figure 5.12, it shows that the three load balancing strategies outperform the scheduler without load balancing at the most sizes of data. However, it can be observed that at certain point (like 70GB), the slowest *mapper* finish processing its own data without assigning new data chunk occasionally, thus the overall performance of the cluster is not affected by the slowest *mapper* greatly. The figure also shows that the load balancing with dynamic window size gives the best performance while the performances of the other two strategies are various.

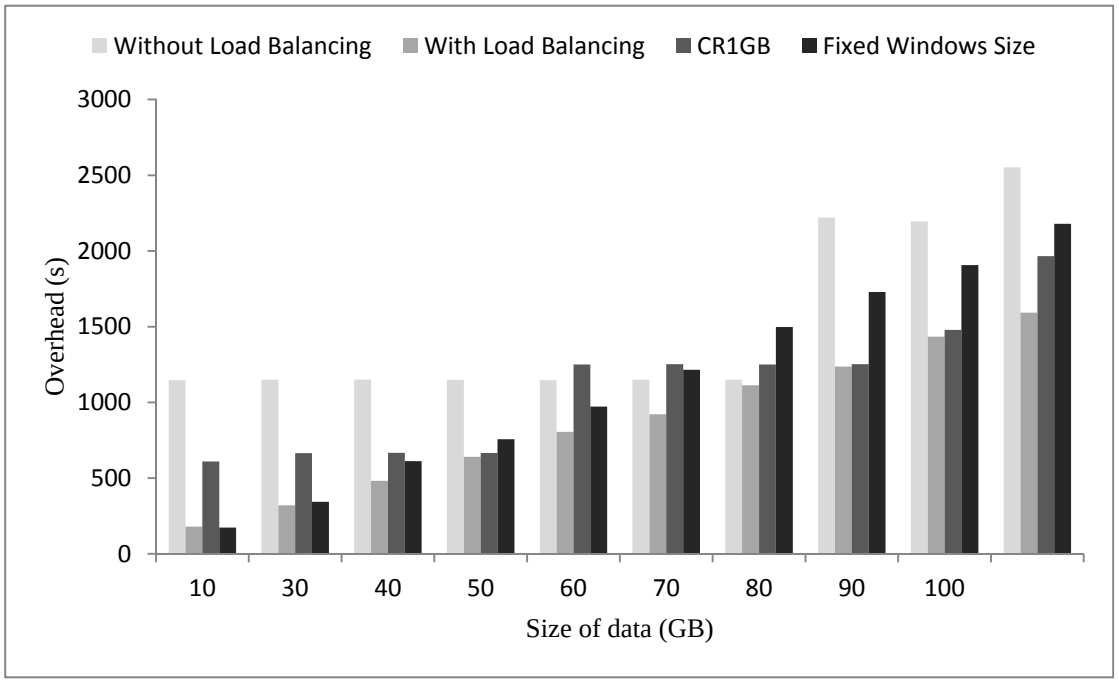


Figure 5.12: Comparison of different load balancing strategies with different data sizes.

Figure 5.13 indicates the dynamic window sizes during the processing. It can be observed that the windows sizes dynamically changed according to the changing of dynamic environment.

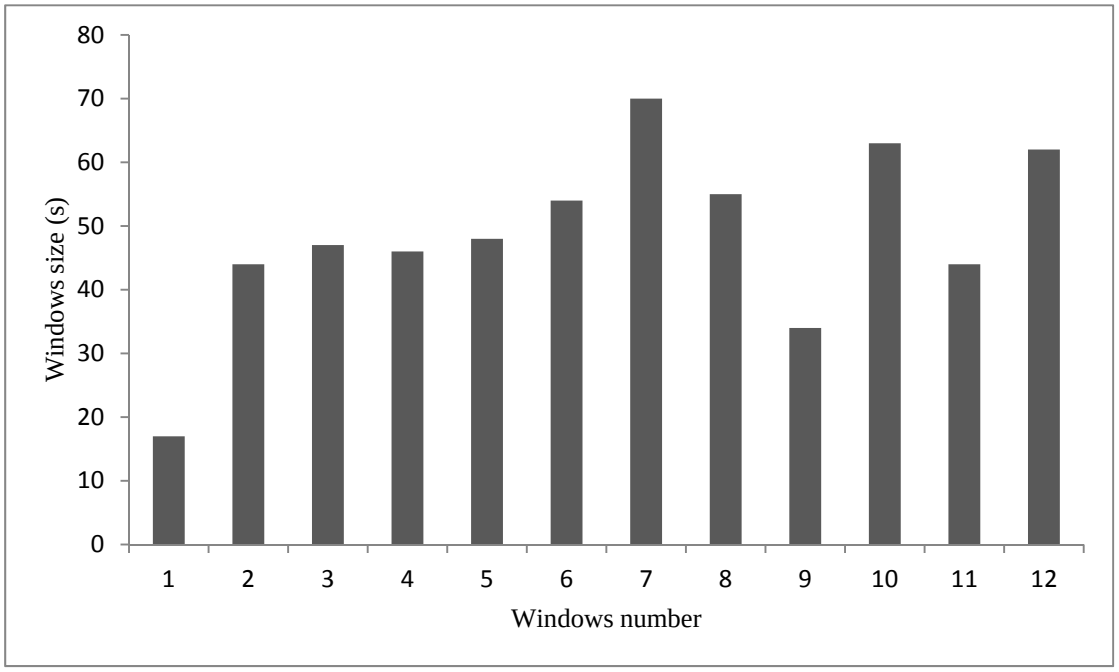


Figure 5.13: The dynamic window sizes.

The load balancing scheme builds on a genetic algorithm whose convergence affects

the efficiency of the tested algorithm MR-LSI. To analyze the convergence of the genetic algorithm, the number of generations is generated and the overhead of MR-LSI is measured in the simulated Hadoop environment. Figure 5.14 shows that MR-LSI reaches a stable performance when the number of generations in the genetic algorithm reaches 300. However it needs to be pointed out that even the solution with 300 generations cannot give enough accuracy for the complex loads of processors. Therefore more generations are need for example 800 generations as shown in the figure. Thus the overhead of the genetic algorithm is large. So the improved dynamic load balancing algorithm needs less number of generations, which generates less overhead.

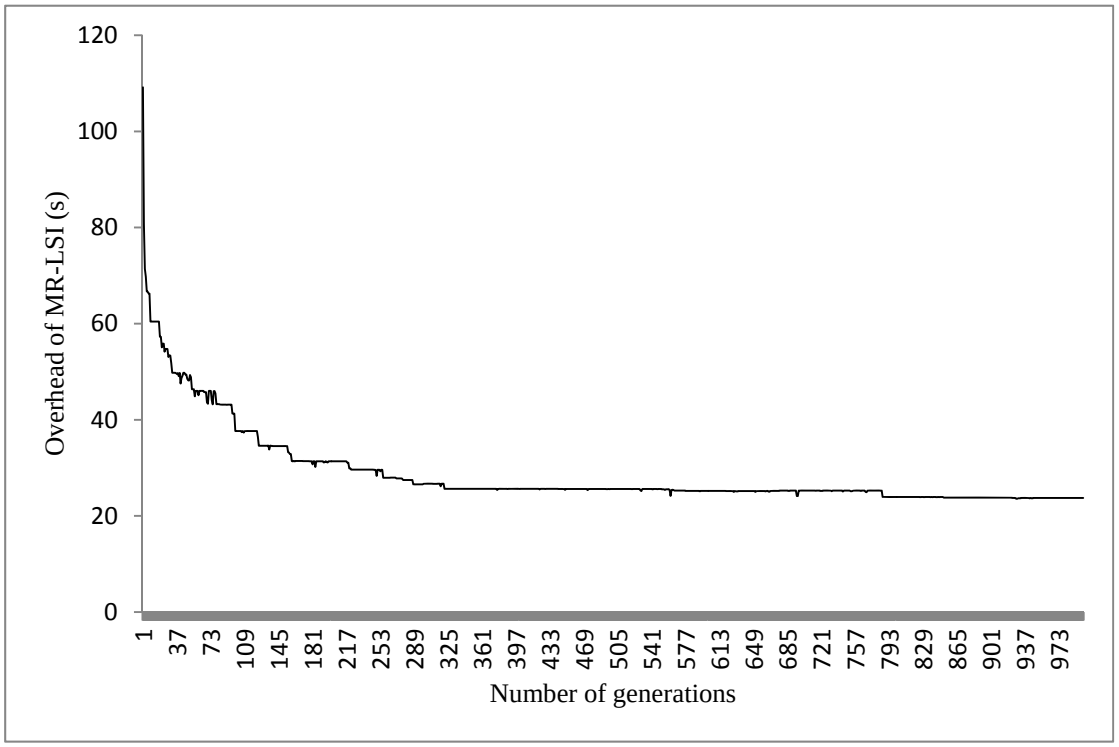


Figure 5.14: Convergence of the algorithm.

Figure 5.15 shows an increased overhead of the load balancing scheme when the number of *mappers* increases across the cluster. However the load balancing overhead is small compared with the overall processing time of *mappers*.

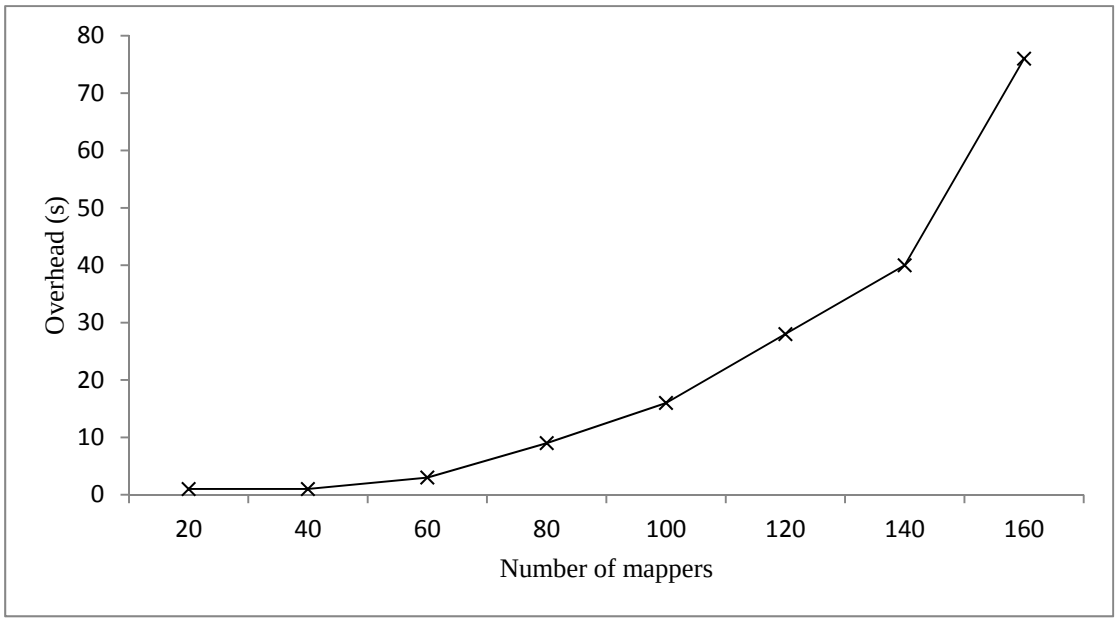


Figure 5.15: The overhead of load balancing with dynamic window size with increasing number of mappers.

## 5.5 Summary

This chapter proposes a centralized, unsupervised, dynamic window-sized load balancing algorithm for Hadoop framework which is based on MapReduce model. The performance of the algorithm has been evaluated by the Hadoop simulator HSim. The results show that the algorithm has remarkable efficiency on solving the load issue when the heterogeneity increases to a certain level. Especially for the improved algorithm, it can solve the issue even with a lower heterogeneity. The results also show that either with small volume of data or great volume of data, the algorithm can still perform well. The dynamic window can control the time when the balancing algorithm starts to execute so that there is no need for people to determine and intervene the computation to get the solution of the optimized scheduler.

# Chapter 6
# Conclusion and Future Work

## 6.1 Conclusion

This thesis proposes a resource aware distributed LSI algorithm based on Hadoop framework which is an implementation of MapReduce model. The typical LSI suffers a considerable issue that it is extremely computationally intensive. It is also reported that when the scale of a T-D matrix is large, the performance of LSI is deteriorated. Though several works have been done on solving the scalability issue of the algorithm, the proposed approaches are simply combined the clustering algorithm *k-means* with LSI. From the experimental results it can be observed that the scalability issue has been solved to some extent. However, a new issue has been introduced, which is the overhead of the involved clustering algorithm *k-means*. The experimental results indicate that when the size of the document corpus increases to a certain level, the overhead of the pre-clustering becomes highly considerable due to both the computing complexity and the limitations of the computing resources. MR-LSI successfully solved such an issue based on distribute the *k-means* algorithm in a Hadoop cluster. The experimental results show that with a proper number of centroids, the recall and precision of the MR-LSI algorithm are highly close to the typical standard LSI. In terms of algorithm executing time, due to the system overhead of the Hadoop framework, it is quite high with processing small size of document collections. The work also shows, when the size of the document collection increases to a certain critical boundary, the overhead of the framework ca be overcome, which means MR-LSI outperforms the other algorithms in terms of overhead with maintaining the similar recall and precision levels.

Firstly the MR-LSI algorithm has been evaluated in a small Hadoop cluster which contains a number of four nodes. However, the small environment cannot help to

study the performance of MR-LSI with mass computing resources. Thus to study the scalability of the MR-LSI algorithm, a simulator has been developed to perform a number of simulations. Though currently there is one published simulator MRPerf available and it claims that it can simulate the Hadoop framework accurately. However, according to the results of evaluating a number of MapReduce applications, MRPerf cannot give satisfied performance. The simulator proposed in the thesis HSim modeled the parameters of Hadoop framework from several aspects including node parameters, cluster parameters and Hadoop system clusters. These parameters can help HSim to create a simulated Hadoop cluster with detailed specifications which are mainly employed in a real cluster. The validations show that the performance of HSim is quite close to that of the practical experimental cluster. And also HSim outperforms MRPerf of which the performance is highly different from that of experimental environment.

Therefore based on HSim, a simulated Hadoop cluster with the number of 25 nodes up to 250 nodes has been created. Thus a number from 100 to 1000 *mappers* are involved to evaluate the scalability of MR-LSI. The evaluations indicate that generally along with the number of *mappers* increased, the performance of MR-LSI enhanced in terms of overhead. However, due to the wave mechanism in the Hadoop framework, at the points of certain number of *mappers*, simply keep increasing the number of *mappers* cannot gain the performance enhancement. A number of tests have also been done to study the impacts brought by tuning parameters on the MR-LSI algorithm. The results indicate that the performance of the algorithm can be significantly affected by the different configurations of the cluster.

It is recognized that in the current version of Hadoop framework, the load balancing strategy is quite weak. Only two types of the strategies FIFO and fair scheduler have been supported yet. The two types of the schedulers aim to balance the resources among jobs of different users. However, as the framework supports heterogeneous nodes, only balancing the resources among users may not get the satisfied optimized

performance due to the unbalanced load among *mappers* employed by different nodes. Thus a static load balancing strategy has been proposed firstly by this work. In the strategy, the working mechanism of *mapper* is modeled from four aspects which include copying time, processing time, spilling time and merging time. The copying time represents the time of copying a data chunk to local hard disk of the *mapper*. The processing time represents the actual data processing by a processor. The spilling time represents the time of emptying a buffer while the buffer is filled up. The merging time represents the time of merging intermediate data into a whole chunk which will be ultimately send to *reducer(s)*. By modeling these overheads of working mechanism of a *mapper*, the data sizes that are initially sent to mappers involved in the processing can be calculated. Therefore to balance the load among *mappers*, according to a certain scheduler, if the overall overhead which is the sum of the above four sub-overheads of each *mapper* could be close enough to those of the other *mappers*. And then the scheduler can be regarded as a best solution. Instead of stiffly and directly measuring a solution with a complex way, the Means Square Error (MSE) has been used. MSE can represent how different a series of data is. Therefore, by calculating the MSE of all *mappers'* overhead, the differences among them can be quantitatively measured. Aiming to get the optimized solutions from the combinations of a number of complex equations, the genetic algorithm has been involved. The genes are the volume of data to be allocated to *mappers* while the chromosomes are the schedulers and the fitness is using MSE. Within a number of generations, an optimized scheduler can be found. In a static environment, as long as the scheduler has been worked out, the *mappers* can use the scheduler in the whole data processing duration until the job is finished. The evaluations of the load balancing algorithm have been done in a simulated cluster with different heterogeneities. The concept of heterogeneity is involved to measure the level of differences among nodes employed in the cluster. The evaluated results show that:

- The load balancing algorithm significantly enhances the performance of the cluster when the heterogeneity increasing to a certain level. When the levels of heterogeneity are lower, due to the overhead of the load balancing algorithm

itself, it cannot outperform the scheduler without load balancing in terms of overhead. However, with large levels of heterogeneity, the algorithm can be three times faster compared to the scheduler without load balancing strategy.

- The load balancing algorithm is suitable for jobs with different sizes. In the simulation result, it indicates that the when size of data increases from 10GB to 100GB, the algorithm can give stable performance with varying sizes of data.
- Though along with the increasing of number of *mappers* and size of data, the overhead of the load balancing algorithm keeps increasing, compared to the enhancement gained by the algorithm, the impact brought by the overhead is negligible.

However, frequently a cluster is not simply static but dynamic. There are lots of factors affects the computing capacity of a cluster dynamically along with the time passes. To balance the load among *mappers* in a dynamic computing environment, a dynamic load balancing strategy has been proposed by this work. The strategy is consisted by two parts.

1. A data selection solution has been given to decide the volume of data for each processing wave. The target is trying to use the higher computing capacity time interval to process the data. The algorithm will be executed in next wave again to correct the error caused by the IO operations. The data selection also results in dynamic window sizes in launching the load balancing algorithm.
2. The copying time, processing time, spilling time and merging time have been modeled in the dynamic environment. Based on the equations of these four times, the overall overhead of a *mapper* can be represented. Finally the relationships between sizes of data and the allocated *mappers* can be established.

Due to the complexity of the equations to get the optimized scheduler, genetic algorithm has been involved which the fitness function is based on calculating the MSE of the overhead of *mappers*. In the previous researches, a number of researchers

claimed that with a less number of generations the GA algorithm can gain the optimized solution. However, based on the experimental results, it is not suitable for the Hadoop framework. Therefore, to reduce the overhead of the genetic algorithm, combining with the characteristics of the Hadoop framework, an improvement has been done for the genetic algorithm, which can significantly reduce the number of generations. The simulator HSim also offers a way based on two features to create a dynamic environment. The dynamic features include speed of hard disk and load of processor. The dynamic factor of hard disk can create dynamic IO environments for the cluster. Similarly the dynamic factor of the processor load can create dynamic computing capacity of the cluster. This work presents two different kinds of processor loads of which one is simple and the other one is complex. Thus, based on different heterogeneities, a number of evaluations have been done. Compared to the performances of computing ratio strategy and fixed window size strategy, the dynamic load balancing algorithm achieves significant enhancement when the level of heterogeneity is larger than a certain value.

## 6.2 Future Work

The work presented in this thesis opens a new way to build up a resource aware distributed LSI algorithm for scalable information retrieval. Though based on the experimental and simulation results the algorithm shows satisfied performance, it is clear that still a variety of opportunities exist, for example:

- Determining the best value of rank *k* that is used in SVD can be investigated further to gain the most efficient computation.
- Determining the best value of centroids *k* which is used in k-means can be considered further to get the best clustering results.
- Evaluating the MR-LSI algorithm in a large Hadoop cluster such as Amazon Elastic Compute Cloud (Amazon EC2).
- Though the experimental code of combiner in HSim shows certain level of accuracy, it can be improved further by using a better mathematical model.

- The load balancing strategies are implemented based on the simulator HSim. They may have a chance to be added in the practical Hadoop code to gain better performance in a real Hadoop cluster.

# References

[1] Aarnio, T. (2009). Parallel Data Processing with Mapreduce. *TKK T-110.5190, Seminar on Internetworking,* [Online] Available: http://www.cse.tkk.fi/en/publications/B/5/papers/Aarnio_final.pdf.

[2] Apache Hadoop! Available at: http://hadoop.apache.org/ [Accessed November 2, 2009].

[3] Bassu, D., and Behrens, C. (2003). Distributed LSI: Scalable Concept-based Information Retrieval with High Semantic Resolution. *In Proceedings of Text Mining 2003, a workshop held in conjunction with the Third SIAM Int'l Conference on Data Mining.*

[4] Berry, M. W. (1992). Large scale singular value computations. *International Journal of Supercomputer Applications and High Performance Computing, 6, 13–49.*

[5] Dean, J., and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *In Proc. of OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA*.

[6] Deerwester, S., Dumais, S. T., and Furnas, G. (1990). Indexing By Latent Semantic Analysis. *Journal of the American Society For Information Science, 41, 391-407.*

[7] Ding C. (1999). A Similarity-based Probability Model for Latent Semantic Indexing. *In Proc. of 22nd ACM SIGIR Conference, 59-65.*

[8] Du, L., Jin, H., de Vel, O. Y., and Liu, N. (2008). A Latent Semantic Indexing and WordNet based Information Retrieval Model for Digital Forensics. *In Proc. of Intelligence and Security Informatics, 2008. ISI 2008, IEEE International Conference, Taipei, 70-75.*

[9] Dumais, S. (1993). LSI meets TREC: A Status Report. *In D. Harman (ed.), The First Text Retrieval Conference (TREC1), NIST Special Publication 500-207, 137-152.*

[10] Dumais, S. (1994). Latent Semantic Indexing (LSI) and TREC-2. *In D.Harman (ed.), The Second Text Retrieval Conference (TREC2), NIST Special Publication 500-215, 105-116.*

[11] Dumais S. (1995). Using LSI for Information Filtering: TREC-3 experiments. *In D. Harman (ed.), The Third Text Retrieval Conference (TREC3), NIST Special*

*Publication, 219-230.*

[12] Foltz, P., and Dumais, S. (1992). Personalized Information Delivery: An Analysis of Information Filtering Methods. *Communications of the ACM, 35, 51-60.*

[13] Gao, J., and Zhang, J. (2003). Sparsification strategies in latent semantic indexing. *In Proceedings of the 2003 Text Mining Workshop, San Francisco, CA, 93–103*

[14] Gao, J., and Zhang, J. (2005). Clustered SVD strategies in latent semantic indexing. *Information Processing and Management, 41, 1051–1063.*

[15] He, B., Fang, W., Luo, Q., Govindaraju, N. K., and Wang, T. (2008). Mars: a MapReduce framework on graphics processors. *In PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques, 260–269.*

[16] Hotho, A., Maedche, A., and Staab, S. (2001). Text clustering based on good aggregations. *In Proc. of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society, San Jose, CA, 607-608.*

[17] Husbands, P., Simon, H., and Ding, C. (2001). On the use of singular value decomposition for text retrieval. *In Computational Information Retrieval, Philadelphia, PA, 45–156.*

[18] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys, 31, 264–323.*

[19] Jiménez, D., Ferretti, E., Vidal, V., Rosso1, P., and Enguix, C. F. (2003). The Influence of Semantics in IR using LSI and K-means Clustering Techniques. *In Proceedings of the 1st international symposium on Information and communication technologies.*

[20] Karypis, M. S. G., and Kumar, V. (2000). A Comparison of Document Clustering Techniques. *Technical Report 00-034, Department of Computer Science and Engineering, University of Minnesota.*

[21] Koller, D., and Sahami, M. (1997). Hierarchically classifying documents using very few words. *In Proceedings of the Fourteenth International Conference on Machine Learning.*

[22] Kumar, C. A., and Srinivas, S. (2006). Latent Semantic Indexing Using Eigenvalue Analysis for Efficient Information RetrIeval. *International Journal of Applied Mathematics and Computer Science, 16, 551–558.*

[23] Lämmel, R. (2007). Google's MapReduce programming model — Revisited. *Sci. Comput. Program. 68, 208-237.*

[24] Majavu, W., and van Zyl, T. (2008). Classification of web resident sensor resources using Latent Semantic Indexing and Ontologies. *Proceedings of the IEEE International Conference on Man, Systems and Cybernetics.*

[25] Oksa, G., Becka, M., and Vajtersic, M. (2002). Parallel SVD Computation in Updating Problems of Latent Semantic Indexing. *In Proceedings of ALGORITMY 2002 Conference on Scientific Computing, 113-120.*

[26] Park, H., and Elden, L. (2003). Matrix rank reduction for data analysis and feature extraction. *Tech. Rep., Dept. Computer Science and Engineering, University of Minnesota.*

[27] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, and S., Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. *In: Proceedings of the 35th SIGMOD international conference on Management of data, New York, NY, USA.*

[28] Seshadri, K., and Iye, K. V. (2010). Parallelization of a dynamic SVD clustering algorithm and its application in information retrieval. *Software: Practice and Experience, 40, issue 10, 883-896.*

[29] Song, W. and Park, S. (2007). Analysis of Web Clustering Based on Genetic Algorithm with Latent Semantic Indexing Technology. *In Proc. of Advanced Language Processing and Web Information Technology, 21-26.*

[30] Steinbach, M., Karypis, G., and Kumar, V. (2000). A Comparison of Document Clustering Techniques. *KDD-2000 Workshop on Text Mining, Boston, MA, USA*

[31] Tarpey, T., and Flury, B. (1996). Self-Consistency: A Fundamental Concept in Statistics. *Statistical Science, 11, 229-243.*

[32] Taura, K., Kaneda, K., Endo, T., and Yonezawa, A. (2003). Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources. *SIGPLAN Not., 38, 216–229.*

[33] Venner, J. (2009). *Pro Hadoop* (1st ed). New York: Springer.

[34] Wang, G., Butt, A. R., Pandey, P., and Gupta, K. (2009). Using realistic simulation for performance analysis of mapreduce setups. *In LSAP '09: Proceedings of the 1st ACM workshop on Large-Scale system and application performance.*

[35] White, T. (2009). *Hadoop: The Definitive Guide* (2nd ed.). CA : O'Reilly Media.

[36] Yan, B., Du, Y., and Li, Z. (2008). The New Clustering Strategy and Algorithm Based on Latent Semantic Indexing. *In Proc. of Natural Computation, 2008. ICNC '08. Fourth International Conference, 1, 486-490.*

[37] Yates, R. D., and Neto, B. R. (1999). *Modern Information Retrieval* (1st ed.). US: Addison- Wesley.

[38] "Url: www.google.com," 2005-2008.

[39] Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM, 30, 964-971.*

[40] Zhao, R., and Grosky, W. I. (2002). Narrowing the semantic gap-improved text-based web document retrieval using visual features. *IEEE Transactions On Multimedia, 4, 189-200.*

[41] Kurimo, M. (1999). Indexing audio documents by using latent semantic analysis and som. *In: Oja, E., Kaski, S. (Eds.), Kohonen Maps. Elsevier, Amsterdam, 363-374.*

[42] Souvannavong, F., Merialdo, B., and Huet, B. (2003). Video content modeling withlatent semantic analysis. *In the* 3*rd International Workshop on Content-Based Multimedia Indexing*.

[43] Littman, M., Dumais, S., and Landauer, T. (1996). Automatic cross-language information retrieval using latent semantic indexing. *In SIGIR'96 - Workshop on Cross-Linguistic Information Retrieval, 16-23.*

[44] Berry, M., Dumais, S., and O'Brien, G. W. (1995). Using linear algebra for intelligent information retrieval. *SIAM Review, 37, 573-595.*

[45] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18, 613-620.*

[46] He, B., Fang, W., Luo, Q., Govindaraju, N. K. and Wang, T. (2008). Mars: a MapReduce framework on graphics processors. *In: PACT '08: Proceedings of the 17thinternational conference on Parallel architectures and compilation techniques, 260–269.*

[47] Taura, K., Kaneda, K., Endo, T., and Yonezawa, A. (2003). Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources.

*SIGPLAN Not., 38, 216–229.*

[48] Aarnio, T. Parallel Data Processing with Mapreduce, Available at: http://www.cse.tkk.fi/en/publications/B/5/papers/Aarnio_final.pdf. (Last accessed: 29-Apr-2010).

[49] Wang, G., Butt, A. R., Pandey, P., and Gupta, K. (2009). Using Realistic Simulation for Performance Analysis of MapReduce Setups. *In: Proceedings of the 1st ACM workshop on Large-Scale System and Application Performance (LSAP '09), Garching, Germany.*

[50] Wang, G., Butt, A. R., Pandey, P., and Gupta, K. (2009). A Simulation Approach to Evaluating Design Decisions in MapReduce Setups. *In: Proceedings of the 17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '09), London, UK.*

[51] The Network Simulator - ns-2 Available at: http://www.isi.edu/nsnam/ns (Last accessed: 15-May-2010).

[52] Liu, Y., Li, M., Hammoud, S., Alham, N. K., and Ponraj, M. (2010). A MapReduce based distributed LSI. *In: Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), YanTai, China.*

[53] Alham, N. K., Li, M., Hammoud, S., Liu, Y., and Ponraj, M. (2010). A distributed SVM for image annotation. *In: Proceedings of the 7th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), YanTai, China.*

[54] Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., and Rasin, A. (2010). MapReduce and Parallel DBMSs: Friends or Foes? Communications of the ACM*, 53, 64-71*.

[55] Xie, J., Yin, S., Ruan, X., Ding, Z. Tian, Y., Majors, J., Manzanares, A., and Qin, X. (2010). Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on parallel & distributed processing symposium., Atlanta, GA*

[56] Sadasivam, G. S., and Selvaraj, D. (2010). A Novel Parallel Hybrid PSO-GA using MapReduce to Schedule Jobs in Hadoop Data Grids. *2010 Second World Congress on Nature and Biologically Inspired Computing, Kitakyushu, Fukuoka, Japan.*

[57] Groot, S. (2010). Jumbo: Beyond MapReduce for Workload Balancing. *VLDB*

*2010 , 36th International Conference on Very Large Data BasesSingapore.*

[58] Ghemawat, S., Gobioff, H., and Leung, S.T. (2003). The google file system. *In SOSP '03, 29-43, New York, NY, USA.*

[59] Devaraj Das. How to Hadoop. Available at: http://trac.nchc.org.tw/cloud/raw-attachment/Fwiki/HadoopWorkshop/hadoop-assembled.pdf (Last accessed: 03-Aug-2010)

[60] Lin, H., and Raghavendra, C. S. (1992). A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC). *IEEE Transactions on software engineering, 18, no. 2.*

[61] Zhang, Y., Kameda, H., and Hung, S. L. (1997). Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems. *IEEE Proc., Comput. Digit. Tech. 144, 2, 100–106.*

[62] Cai, W., Lee, B., Heng A., and Zhu, Li. (1997). A Simulation Study of Dynamic Load Balancing for Network-based Parallel Processing. *In Proceeding of ISPAN '97 Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks. 383-389.*

[63] Maeng, H. S., Lee, H. S., Han, T. D., Yang, S. B., and Kim, S. D. (2002). Dynamic Load Balancing of Iterative Data Parallel Problems on a Workstation Clustering. *High Performance Computing and Grid in Asia Pacific Region, International Conference on High-Performance Computing on the Information Superhighway, HPC-Asia '97. Seoul, Korea.*

[64] Zomaya, A. Y., and Teh, Y. H. (2001). Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE transactions on parallel and distributed systems, 12, 899-911.*

[65] Dhakal, S., Hayat, M. M., Pezoa, J. E., Yang, C. and Bader, D. A. (2007). Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach. *IEEE transactions on parallel and distributed systems, 18, 485-497.*

[66] Tonguz, O. K., and Yanmaz, E. (2008). The Mathematical Theory of Dynamic Load Balancing in Cellular Networks. *IEEE transactions on mobile computing, 7, 1504-1518.*

[67] Mahapatra, N. R., and Dutt, S. (1996). Random Seeking: A General, Efficient, and Iinformed Randomized Scheme for Dynamic Load Balancing. *10th International Parallel Processing Symposium (IPPS '96).*

[68] Zhou, S. (1988). A Trace-Driven Simulation Study of Dynamic Load Balancing. *IEEE transactions on software engineering, 14, no. 9.*

[69] Sun, N., and Lian, G. (2009). Dynamic Load Balancing Algorithm for MPI Parallel Computing. *2009 International Conference on New Trends in Information and Service Science.*

[70] Bahi, J. M., Vivier, S. C., and Couturier, R. (2005). Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms. *IEEE transactions on parallel and distributed systems 16,289-299.*

[71] Fatta, G. D., and Berthold, M. R. (2006). Dynamic Load Balancing for the Distributed Mining of Molecular Structures. *IEEE transactions on parallel and distributed systems. 17, 773-785.*

[72] Jie, W., Cai, W., and Turner, S. J. (2001). Dynamic Load-Balancing Using Prediction in a Parallel Object-oriented System. *IPDPS '01 Proceedings of the 15th International Parallel & Distributed Processing Symposium. Washington, DC, USA.*

[73] Hui C. C., and Chanson, S. T. (1999). Improved strategies for dynamic load balancing. *Journal IEEE Concurrency, 7 Issue 3.*

[74] Willebeek-LeMair, M. H., and Reeves, A. P. (1993). Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE transactions on parallel and distributed systems. 4, no. 9.*

[75] Dobber, M., van der Mei, R., and Koole, G. (2009). Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison. *IEEE transactions on parallel and distributed systems. 20, no. 2, 207-218.*

[76] Dean, J., and Ghemawat, S. (2010). MapReduce: A flexible data processing tool. *Commun. ACM, 53, 72-77.*

[77] Goda, K., Tamura, T., Oguchi, M., and Kitsuregawa, M. (2002). Run-time load balancing system on san-connected pc cluster for dynamic injection of cpu and disk resource - a case study of data mining application. *In DEXA, 182-192.*

[78] Groot, S., Goda, K., and Kitsuregawa, M. (2010). A study on workload imbalance issues in data intensive distributed computing. *In DNIS, 27-32.*

[79] Tamura, M., and Kitsuregawa, M. (1999). Dynamic load balancing for parallel association rule mining on heterogenous pc cluster systems. *In VLDB '99, 162-173, San Francisco, CA, USA.*

[80] Bonomi, F., and Kumar, A. (1990). Adaptive Optimal Load-Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler. *IEEE Trans. Computers, 39, no. 10, 1232-1250.*

[81] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, Mass.: Addison-Wesley, 1989.

[82] Pico, C. A. G., and Wainwright, R.L. (1994). Dynamic Scheduling of Computer Tasks Using Genetic Algorithms. *Proc. First IEEE Conf. Evolutionary Computation, IEEE World Congress Computational Intelligence, 2, 829-833.*

[83] Kidwell, M. D., and Cook, D. J. (1994). Genetic Algorithm for Dynamic Task Scheduling. *Proc. IEEE 13th Ann. Int'l Phoenix Conf. Computers and Comm., 61-67.*

[84] Lan, Y. and Yu, T. (1995). A Dynamic Central Scheduler Load-Balancing Mechanism. *Proc. IEEE 14th Ann. Int'l Phoenix Conf. Computers and Comm., 734-740.*

[85] Lee, S., Kang, T., Ko, M., Chung, G., Gil, J., and Hwang, C. (1997). A Genetic Algorithm Method for Sender-Based Dynamic Load-Balancing Algorithm in Distributed Systems. *Proc. 1997 First Int'l Conf. Knowledge-Based Intelligent Electronic Systems, 1, 302-307.*

[86] Sharykin, R. Hard Drive Modeling in CHARON. Available at: http://www.math.upenn.edu/~rsharyki/HDDM03.pdf. (Lasted accessed: 09-May-2010)

[87] Li, H., Wang, Y., Zhang, D., Zhang, M., and Chang, E. Y. (2008). Pfp: parallel fp-growth for query recommendation. *In RecSys '08, 107-114, New York, NY, USA.*

[88] Yang, C., Yen, C., Tan, C., and Madden, S. (2010). Osprey: Implementing mapreduce-style fault tolerance in a shared-nothing distributed database. *In ICDE '10, 2010.*

[89] Berry, M. W., Drmac, Z., and Jessup, E. R. (1999). Matrix, vector space, and information retrieval. *SIAM Review, 41, 335–362.*

[90] Dhillon, I. S., and Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. *Machine Learning, 42, 143–175.*

[91] Golub, G., and van Loan, C. (1989). *Matrix computation* (2nd ed.). Baltimore, MD: John Hopkins.

[92] Zha, H., and Zhang, Z. (1999). On matrices with low-rank-plus-shift structures: partial SVD and latent semantic indexing. *SIAM Journal on Matrix Analysis and Applications, 21, 522–536.*

[93] Zhang, Z., and Zha, H. (2001). Structure and perturbation analysis of truncated SVD for column-partitioned matrices. *SIAM Journal on Matrix Analysis and Applications, 22, 1245–1262.*

[94] April, K., and Pottenger, W. M. (2006). A framework for understanding latent semantic indexing performance. *J. Inf. Process. Manag., 42, no. 1, 56–73.*

[95] AswaniKumar, C., Gupta, A., Batool, M., and Trehan, S. (2005). An information retrieval model based on latent semantic indexing with intelligent preprocessing. *J. Inf. Knowl. Manag., 4, no. 4, 1–7.*

[96] Berry, M. W., and Shakhina, A. P. (2005). Computing sparse reduced-rank approximation to sparse matrices. *ACM Trans. Math. Software, 31, no. 2, 252–269.*

[97] Hua, Y. (2000). Searching beyond SVD for rank reduction. *Proc. IEEE Workshop Sensor Array and Multi Channel Signal Processing, Cambridge, MA, USA, 395 397.*

[98] Praks, P., Dvorsky, J., Snasel, V., and Cernohorsky, J. D. (2003). On SVD free latent semantic indexing for image retrieval for application in a hard real time environment. *Proc. IEEE Int. Conf. Industrial Technology, Maribor, Slovenia, 466–471.*

[99] AswaniKumar, C., and Srinivas, S. (2006). On the effect of rank approximation on information retrieval. *Proc. Int. Conf.Systemics Cybernetics and Informatics, Hyderabad, India, 876–880.*

[100] Yahoo. Hadoop at Yahoo! Available at: http://developer.yahoo.com/hadoop/. (Lasted accessed: 20-Nov-2010).

[101] Fan, J., Ravi, K., Littman, M. L., and Santosh, V. (1999). Efficient singular value decomposition via document samplings. *Tech. Rep. CS-1999-5, Dept. Computer Science, Duke University, North Carolina.*

[102] Shokripour, A., and Othman, M. (2009). Survey on Divisible Load Theory and its Applications. *International Conference on Information Management and Engineering, 300-304.*

[103] Robertazzi, T. G. (2003). Ten reasons to use divisible load theory. *Computer, 36, 63-68.*

[104] Li, X., Liu, X., and Kang, H. (2007). Sensing Workload Scheduling in Sensor Networks Using Divisible Load Theory, *Global Telecommunications Conference, 2007. GLOBECOM '07, 785-789.*

[105] Thysebaert, P., De Leenheer, M., Volckaert, B., De Turck, F., Dhoedt, B., and Demeester, P. (2005). Using Divisible Load Theory to Dimension Optical Transport Networks for Grid Excess Load Handling, *Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005, 89-94.*

[106] Jaber, T. (2008). Lexical Noise Modelling and Removal in Intelligent Information Retrieval, *1.*

[107] Trappey, AJ. C., Trappey, C. V., Hsu, F., and Hsiao, D. W. (2009). A Fuzzy Ontological Knowledge Document Clustering Methodology, *IEEE transactions on systems, man, and cybernetics—part B: cybernetics, 39, 806-814.*

[108] Ma, W., Wang, G., and Liu, J. (2007). Scalable Semantic Search with Hybrid Concept Index over Structure Peer-to-Peer Network, *The Sixth International Conference on Grid and Cooperative Computing(GCC 2007), 42-48.*

[109] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2006). *22nd International Conference on Data Engineering (ICDE'06), 95-104.*

[110] Pan, S., Mao, Q., and Zhang, Y. (2009). A Hybrid Web Service Selection Approach Based on Singular Vector Decomposition, *2009 Congress on Services - I, 724-731.*

[111] Ungrangsi, R., Anutariya, C., and Wuwongse, V. (2007). Enabling Efficient Knowledge Reuse in the Semantic Web with SQORE*, *Third International Conference on Semantics, Knowledge and Grid (SKG 2007), 92-97.*

[112] Ren, K., Liu， X., Chen, J., Xiao, N., Song, J., and Zhang, W. (2008). A QSQL-based Efficient Planning Algorithm for Fully-automated Service Composition in Dynamic Service Environments, *2008 IEEE International Conference on Services Computing, 301-308.*

[113] Ren, K., and Yang, C. (2008). Building Quick Service Query List(QSQL) to Support Automated Service Discovery and Composition, *Proceedings of 2008 IEEE International Symposium on IT in Medicine and Education, 4-4.*

[114] Guo, P., Wang, X., and Han, Y. (2010). The enhanced genetic algorithms for the optimization design, *2010 3rd International Conference on Biomedical Engineering and Informatics, 2990-2994.*

[115] Han, L. Q. (2002). Theory, design and application of artificial neural network, *Chemical industry publishing company.*

[116] Yan, T., Cui, D., and Tao, Y. (2010). A New Evolutionary Neural Network Algorithm Based on Improved Genetic Algorithm and its Application in Power Transformer Fault Diagnosis, *2007 Second International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2007), 1-5.*

[117] Yan, T. (2010). An Improved Genetic Algorithm and its Blending Application with Neural Network, *2010 International Conference on Intelligent Computation Technology and Automation, 548-551.*

[118] Dong, Y. F., Gu, J. H., Li, N. N., Hou, X. D., and Yan, W. L. (2007). Combination of Genetic Algorithm and Ant Colony Algorithm for Distribution Network Planning, *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, 19-22.*

[119] Qiu, H., Zhou, W., and Wang, H. (2009). A Genetic Algorithm-Based Approach to Flexible Job-Shop Scheduling Problem, *2009 Fifth International Conference on Natural Computation, 81-85.*

[120] Burke, H. B., Rosen, D. B., and Goodman, P. H. (1994). Comparing artificial neural networks to other statistical methods for medical outcome prediction, *1994 IEEE World Congress on Computational Intelligence, 2213-2216.*

[121] Bandyopadhyay, M., and Bhaumik, P. (2010). Zone based ant colony routing in mobile ad-hoc network, Communication Systems and Networks (COMSNETS), *2010 Second International Conference on COMmunication systems and NETworks.*