Performance Based Data-Distribution Methodology In Heterogeneous Hadoop Environment

Miss.Vrushali Ubarhande

Submitted as part of the requirements for the degree
of MSc in Cloud Computing
at the School of Computing,
National College of Ireland
Dublin, Ireland.

August 2014

Supervisor Dr. Alina-Madalina Popescu

Abstract

Hadoop has been developed to process the data-intensive applications.However, the current data-distribution methodologies are inefficient for heterogeneous environment such as cloud computing. Performance of Hadoop may degrade in heterogeneous environment, whenever data-distribution is not as per the computing capability of the nodes. In this work, the existing research methodologies have been critically evaluated to understand the current data-distribution techniques developed.

In Hadoop framework, users specify the application computation logic in terms of a map and a reduce function, often termed as MapReduce applications. Hadoop distributed file system is used to store the MapReduce application data on the Hadoop cluster nodes called as Datanodes, whereas Namenode is a control point for all Datanodes.

The concept of data-locality and its impact on the performance of Hadoop are discussed. The data-distribution is a key factor in Hadoop, because it may affect performance in Map phase for scheduling task. The task scheduling techniques in Hadoop consider the data-locality as a key factor to enhance performance. Various task scheduling techniques have been analyzed to understand the affirmative effect of the emphasizing high data-locality while scheduling. Other system factors also play a major role while achieving high performance in Hadoop data processing.

The main contribution of this work is to prove a performance increase in Hadoop by the effective distribution of data in heterogeneous environment. An experiment has been proposed to adopt novel data placement strategy based on Datanodes's capability in Hadoop. In this experiment, Speed Analyser component is created to measure the processing capability of each Datanode. Thus, Speed Analyser calculates the computing ratio of each Datanode based on their response times. The Data-Distribution Technique is integrated with the traditional Hadoop and uses the calculated computing ratio. Based on the computing ratio, Namenode decides the assignment of the data blocks to the Datanodes.

Thereafter, two MapReduce applications were executed to understand the performance

improvement after the implementation of the proposed Data-Distribution Technique. Further, the future scope for improving the proposed solution is identified.

Keywords: MapReduce, data placement, Hadoop, data locality, performance based data distribution, cloud computing, heterogeneous Hadoop cluster.

Acknowledgement

This dissertation work “Performance Based data-distribution methodology in Hadoop for heterogeneous cloud infrastructure” for Masters in Cloud computing is accomplished at National College of Ireland, Cloud competency Centre.

I am grateful to my supervisors Dr.Horacio Gonzalez-Velez and Dr.Alina-Madalina Popescu. I would like to thank them for their guidance, efforts and discussions during the course work. Every meeting including the dissertation clinic helped me to achieve the desired results in the research work.

I wish to thank Dr.Keith Brittle, Mr.Michael Bradford and Mr.Robert Duncan for their quick help for completion of this course work successfully. I would like to thank all people who helped directly and indirectly to complete the research work.

Declaration

All the work submitted as a part of this research work is solely implemeneted and edited by me. All the sources of information and knowledge are reference properly inside the document.

Signed ..... Date .....

Miss. Vrushali Ubarhande

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The processing of data intensive applications is becoming complex.MapReduce is a prototype designed by Google, which enables the processing of large datasets in heterogeneous environment. MapReduce uses the Map and Reduce functions to achieve maximum data parallelization in the cluster.

Similarly, Hadoop is an open-source implementation of the MapReduce model by Apache. Kala Karun & Chitharanjan (2013) have mentioned that Hadoop handles data intensive applications by the division of large tasks into smaller jobs and breaking substantial datasets into smaller partitions, so that each small job operates on an individual partition in parallel. Although, Hadoop maximizes the parallelization of data processing, Xie, Yin, Ruan, Ding, Tian, Majors, Manzanares & Qin (2010) suggest that the available Hadoop data processing schemes are designed for the homogenous environment. Such schemes may not be so useful considering the dynamic nature of Hadoop while operating applications in a cloud.

Xie et al. (2010) also point out that the current Hadoop data processing framework considers all nodes as having equal computing capability with the same disk space. They explain the situation in a heterogeneous environment where the performance of nodes varies and disk space also. In such a case, data may need to be transferred from low-speed nodes to the high-speed nodes to finish the task as early as possible. As the amount of data is large in MapReduce Hadoop, data transfer is more expensive and can consume the significant bandwidth in the network leading to a decrease in performance. Therefore, in order to improve the Hadoop infrastructure performance, an effective division of data is essential to avoid extra overheads of data transfer.

Therefore, this research work is mainly focused on the creation of the Data-Distribution Technique which will consider the processing power of each Datanode before assigning

data blocks. The proposed solution intends to improve the performance of heterogeneous Hadoop environment by effective Data-Distribution. This is because the existing data-distribution methodology in a Hadoop does not consider the heterogeneous nature of Hadoop in cloud as suggested by Fan, Wu, Cao, Zhu, Zhao & Wei (2012).

This paper starts with the evaluation of the previous solutions proposed for Data-Distribution in heterogeneous Hadoop cluster. Further, the design of the experiment is discussed in Chapter 3. Chapter 3 includes the explanation of the component such as Speed Analyser and Data-Distribution Technique. In addition, the implementation details will be discussed in the Chapter 4. Thereafter, the results recorded after execution of the MapReduce application on traditional Hadoop and Hadoop system with proposed solution is analysed. Finally, the Chapter 6 includes the conclusions of this research work and the future scope is identified.

Chapter 2

Background

The main purpose of this chapter is to evaluate the literature review in order to understand the research gap. This chapter starts with the explanation of the Hadoop basic architecture in Section.2.1. Further, Section.2.2 describes the evaluation of the existing proposed methodologies for effective data distrbution for heterogeneous Hadoop cluster with respect to the data locality. In addition, different scheduling techniques are also studied to understand the importance of the data-locality in scheduling approaches under Section.2.3.

2.1 Hadoop Background

Apache Hadoop is an open source implementation of Google MapReduce programming model. The Hadoop software framework has two main parts Hadoop Distributed File System (i.e.HDFS) and the MapReduce programming model. HDFS works as a storage for MapReduce application data, while the MapReduce programming includes the logic to process the MapReduce application data stored on HDFS.

Master node is a collective term used for JobTracker and Namenode. The Namenode is a part of HDFS, whereas the JobTracker is a part of MapReduce programming model. The HDFS and MapReduce programming model is described in detail below.

2.1.1 Hadoop Distributed File System

The Fig.2.1 describes the data flow management in HDFS.

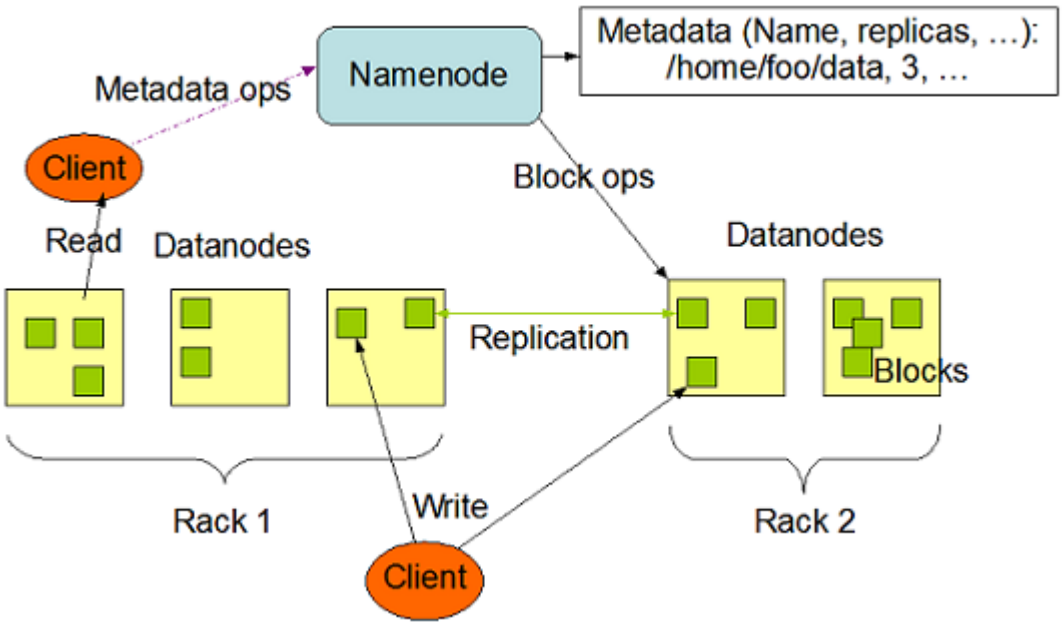

Figure 2.1: Hadoop distributed file system (*HDFS Architecture Guide* (2008))

The master/slave architecture is a basic approach used in HDFS. Namenode is a master server which administers the allocation of files (i.e. file system namespace), maintenance of file metadata and clients data access in Hadoop cluster. Under Namenode, many Datanodes can co-exist. Usually, there is one Datanode per node in the HDFS. Datanode acts as storage for a node on which Datanode runs. HDFS discloses the file system namespace and ease the storage of application data in files for clients. In HDFS, the application data files is divided into data blocks. These data blocks are eventually stored on datanodes. Namenode performs all the file namespace operations such as deciding destination datanode storage, opening, closing and renaming of files. Datanode can also perform the data read operations from client (*HDFS Architecture Guide* (2008)).

2.1.2 MapReduce programming model

The below Fig.2.2 represents the execution of MapReduce applications in Hadoop.

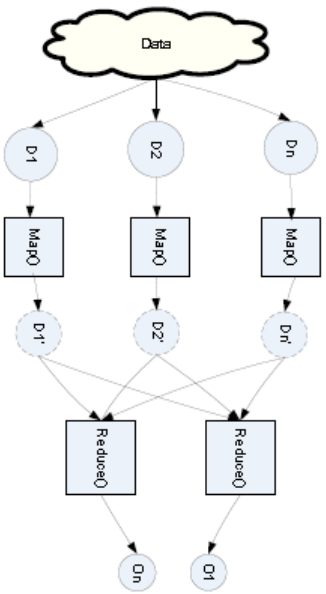

Figure 2.2: MapReduce Programming Model (Xie et al. (2010))

In Hadoop system, when MapReduce applications start executing, the MapReduce programs application data requests are processed by Namenode. The clients upload the application data in HDFS before running MapReduce application. MapReduce application always run in two parts, first map function and then reduce function. Namenode transfers the data requests to individual Datanodes which feed the data to map functions (Xie et al. (2010)).

The below figure introduced the MapReduce programming mode hierarchy.

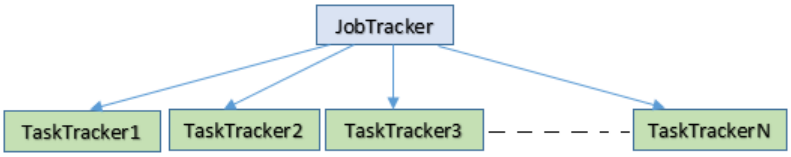

Figure 2.3: MapReduce Programming Model Hierarchy

The JobTracker distributes the job (i.e. MapReduce application) breaking down into pieces called as tasks to each Datanode. In MapReduce programming model, each Datanode acts as a TaskTracker also. The subsequent reduce function after map function also run on Datanodes. The progress of the MapReduce application can be checked using the web interface of the Hadoop system.

2.2 Importance of data locality in Hadoop

2.2.1 Exploring Data Locality

Guo, Fox & Zhou (2012) define data-locality as closeness between a node with input data and a compute node. "Data locality is defined as how close compute and input data are, and has different levels node-level, rack-level, etc."(Guo, Fox & Zhou 2012, p.26)

Nguyen, Simon, Halem, Chapman & Le (2012) remark that Hadoop exploits data-locality for scheduling task to improve performance. Here, the exploitation of data-locality in Hadoop means the execution of a task should be on a node or near a node where input data is residing.

Chung, Graham, Bhagwan, Savage & Voelker (2006) have proposed one of the few early data division techniques for distributed systems. In this methodology, minimum data-locality is achieved for each task by reducing the number of fragments for each task. This data division technique using minimum data-locality may not be proved efficient in a heterogenous environment like Hadoop where the number of nodes and data size and its nature varies according to applications. Guo et al. (2012) mention that the achievement of data-locality can be perceived as keeping data near to the compute node. Zhang, Feng, Feng, Fan & Ming (2011) explain that the input data for Hadoop MapReduce is bigger in size compared to normal applications. The movement of such a vast amount of data can affect performance of Hadoop by increasing overheads on network bandwidth as suggested by Xie et al. (2010).

2.2.2 Replication for achieving data-locality

Eltabakh, Tian, Özcan, Gemulla, Krettek & McPherson (2011) have introduced the approach CoHadoop in which relevant data-placement on the same group of nodes is proposed. CoHadoop allows applications to control where data is stored and address the performance bottleneck identified in Hadoop. They consider the data - relevance as an important factor to separate out data in Hadoop in the data-distribution phase. CoHadoop makes use of data replication retaining the strong fault tolerance properties of Hadoop CoHadoop by is mainly designed to colocate files for faster query execution.

To place relevant files on the same group of nearer nodes may not be always possible in Hadoops filesystem as the amount of space varies for each node. Moreover, Kala Karun & Chitharanjan (2013) agree that the colocation approach in Hadoop can work well if there is an ample amount of space on each node and failures are less. In addition, Jin,

Luo, Song, Dong & Xiong (2011) mention that a file is divided into a number of blocks. So, the CoHadoop approach by Eltabakh et al. (2011) might not work where the size of data is big which is common in Hadoop applications.

According to Xie et al. (2010), tasks may need to be transferred from a low-performance to a high - performance node to boost response time in case of delay due to computing limitations in heterogeneous systems. Zaharia, Borthakur, Sen Sarma, Elmeleegy, Shenker & Stoica (2010) define such a sufficiently low-performance node in Hadoop as a straggler. They argue that the nodes can be low-performing because of many factors such as network sharing or disk sharing in case of virtual machines.

Guo et al. (2012) explain that the reduction of network congestion by increasing the data - locality can be considered as an effective way to improve performance as the data size is larger in Hadoop. More data-locality, less the data transfer. Dean & Ghemawat (2008) mention that the network bandwidth is a limited resource compared to the computing power of nodes in the Hadoop infrastructure. So, reduction in data movement in Hadoop can be considered as the key factor to improve Hadoop performance as mentioned by Xie et al. (2010). They mention that the measurement of heterogeneity is essential in a dynamic Hadoop as the capability of all nodes are not same.

2.2.3 Node capability based data-distribution techniques

Xie et al. (2010) and Fan et al. (2012) have developed performance-aware data placement and data-distribution techniques respectively.

In the data-distribution technique outlined by Xie et al. (2010), the data-locality factor can be increased by a proper data distribution to the nodes on the basis of the nodes computation ratio. Computation ratio is calculated by running small test jobs on nodes in this data-distribution technique. Zaharia et al. (2010) mention that the distribution of data should be very cautious as the tasks will be scheduled according to data-locality in Hadoop.

Even though the approach developed by Xie et al. (2010) considers the heterogeneity of the environment through calculating the computing ratio by running test jobs on all nodes, it may not be a suitable approach in Hadoop where nodes are very large in numbers. The distribution of test jobs may lead to congestion in the network. Moreover, in the dynamic cloud, Fan et al. (2012) suggest that the node on which test script runs may not actually have same capability for actual task execution in Map phase. In case of a straggler node, to ease data transfer from low-performance node to high-performance node, Xie et al. (2010) have used the data replication approach to

increase data-locality.

On the other hand, Guo et al. (2012) explain the contrary effect of a data replication factor in the performance of Hadoop if used on a larger scale. On a broader scale, replication may use vast amounts of disk space and network bandwidth. Even though, replication of data improves data-locality, the performance may gradually decrease as the amount of replicated data increases. Guo et al. (2012) prove that careful judgement of replication policy in Hadoop is important.

In addition, Ye, Huang, Zhu & Xu (2012) proposed a unique way of replication in Hadoop. Ye et al. (2012) have considered the default replication factor in Hadoop i.e. 3, they have focused on the realtime conditions of nodes for replication. Consideration of real time space utilization of nodes compared to threshold platform utilization for Hadoop, while selecting datanode in local rack and second-rack for replication can be fruitful. This approach can be more suitable for Hadoop due to realtime situation analysis for replication of data blocks resulting in reduction in the network traffic because of less transfer of data.

Chervenak, Deelman, Livny, Su, Schuler, Bharathi, Mehta & Vahi (2007) have suggested an earlier methodology to place data near high-performance nodes, so that the task can be redirected to a high-performance computing resources with data easily to gain high data-locality in distributed systems.They also emphasize moving data off computational resources quickly when computation is complete.

Shen & Zhu (2009) have introduced quite similar methodology to use replication effectively to increase data-locality in the network. Shen & Zhu (2009) have introduced the making of replicas on physically high-performance close nodes. The replication method by Shen & Zhu (2009) also includes the redirection algorithms between high-performance replica nodes and low-performance nodes which will improve response time.

A distinctive performance aware technique has been created by Fan et al. (2012).They have assessed the node performance through historical logs on nodesand measure node capability based on the executed task and computing node pair system logs. Such measurement of node capability based on logs may fail as the performance of the node can be affected by many other system factors suggested by Guo et al. (2012). They have evaluated the impact of system factors such as Input/output delay, scheduling wait time, free slots, number of tasks and the number of nodes on the data-locality. So, node capability measured on the basis of historical logs cannot be considered an accurate heterogeneous measurement.

To calculate node capability in case of an input change or termination of a job which may influence logs, Fan et al. (2012) have used benchmarking for the ideal finish time of faster nodes and slower nodes.They divide the nodes based on finish time to a slow node queue and a fast node queue. When half of the map tasks in the slowest node in slow node queue are completed, the map task load will be transferred from the slowest node to the fastest node in the fast node queue. Monitoring of the computing capability of nodes based on historical logs may cause issues when a large number of new nodes are added to a cloud for which no historical logs are available.

A mixed approach have been designed by Arasanal & Rumani (2013). In this approach, the nodes will send signals back to namenodes in Hadoop for the available hardware resources, then the computing ratio for each node will be calculated. This computing ratio cannot be considered as optimal because of network bottlencks, latency in I/O operations or the type of data processed. Moreover, the signals from the nodes to the namenodes can create unnecessary network traffic. Arasanal & Rumani (2013) compare the computing ratio calculated in the above step with the historical logs of the previous jobs to come up with the commensurate computing ratio. On the other hand, they mention that the computing capability of nodes cannot be determined properly for quadratic complexity (i.e. in which problem solving takes four times the duration when the problem size doubles compared to original problem size) of map() function, so they have considered the quadratic complexity of map() function as well for distribution of data along with the computing ration.

The huge amount of tasks may need to be rescheduled on another high-performance node if the new nodes are low-performing in the technique outlined by Fan et al. (2012). Guo et al. (2012) explain that scheduling should be optimized globally in the case of heterogeneous systems to improve Hadoop performance.

Table 2.1: Comparison of different data-distribution techniques

Technique	**Computing ratio calculation**	**Possible weakness**
Data-distribution by Xie et al. (2010)	Run test job to know node's speed	Unnecessary use of bandwidth for distribution
Data-distribution by Fan et al. (2012)	based on historical logs	May not be fruitful when history is unavailable
Data-distribution by Arasanal & Rumani (2013)	based on historical logs complexity of map() function configuration available through heartbit signal	Complexity of map function may not affect response time after certain cluster size.

2.3 Need of data-locality in scheduling approaches

The Hadoop MapReduce framework is used to process tasks in parallel as described by Kurazumi, Tsumura, Saito & Matsuo (2012).

Zaharia et al. (2010) mention that the map task scheduler in Hadoop chooses the data-locality of a task while scheduling over all other system factors. The task will wait for a small amount of time to get executed on a data local node in delay scheduling as outlined by Zaharia et al. (2010).

Guo et al. (2012) mention that the basic principle for Hadoop is that the compute node and input node should be the same. Kurazumi et al. (2012) explain that tasks may need to be scheduled on a different node than an input node because of the lack of availability of input nodes, task failure or the delay in task processing in heterogeneous environments.

Xie et al. (2010) mention that the change in input data can be a major concern after task scheduling in Hadoop, where rescheduling and data transfer is needed. According to Kurazumi et al. (2012), shifting map tasks as well as input data from one node to another may reduce actual utilization of CPU resources because of the Input / Output wait. So, to improve performance, effective distribution of data in the initial phase can be considered as an important factor so that a maximum number of map tasks can be completed on the same node reducing data transfer in Hadoop as suggested by Guo et al. (2012). Similarly, Cardosa, Wang, Nangia, Chandra & Weissman (2011) suggest that tasks scheduled on a data-local node usually tend to finish quickly as no network transfer and latency are involved.

Zaharia et al. (2010) have proposed the delay scheduling algorithm which uses fairness i.e. giving new jobs their fair share of resources quickly and data-locality as two key factors to improve Hadoop performance. To achieve data-locality, Zaharia et al. (2010) incorporate waiting for a scheduling task on a node with input data. Such a delay in the scheduling of tasks may hamper overall runtime performance in a cloud which can sometimes be greater than the actual delay in the transfer of data to the new node. Jin et al. (2011) argue that the delay scheduling outlined by Zaharia et al. (2010) may not work properly when the nodes with input data are freed up slowly.

Another scheduling technique BAR: Balance-Reduce is proposed by Jin et al. (2011) where task schedules globally consider the overall network condition and data-locality is achieved according to workload and network state. In BAR: Balance-Reduce by Jin et al. (2011), data-locality is achieved by running a task on a node near to a data-local node when Hadoop is overloaded.

Zhao, Wang, Meng, Yang, Zhang, Li & Guan (2012) propose a distinct approach of co-locating relevant data during data distribution which may increase performance. The use of method BAR: Balance-Reduce as outlined by Jin et al. (2011) with co-locating relevant data as per Zhao et al. (2012) can increase the overall Hadoop performance as the relevant data will be situated near to the data-local node.

Zhang et al. (2011) have explained that the response time for the execution of the map task may be affected by the waiting time to schedule tasks on a data-local node and data transfer time between the requesting node and an input node. In the case of a straggler node, Zaharia, Konwinski, Joseph, Katz & Stoica (2008) have developed the approach LATE in which the straggler node is discovered easily. In LATE, data-locality is achieved through the effective replication of data in the first phase which increases performance in the case of a straggler node. Zaharia et al. (2008) have improved the speculative execution of tasks in LATE. Speculative execution of the task is defined as running multiple copies of the same task on different nodes, whichever node finishes a task first will report to the task scheduler.

Nguyen et al. (2012) have proposed a HybS task scheduling policy for MapReduce. In this policy, they have given priority to data-locality instead of other factors such as priority of jobs. Nguyen et al. (2012) decide task priority on the basis of Hadoop historical logs in the HybS scheduling algorithm. In a similar way, Fan et al. (2012) propose data - distribution on the basis of historical logs as we have covered in the above data-locality approach section.

Nguyen et al. (2012) choose to implement reordering of task priority to gain the benefit of data-availability. Scheduler by Nguyen et al. (2012) may work better than the delay scheduling algorithm by Zaharia et al. (2010). In case of unavailability of data-local node, scheduler by Nguyen et al. (2012) favors the data-locality as maximum as possible by re-ordering of tasks priority.

The main issue in HybS by Nguyen et al. (2012) may arise when the vast number of nodes are added to the network as no logs will be available for new nodes.

Table 2.2: Comparison of different task-scheduling techniques based on data-locality

Scheduling Technique	**Factor for achieving data-locality**	**Possible weakness**
Delay-scheduling algorithm by Zaharia et al. (2010)	resource fairness and running tasks on data-local node	Nodes can free up slowly
BAR: Balance-Reduce by Jin et al. (2011)	running tasks on node near to data-local in case of overhead	Near nodes may not be free always.
LATE by Zaharia et al. (2008)	Effective replication and running same task parallely, whichever node finish early will report, other copies of same task will be stopped.	Unnecessary CPU utilization and use of storage for replicas.
HybS task scheduler by Nguyen et al. (2012)	Reordering of task priority on the basis of historical logs	May not be fruitful in case of new nodes.

Dean & Ghemawat (2008) mention that the network bandwidth is a limited resource compared to the computing power of nodes. To reduce the stress on bandwidth, the scheduling of map tasks on the basis of data-locality is crucial. Data-locality can be achieved through effective distribution of data during the data placement scheme before Map task scheduling. Guo et al. (2012) have analyzed the Hadoop performance in terms of different system factors and the data-locality. The impacting factors may include the number of nodes, the overall number of tasks for particular applications, the rate of map tasks executed by each node, the replication factor and the amount of idle time (map) slots. The effects of these system factors are explained below.

Chapter 3

Design

The homogeneous data placement and data distribution policies in Hadoop may not prove efficient in long run for heterogeneous Hadoop environment. To improve the performance of Hadoop, the effective data distribution technique is proposed which will exploit the heterogeneous nature of slave nodes in a cluster. The different steps involved to achieve end results are explained here. Firstly, the proposed architecture of the Data-Distribution technique for heterogeneous Hadoop cluster is introduced. After that all the components from the proposed experimental design are discussed below. It includes the Speed Analyser and the Data-distribution technique. Moreover, the different users of the system are identified and explained with the help of use cases. To conclude, the interaction of the user and data-flow within the Hadoop system is presented.

3.1 Proposed Data-Distribution architecture for Heterogeneous Hadoop cluster

The proposed solution for this research work ensures the distribution of the data blocks within the Hadoop cluster based on the computing capability of each slave node, in order to improve the performance. Thus the overall view of the proposed solution is represented in Fig. 3.1. The proposed solution is intended to be developed within a heterogeneous Hadoop environment with seven slave nodes. First, the Speed Analyser component will be created on Namenode, which will be installed and executed on each slave node. Furthermore, the master node will read the response time taken by each slave node from respective log file, and will create a file with the computing ratio. This file will be feed to Data-Distribution algorithm through the Hadoop Distributed

FileSystem. More information about the Speed Analyser and Data-Distribution components are explained in below subsections.

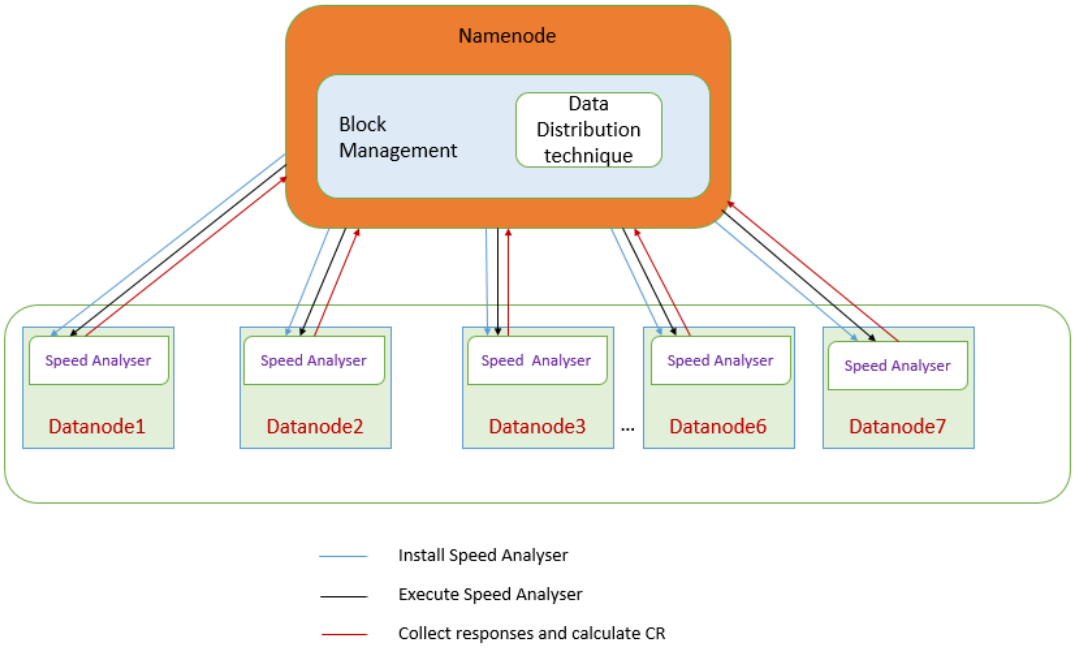

Figure 3.1: Proposed Data-Distribution architecture for Heterogeneous Hadoop cluster

3.1.1 Speed Analyser

To achieve the desired results for specified experiment, Speed Analyser component is designed as below.

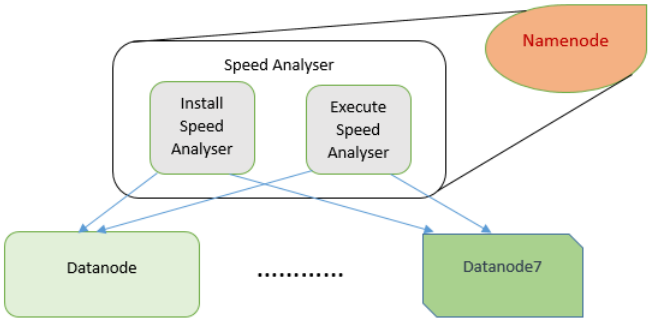

Figure 3.2: Speed Analyser

The Speed Analyser agent was created to measure the processing capacity of each slave node. Namenode will install and execute Speed Analyser agent on each slave node.

Speed Analyser records response time taken by Speed Analyser in the log file. The implementation details of Speed Analyser is explained in Section 4.

3.1.2 Data Distribution Technique

The Data-Distribution technique makes use of response times available after the execution of the Speed Analyser. The main motive of data distribution technique is to distribute data according to the calculated computing ratio of the slave nodes. The computing ratio is calculated using the response time available using Speed Analyser.

Dynamic data placement algorithm for heterogeneous nodes

The current data placement and data distribution algorithm in Hadoop is devised for the homogeneous cluster. To exploit the resources in heterogeneous Hadoop, it is necessary to develop a new dynamic data placement algorithm which will take the capability of datanodes into account while data distribution.
New data placement strategy will involve below steps:

1. Get the responses from the SpeedAnalyser agent.
2. The computing ratio will be calculated based upon response times compared to slowest node.
3. The number of data blocks processed by slowest node will be considered while assignment.
4. Assign the number of blocks to the datanodes as per their assignment ratio.
5. The average response time will be calculated.
6. All nodes having response time more than average will be assigned replicated data blocks for high performance nodes.

Thus, the Data-Distribution component of the proposed solution is based on the data-distribution algorithm, which will provide the distribution mechanism to utilize across all the slave nodes involved. In this sense, the data-distribution algorithm mainly involves the calculation of the processing capability of each node in the specified Hadoop cluster. This algorithm starts with the collection of the response time from the log file of each slave node generated by the Speed Analyser execution. The computing ratio for each slave node will be calculated based on the below formula, where the Highest Response Time (i.e HTR) and Individual Node Response Time (i.e. INRT) are

considered for the computation:

$$CR = HRT/INRT$$

where CR stands for Computing Ratio, HRT stands for Highest Response Time and NRT stands for Individual Node Response Time.

Furthermore, as a part of the Data-Distribution technique, the total number of blocks will be calculated by dividing the input file's size from the NameNode with the default Hadoop block size (i.e. 64MB), as described in the below formula:

$$TNB = IFS/DHBS$$

where TNB stands for Total No. of Blocks, IFS stands for Input File's Size, DHBS stands for Default Hadoop Block Size and the unit of data was considered as MB.

After this, the number of blocks processed by the slowest node is calculated by dividing the total number of blocks with the sum of computing ratios for each node.

$$x = TNB/\sum_{i=1}^{n} CR$$

Moreover, as the last step of the algorithm, the data is assigned to the nodes based on the below formula:

$$y = CR * x.$$

In order to describe all the process of the proposed data-distribution algorithm, an example of the calculation of the processing capability of the slave nodes based on the proposed algorithm is provided in the below subsection.

Calculation of processing capability

The amount of data processed by each node is inversely proportional to the response time for each application. As the response time is increasing, the amount of the processed data will be decreased for that node. Thus, the relation between the processed data and the time taken to process it (i.e. response time) can be represented as:

$$y \propto 1/T$$

where y is the processed data by a particular node and T is the Time taken to process data y.

Let's consider that the three slave nodes have the following response times as T1, T2 and T3 from Speed Analyser, whereas x is considered as a number of blocks processed by the slowest node which has highest response time (i.e. HTR). Furthermore, the data processed by each node can be represented with the help of inverse proportion as in Table 5.2.

Table 3.1: Respose time and data proportion

Response Time in secs	**Data processed**
T1	T3/T1x
T2	T3/T2x
T3	x

Let's consider TNB as the Total Number of blocks in an MapReduce application. T3 is the highest response time (i.e. slowest node's response time). So the equation can be formed from above values as below:

$$T3/T1 * x + T3/T2 * x + x = TNB$$

So, x (i.e. no of blocks processed by slowest node) can be calculated as

$$x = TNB/(T3/T1 + T3/T2 + T3/T3)$$

where T3/T1 is the computing ratio of slave node with response time T1. The resulting equation for x can be described as below:

$$x = TNB/\sum_{i=1}^{n}(HRT/INRT)$$

After calculating the number of blocks processed by slowest node, the data assigned to each node can be mentioned as:

$$y = CR * x.$$

In the end, data-distribution will assign y number of blocks to each node depending on their Computing ration value.

3.2 Use cases for the proposed solution

The user interaction with the proposed system is explained in Fig. 3.3.

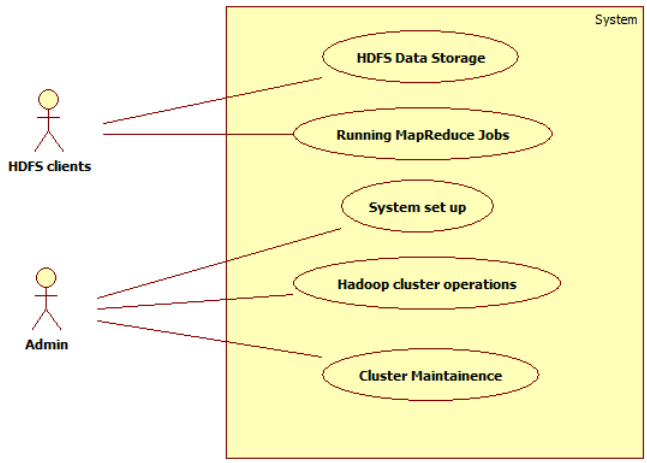

Figure 3.3: Use case for the proposed solution

There are two main identified roles of users such as client and system administrator in the proposed system. The Admin user is responsible for setting up of the Heterogeneous Hadoop cluster on the OpenStack private cloud. The Admin user will also carry out the responsibilities such as: starting and stopping of Hadoop cluster, the operations related with the maintenance and the integration of new components.

In addition, the clients will use the existing Hadoop cluster to store the application data on HDFS storage. The clients mainly use the Hadoop cluster to run MapReduce applications. The client user mainly comes into picture during the evaluation process of the experiment.

3.3 Sequence diagram with the Data Flow of the Proposed Solution

The set of events while storing data in Hadoop have been tracked in the below sequence diagram (Fig. 3.4). The main purpose of this diagram is to capture the flow of the Speed Analyser interactions with each Datanode (i.e. slave node). Thus, this diagram explains the interaction of clients with the proposed system. This client interaction starts with the request to store data on HDFS storage.

Before starting to store data on HDFS, the Speed Analyser will be installed and executed on each Datanode. Furthermore, the response from Speed Analyser will be sent back to namenode from each Datanode. The data storage process in HDFS starts with calling the object of DistributedFileSystem which eventually uses the FSDataOutputStream object to write data to each Datanode. FSDataOutputStream object interacts with Namenode in order to receive the list of suitable Datanodes for storing the data blocks.

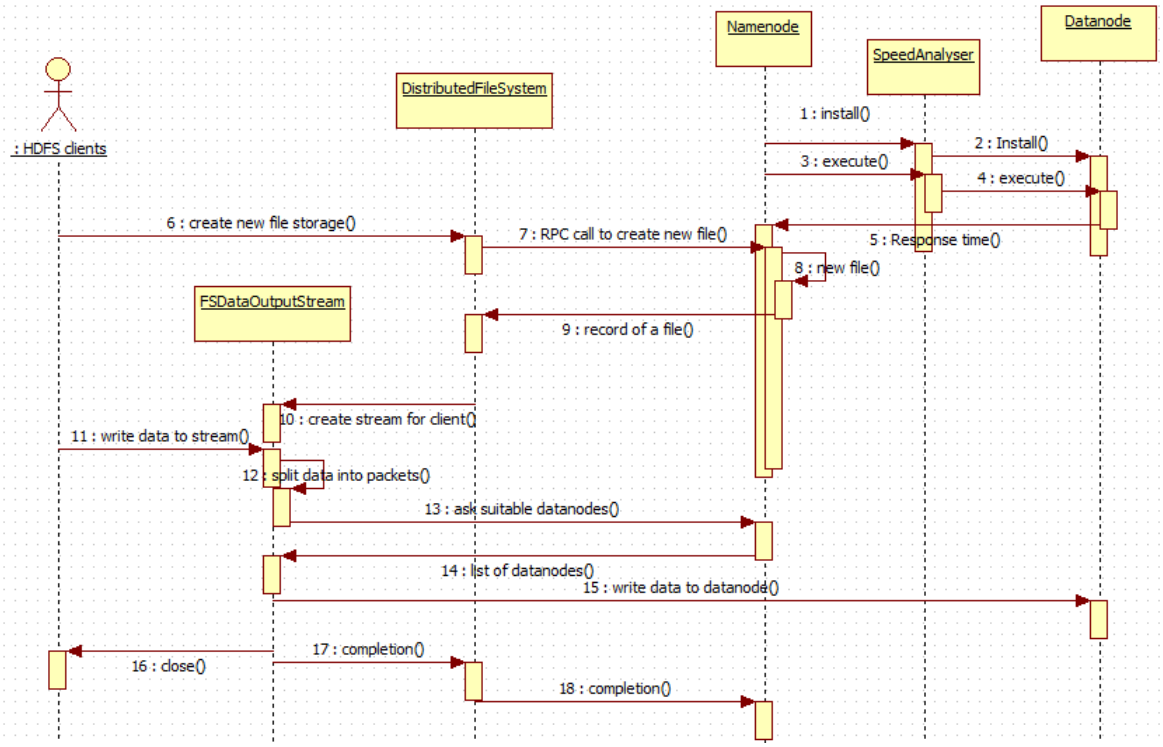

Figure 3.4: Sequence diagram with the Data Flow of the Proposed Solution

Chapter 4

Implementation

The implementations details of all the components of the proposed solution are covered in this section. Firstly, the heterogeneous Hadoop cluster environment was set up to implement and evaluate the proposed Data-Distribution solution. In the beginning of this chapter, the OpenStack cloud architecture is explained as the specified Hadoop cluster was created on OpenStack private cloud. Secondly, the procedure adopted to create a heterogeneous Hadoop cluster is described. Furthermore, the proposed Speed Analyser implementation details are explained. The chapter ends with the description of the implementation and the integration of the proposed Data-Distribution component with the Hadoop source code.

4.1 OpenStack Architecture

This section provides a brief description of the OpenStack cloud architecture. The OpenStack private cloud was selected as a environment where Hadoop cluster was created for this proposed research work. The OpenStack is an open source software which provides an efficient way to implement and management of scalable cloud infrastructure as a service platform. User can manage to access different OpenStack cloud services with the help of different Application Programming Interfaces (API) based on their choice. The basic architecture of the OpenStack is described below in Fig. 4.1.

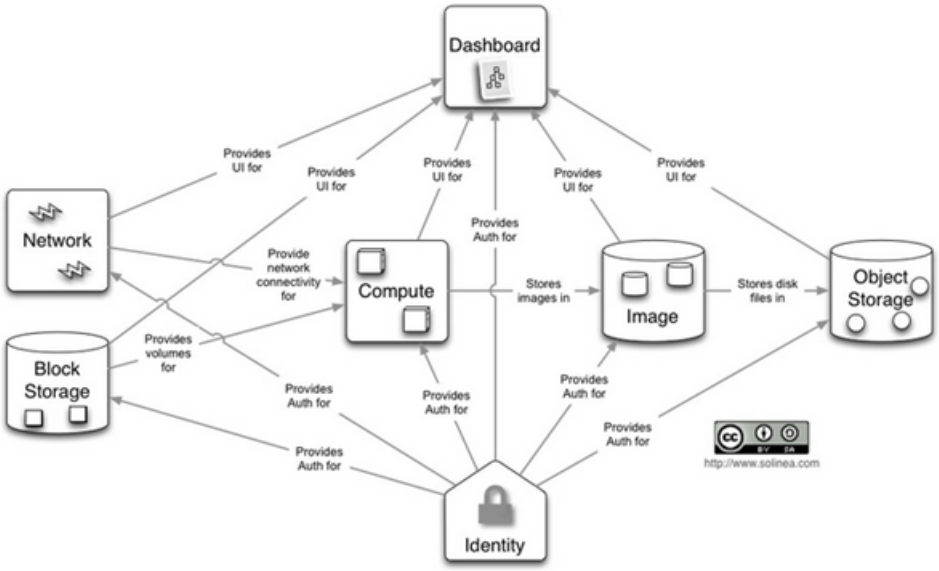

Figure 4.1: OpenStack Conceptual Diagram(*OpenStack Training Guides* (2014))

OpenStack can be used to implement private and public cloud infrastructures. For the specified experiment, OpenStack private cloud at National College of Ireland is used to create a heterogeneous Hadoop cluster. Different set of services provides the desired functionality in OpenStack Cloud. The services offered by OpenStack can be viewed with the help of user interface service (i.e. OpenStack dashboard). Internally, OpenStack dashboard is addressed as Horizon (*OpenStack Training Guides* (2014)).

On the OpenStack dashboard, user can opt for many OpenStack services. One of the crucial service is the management and administration of virtual machines. For this purpose, Nova acts as a main compute system controller and administrator of virtual machines. In addition, Nova also keeps track of all the metadata related to images. Apart from Nova, Cinder is the OpenStack block storage which provides permanent storage volumes for the virtual compute. These volumes can be attached or detached from virtual machine easily (*OpenStack Training Guides* (2014)).

Moreover the Network component also called as Quantum provides the network communications between different virtual machines. Furthermore, Swift acts as an object storage which stores the disk and image files with the help of OpenStack image component (i.e. Glance). To access OpenStack services, users need to authenticate their identity which is achieved with the help of identity mechanism (*OpenStack Training Guides* (2014)).

4.2 Heterogeneous Hadoop cluster setup on OpenStack private cloud

The Hadoop cluster has been set up on the Openstack private cloud of National College of Ireland, Cloud Competency Centre (see Fig.4.2). Openstack private cloud is chosen because of the easy management of Hadoop cluster. Moreover, Openstack Savanna provides easy Graphical User Interface (GUI) to add extra volumes to the machines, which is useful in case of large data.

The Hadoop cluster for this research work was setup using a heterogeneous architecture with one Master and seven Slaves (see Figure. Heterogeneous Hadoop cluster architecture) , where different configurations were used for the nodes as follow:

- The master node has the OpenStack basic A4 configuration.
- Three of the slave nodes are small OpenStack instances.
- Two of the slave nodes are medium OpenStack instances.
- Two of the slave nodes are large OpenStack instances.

Their configuration is explained below in the next subsections.

4.2.1 Heterogeneous Hadoop cluster configuration

The steps followed for the creation of Hadoop cluster are described in the below steps:

- Step 1: Creation of instances in OpenStack private cloud.
- Step 2: Preparing instances ready with Hadoop installation. (i.e. Independent single node Hadoop cluster)
- Step 3: Configuring Hadoop cluster using independent eight single node Hadoop instances.

After installing Hadoop on each instance, the instances have been configured to assign the particular responsibilities such as Namenode, JobTracker, TaskTracker and Datanode (i.e. slaves).

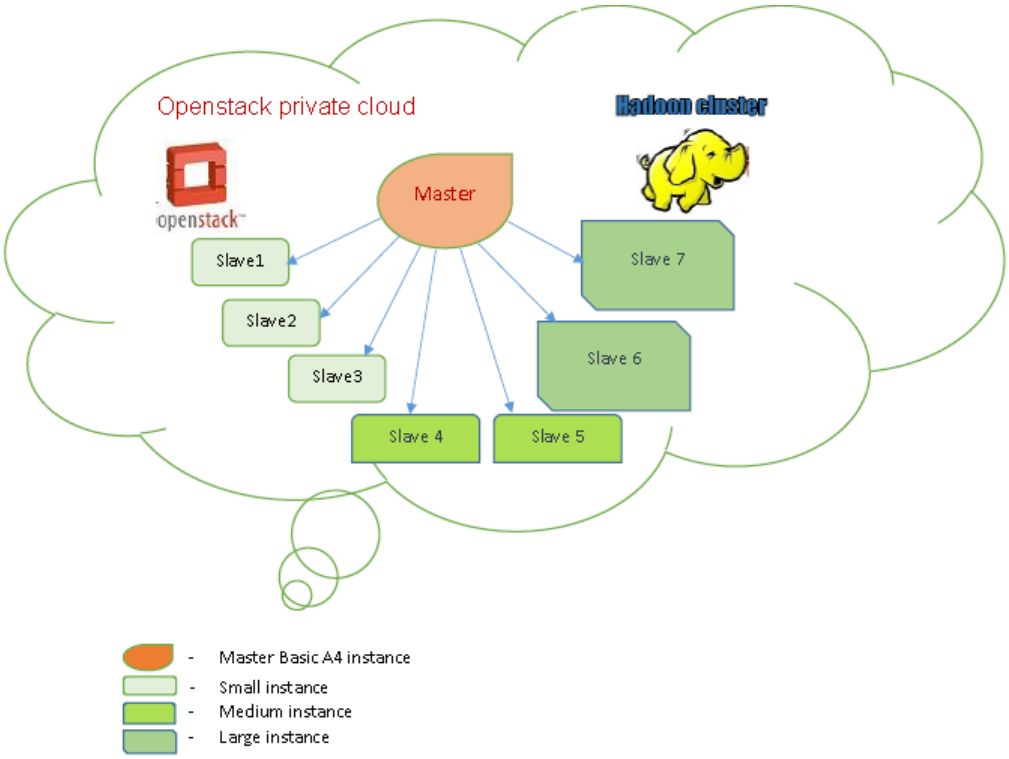

Figure 4.2: Heterogeneous Hadoop cluster architecture on OpenStack private cloud

Step 1: Creation of instances in OpenStack private cloud.

Hadoop cluster setup was achieved by creating eight instances with different configurations in the OpenStack private cloud. The configurations of the instances are:

- Three instances have 2GB RAM, 1CPU and 20 GB hard-disk resources each (i.e. OpenStack small instance called m1.small), where three Hadoop slave nodes were configured;
- Two instances have 4GB RAM, 2 CPU and 40 GB hard-disk each (i.e. OpenStack medium instance called m1.medium), where two Hadoop slaves were configured;
- Two instances have 8GB RAM, 4 CPU and 60 GB hard-disk each (i.e. OpenStack large instance called m1.large), where two Hadoop slaves were configured.
- The master node is with 14 GB RAM, 8 CPU and 20 GB hard-disk (i.e. OpenStack basic A4 instance called Basic A4).

Public IPs (i.e. floating IPs) have been assigned to all eight instances in order to allow the connection to the instances from the remote machines. Moreover, a separate security group and a private key are created for the specified Hadoop cluster. Thus,

the actual IPs are used in the Hadoop cluster formation and all the instances are on the same network (same private cloud). It is important to note that all eight instances are available under the zone Nova in OpenStack private cloud, and that all instances have the same operating system (i.e. Ubuntu 12.04.3).

As described in Section 4.1, Nova is the main compute component which provides compute to all instances for this Hadoop cluster. Usually, Nova-api handles the requests for creation of instances and enforces the created security rules. After the submission of the requests, Nova compute process (i.e. Nova compute daemon) manages the creation and termination of virtual instances with the help of hypervisors APIs. The new volumes can be created and added to existing instances using Nova-volume component. The communication and networking among the instances is administered by Nova-network. The virtual instance requests are stored in the queue. Nova makes use of queue to process the requests and communication between different Nova components. Nova-schedule daemon decides in which server the virtual instance requests will be executed. All the details related to the virtual instances will be stored in the SQL database.

Additionally, two rules were created for the specified security group (see Table.4.1):

Table 4.1: Configuration for cluster nodes

Direction	**IP Protocol**	**Port Range**	**Remote**
Ingress	TCP	22	0.0.0.0/32(CIDR)
Ingress	TCP	22	193.1.209.100/32(CIDR)

The first rule from above table is opening Port 22 in all instances for all machines, it means any machine can connect to these instances remotely using private key.

The second rule from above table is opening port 22 in all instances for my own machine with IP address 193.1.209.100.

The resulting node structure is highlighted in Fig.4.3

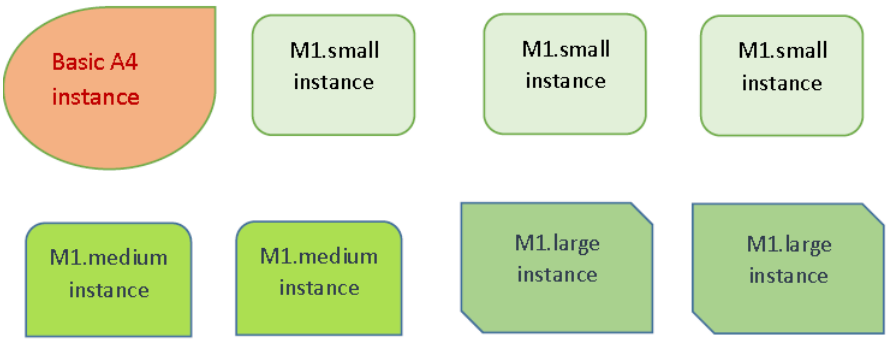

Figure 4.3: OpenStack instances with different configurations

Step 2: Preparing independent Single node Hadoop cluster.

In order to create the Hadoop cluster, the above eight instances should work as independent single-node Hadoop cluster. The similar below process was repeated on each instance to install Hadoop.

First, the necessary software packages for Hadoop such as Python and Java were installed. All the Hadoop files should have separate dedicated username. In this sense, the separate user called "hduser" and the group called "hadoop" were created for installing Hadoop. In addition, Secure Shell (i.e. SSH) key for Hadoop setup was created. SSH key is used to check Hadoop setup which run on localhost. Moreover, the access to localhost has been enabled by copying the public key into the list of authorized keys. The creation of key on each instance is important during formation of cluster. In addition, the IPV6 was not required, so it was disabled, because it might create an issue during cluster formation.

Further, the Hadoop 1.2.1 has been downloaded from https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1.tar.gz, and unzipped under "hadoop" directory. The environmental variable file (i.e. ".bashrc" file) for the "hduser" has been configured for Hadoop environment variable. To store Hadoop temporary files, the separate directory "/app/hadoop/tmp" was created. The Hadoop configuration files were configured in each instance to act as a master node or as a slave node depending on their roles.

After installing the Hadoop on all instances, the Hadoop setup has been checked on each instance by starting Hadoop on localhost. The resulting nodes can be viewed in Fig.4.4:

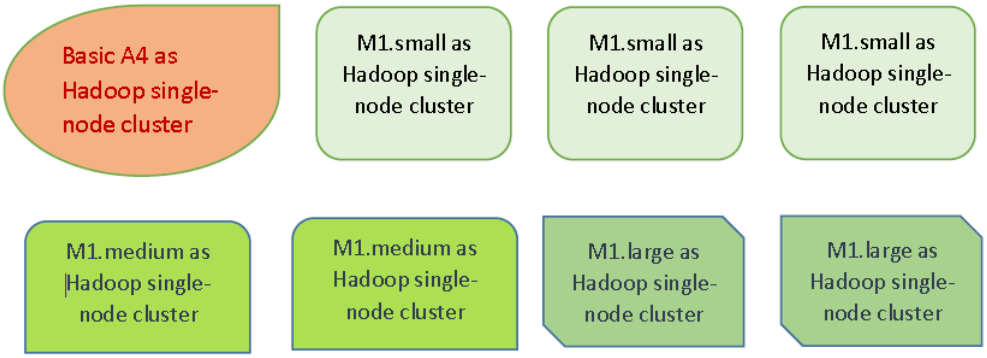

Figure 4.4: OpenStack instances with Hadoop installation working as independent single-node cluster

Step 3: Creation of cluster using eight independent Hadoop nodes.

The instances created in the above step were used to build specified Hadoop cluster. The Hadoop cluster involved a master node and seven slave nodes. The configurations setup for master node and slave nodes are described as below:

- Master node is configured to act as a Namenode and JobTracker. Here, instance Basic A4 is configured as a master node. In "/etc/hosts" file on basic A4 instance, the IP addresses and proposed name of the each instance in Hadoop cluster are mentioned as in the below code:

```
10.1.2.20 master
10.1.2.21 slave1 slave1.localdomain
10.1.2.7 slave2 slave2.localdomain
10.1.2.9 slave3 slave3.localdomain
10.1.2.11 slave4 slave4.localdomain
10.1.2.12 slave5 slave5.localdomain
10.1.2.2 slave6 slave6.localdomain
10.1.2.14 slave7 slave7.localdomain
```

  The Basic A4 instances Hadoop configuration file named "masters" has been edited under "conf" directory to act as master in Basic A4 instance. The other configuration file named "slaves" under the same folder "conf" has been configured with the names of slaves in a cluster.

- On slave nodes, the IP addresses of master and the slave nodes are mentioned in /etc/hosts file as mentioned in master node.

To ease the SSH from master node to slaves (i.e. datanodes), the public key was updated from master node to each slave node using the below command:

```
1 ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slave
```

Configuration files on slave nodes and master node signify the port through which master node is going to communicate with slave nodes. The Hadoop configuration files on each node in cluster including master node and slave nodes have been changed to master ports, so that the master node can distribute data to slave node through the common port and run application through the port dedicated for MapReduce application. The advancement of the MapReduce application can be checked using web interface.

After setting up the configurations of Hadoop cluster, the Hadoop cluster setup is viewed as in Fig.4.5:

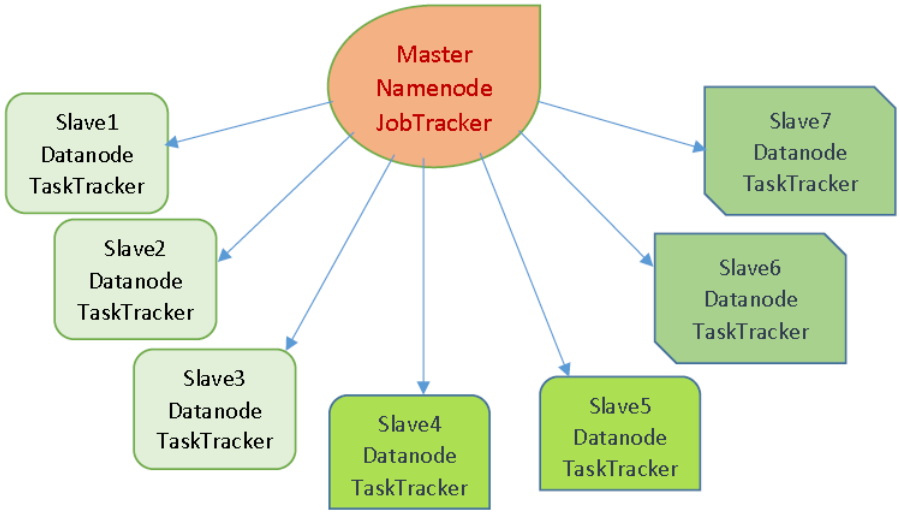

Figure 4.5: Hadoop setup diagram.

According to the Fig.4.5, the Basic A4 instance (i.e. master node) carries the responsibility of a Namenode and a JobTracker. Three small instances are slave1, slave2 and slave3 respectively. Two medium instances are named as slave4 and slave5 respectively and two large instances are called as slave6 and slave7 respectively.

4.3 Speed Analyser

To measure the processing capability (i.e. heterogeneity) of each slave node in the mentioned Hadoop cluster, the Speed Analyser component was implemented. The Speed Analyser was installed and executed on each slave node. The Speed Analyser achieved the functionality of measuring the speed for individual slave nodes using the

following two shell scripts: Install Speed Analyser and Execution of Speed Analyser, which are described below.

4.3.1 Install Speed Analyser

A shell script was created for installing the Speed Analyser agent, which takes a file containing slave node's name as an argument. First, the script makes the connection of the master node to each slave nodes. Further, the script is copying the Matrix multiplication application together with the input files to each individual slave node as in the below code:

```
1
2 for slave in `cat "$HOSTLIST"|sed "s/#.*$//;/^$/d"`; do
3 ...
4 ssh  hduser@$slave "mkdir -p $HOME/speedinstall; chmod 777 *"
5 scp -r hduser@master:$HOME/speedinstall/* hduser@$slave:$HOME/speedinstall
6 ...
```

4.3.2 Execute Speed Analyser

A shell script was created to execute the Speed Analyser. This execution script takes the file containing slave node's names as an argument as in the code below:

```
1 for slave in `cat "$HOSTLIST"|sed "s/#.*$//;/^$/d"`; do
2 ...
3 ssh -q hduser@$slave "cd /home/hduser/speedinstall; touch out log response;(/usr/bin ↩
      /time --format="%E" java MatrixMultiplyRA input.txt out) |& tee log"
4 ...
```

Initially this script executes the Matrix multiplication application using the input files which are available from the Install Speed Analyser script on each slave node. In addition, the script is able to record the response time taken by each slave node to process the Matrix multiplication application. The recording of the response time is stored in a log file on each slave node of the Hadoop cluster.

The computing ratios for the existing nodes were calculated after collecting the responses from each slave node. These responses were written in the file named "computingratio.txt".

4.4 Data-Distribution Component

The Data-Distribution component is based on the Data-Distribution algorithm mentioned in Section 3. The implementation of Data-Distribution component was integrated inside the Namenode under HDFS. The development environment for Hadoop was setup on Namenode for integration. In this case, master node was acting as Namenode.

To achieve the implementation of Data-Distribution component, the source code for HDFS was cloned from git:

```
1 git clone git://git.apache.org/hadoop-hdfs.git
```

To build the downloaded HDFS source code, Maven 3.2.2 was installed on master node. Apache Maven is open-source project management tool. The above cloned HDFS project was built with the help of the below commands:

```
1 mvn install -DskipTests
2 mvn eclipse:eclipse -DdownloadSources=true -DdownloadJavadocs=true
```

After building the source code using Maven, Gooogle's ProtocolBuffers was installed. The instructions from the Apache Hadoop contribution page were followed to install and configure ProtocolBuffers. ProtocolBuffers is an open source project which acknowledge Google's platform-neutral and language neutral interprocess-communication (IPC) and serialization framework. Hadoop uses ProtocolBuffers to communicate between different language platform during the compilation and execution. In addition, the Eclipse Java EE IDE was used as Integrated Development Environment.

All the HDFS source code was imported in the Eclipse IDE for Data-Distribution component implementation. The interaction of the different classes inside Hadoop source code was studied to understand the exact dependency of Namenode's decision-making in terms of the selection of Datanodes for storage. After exploring the source code, it was understood that "FSNamesystem.java" is the class who provides the list of Datanodes to the "DistributedFileSystem.java" class. The dependencies of the proposed Data-Distribution implementation with other current Hadoop classes are highlighted in Fig. 4.6:

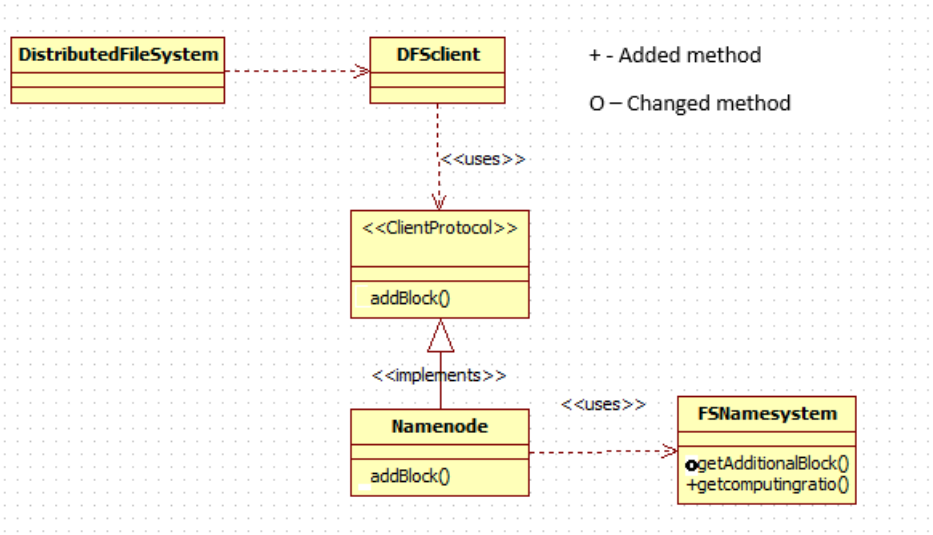

Figure 4.6: Class diagram with the dependencies for the proposed Data-Distribution component with the current Hadoop classes.

As a part of Data-Distribution component implementation, "getcomputingratio()" method was added to the class "FSNamesytem.java". The method is implemented as below:

```
1  public int getcomputingratio(DatanodeSecriptor node)
2   {
3          try{
4          Path pt=new Path("hdfs://master:54310/user/hduser/computingratio.txt");
5          ...
6          while (line != null){
7                    if (words[0] == node.hostname)
8                    {
9                        y = words[1];
10                       return y;
11                   }
12               ...
13         }
14 ...
15  }
```

In the implementation of "getcomputingratio()", the method accepts the "DatanodeDescriptor" class's object as a parameter. Further, function accesses the file "computingratio.txt" to read the computing ratio of each Datanode. The method "getcomputingratio()" matches the hostname of the passed "DatanodeDescriptor" object to the hostname presented in the file "computingratio.txt". At the end, this method returns the number of blocks which is allowed to store on the Datanode parameter.

Furthermore, Namenode keeps track of all the HDFS related operations with the help of class "FSNamesystem.java". Class "FSNamesystem.java" creates the binding of blocks and it's suitable Datanodes. So, this class was selected to implement the decision part of new Data-Distribution technique. The method "getAdditionalBlock()" in "FS-Namesystem.java" class decides the suitable datanodes. As the data blocks are written after sending the list of suitable Datanodes, alteration of data-distribution should be reflected in Namenode.

```
1  DatanodeDescriptor targets[] = replicator.chooseTarget(src,
2                                                          replication,
3                                                          clientNode,
4                                                          excludedNodes,
5                                                          blockSize);
6   if ( oldtargets == null )
7     {
8         oldtargets = targets[0];
9         i = getcomputingratio(targets[0]);
10        i = i -1;
11    }
12    else
13    {
14        if ( i == 0 )
15        {
16        //targets = targets;
17        oldtargets = target[0];
18        i = getcomputingratio(targets[0]);
19        i = i -1;
20        }
21        else
22        {
23        targets[0] = oldtargets;
24        i = i - 1;
25        }
26    }
```

The "getAdditionalBlock()" method asks to the replication policy about the suitable nodes by invoking "chooseTarget()" method. So, here the targets variable will include the list of Datanodes based on the replication factor. In this logic, the same first Datanode will be served to the "DistributeFileSystem.java" class for subsequent "y" blocks, which "getcomputingratio()" method will return to the variable "i".

Chapter 5

Evaluation

To confirm the improvement in the performance of a heterogeneous Hadoop cluster after the implementation of the proposed Data-Distribution technique, two MapReduce applications were executed on the traditional Hadoop system and on the Hadoop system extended with the proposed Data-Distribution Technique.

First, the behavior of the Wordcount MapReduce application is analysed for the traditional Hadoop data-distribution using different data sizes. The response times were obtained as below:

Table 5.1: Response times for Wordcount application on traditional Hadoop

Data-Size	**Real-time**
512 MB	54 secs
1 GB	56 secs
2 GB	92secs
4 GB	143 secs
8 GB	265 secs

For the same set of data, the response times were recorded for the proposed Data-Distribution Technique using the Wordcount MapReduce application. The improved performance was shown by the new Data-Distribution Technique.

Table 5.2: Response times for Wordcount application using proposed solution

Data-Size	Real-time
512 MB	52 secs
1 GB	54 secs
2 GB	91 secs
4 GB	137 secs
8 GB	258 secs

Above two sets of response times are graphically represented as below Fig.5.1

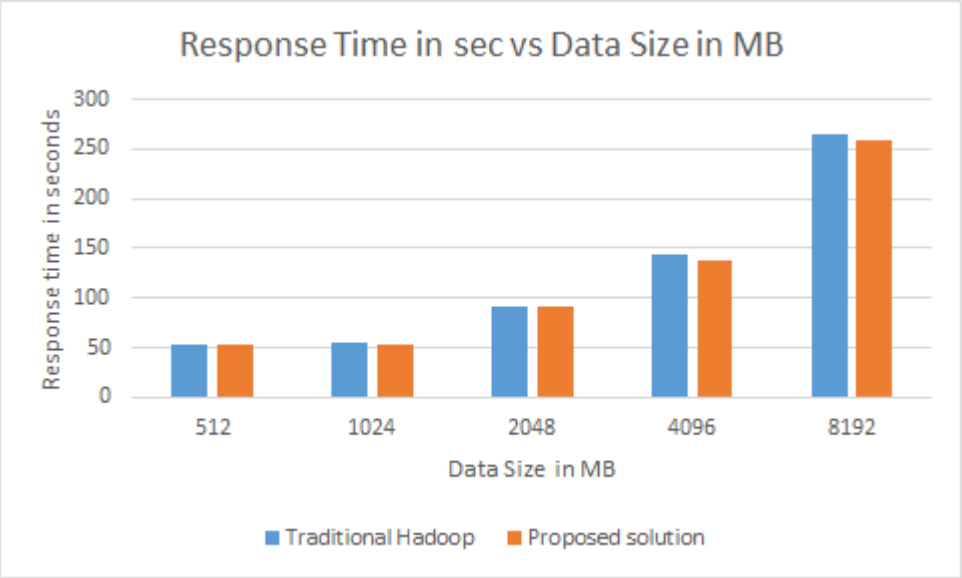

Figure 5.1: Comparison between the traditional Hadoop and the proposed solution while running the Wordcount application.

Fig.5.1 presents the response times taken by Wordcount application for each data size ranging from 512 MB to 8192 MB. As per the graph, the performance times of Wordcound MapReduce application is approximately similar for the initial set of data which were 512 MB, 1024 MB and 2048 MB. On the other hand, when the data size increases (i.e. 4096MB and 8192MB), the proposed solution signifies the improvement in the response time.

Similarly, the Grep MapReduce application is executed on the traditional Hadoop system. The response times are presented in the Table.5.3 below:

Table 5.3: Response times for Grep application on traditional Hadoop

Data-Size	**Real-time**
512 MB	117 secs
1 GB	129 secs
2 GB	227 secs
4 GB	387 secs
8 GB	706 secs

Likewise, the Grep MapReduce application was executed on the Hadoop system with proposed solution for the same data sizes.

The response times recorded are mentioned in below Table.5.4:

Table 5.4: Response times for Grep application using proposed solution

Data-Size	**Real-time**
512 MB	121 secs
1 GB	125 secs
2 GB	220 secs
4 GB	357 secs
8 GB	691 secs

For comparing the behaviour of both the traditional Hadoop system and Hadoop system with proposed solution during the execution of Grep application, the Fig.5.2 is described:

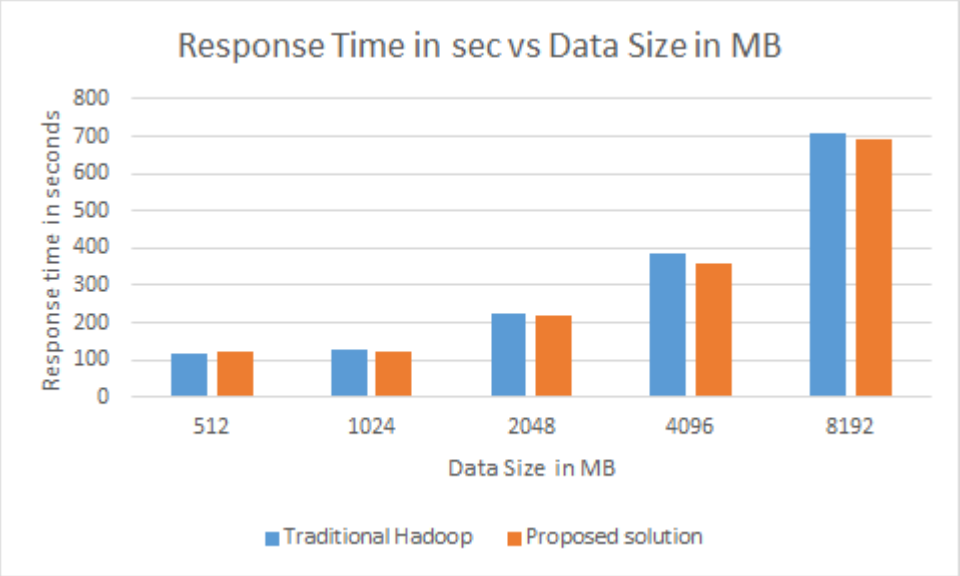

Figure 5.2: Comparison between the traditional Hadoop and the proposed solution while running the Grep application.

In Fig.5.2, it is understood that for the initial set of datasizes 512 MB, 1024 MB and 2048 MB were taking similar time to complete Grep application. On the contrary, when the data size crosses 4GB, the response time taken by the Hadoop system with proposed solution was less than the traditional Hadoop system. So, it is assured that the proposed Data-Distribution is showing improvement in Hadoop performance. The maximum data size was limited to 8 GB because of the storage constraint in the OpenStack instance that was created for representing the Master node. Thus, the evaluation could not be accomplished for bigger data size.

After discussing the execution of Wordcount MapReduce application and Grep application on the Hadoop system with proposed solution, it can be seen that there are certain improvement in the performance because of proposed solution.

Chapter 6

Conclusion

This paper proposed a solution to improve the performance of the heterogeneous Hadoop cluster by assigning the data blocks on the basis of processing speed of the Datanodes. The traditional Hadoop system assigns the same set of data to each Datanode, which may not be fruitful for heterogeneous Hadoop cluster environment. The proposed solution includes Speed Analyser and Data-Distribution component. Speed Analyser component measure the processing speed of each Datanode in terms of computing ratio. The Data-Distribution component is integrated as a part of Hadoop software and it assigns data blocks to Datanodes based on their computing ratio.

To verify the advancements in the response time of Hadoop system with proposed solution, two MapReduce applications were tested. The evaluation section describes outcomes of the testing. Using the proposed solution, there are improvements in the heterogeneous Hadoop clusters performance when the data size increases.

Thus, the proposed solution has the advantage of understanding Datanode processing speed and Data-Distribution Technique which assigns blocks as per computing capacity of Datanodes.

Although, there are certain improvement in the performance because of proposed solution, the improvements are not so much significant. One of the probable reason for this behavior might be the data size. In addition, even after assignment of data blocks to Datanodes based on their processing speed, some Datanodes might have struggled during execution period. However, this behavior will be investigated as part of the future work. In addition, the effect of replication factor on the proposed solution is also not considered on the implementation of the proposed solution. Thus, the proposed solution can be improved by handling a straggler node issue and understanding the impact of replication over the proposed solution.

Bibliography

Arasanal, R. & Rumani, D. (2013), Improving mapreduce performance through complexity and performance based data placement in heterogeneous hadoop clusters, *in* C. Hota & P. Srimani, eds, 'Distributed Computing and Internet Technology', Vol. 7753 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 115–125. [Online]. Available from: http://dx.doi.org/10.1007/978-3-642-36071-8_8 [Accessed 22 February 2014].

Cardosa, M., Wang, C., Nangia, A., Chandra, A. & Weissman, J. (2011), Exploring mapreduce efficiency with highly-distributed data, *in* 'Proceedings of the Second International Workshop on MapReduce and Its Applications', MapReduce '11, ACM, New York, NY, USA, pp. 27–34. [Online]. Available from: http://doi.acm.org/10.1145/1996092.1996100 [Accessed 9 December 2013].

Chervenak, A., Deelman, E., Livny, M., Su, M.-H., Schuler, R., Bharathi, S., Mehta, G. & Vahi, K. (2007), Data placement for scientific applications in distributed environments, *in* 'Proceedings of the 8th IEEE/ACM International Conference on Grid Computing', GRID '07, IEEE Computer Society, Austin, Texas, pp. 267–274. [Online]. Available from: http://dx.doi.org/10.1109/GRID.2007.4354142 [Accessed 5 December 2013].

Chung, F., Graham, R., Bhagwan, R., Savage, S. & Voelker, G. M. (2006), 'Maximizing data locality in distributed systems', *J. Comput. Syst. Sci.* **72**(8), 1309–1316. [Online]. Available from: http://dx.doi.org/10.1016/j.jcss.2006.07.001 [Accessed 7 December 2013].

Dean, J. & Ghemawat, S. (2008), 'Mapreduce: Simplified data processing on large clusters', *Commun. ACM* **51**(1), 107–113. [Online]. Available from: http://doi.acm.org/10.1145/1327452.1327492 [Accessed 7 December 2013].

Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A. & McPherson, J. (2011), 'Cohadoop: Flexible data placement and its exploitation in hadoop', *Proc. VLDB Endow.* **4**(9), 575–585. [Online]. Available from: http://dl.acm.org/citation.cfm?id=2002938.2002943 [Accessed 7 December 2013].

Fan, Y., Wu, W., Cao, H., Zhu, H., Zhao, X. & Wei, W. (2012), A heterogeneity-aware data distribution and rebalance method in hadoop cluster, *in* 'ChinaGrid Annual Conference (ChinaGrid), 2012 Seventh', Beijing, pp. 176–181. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6337296&isnumber=6337273 [Accessed 6 December 2013].

Guo, Z., Fox, G. & Zhou, M. (2012), Investigation of data locality and fairness in mapreduce, *in* 'Proceedings of Third International Workshop on MapReduce and Its Applications Date', MapReduce '12, ACM, New York, NY, USA, pp. 25–32. [Online]. Available from: http://doi.acm.org/10.1145/2287016.2287022 [Accessed 5 December 2013].

HDFS Architecture Guide (2008). [Online].Available from:http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html [Accessed 6 July 2014].

Jin, J., Luo, J., Song, A., Dong, F. & Xiong, R. (2011), Bar: An efficient data locality driven task scheduling algorithm for cloud computing, *in* 'Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing', CCGRID '11, IEEE Computer Society, Newport Beach, CA, pp. 295–304. [Online]. Available from: http://dx.doi.org/10.1109/CCGrid.2011.55 [Accessed 5 December 2013].

Kala Karun, A. & Chitharanjan, K. (2013), A review on hadoop ; hdfs infrastructure extensions, *in* 'Information Communication Technologies (ICT), 2013 IEEE Conference on', JeJu Island, pp. 132–137. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6558077&isnumber=6558050 [Accessed 7 December 2013].

Kurazumi, S., Tsumura, T., Saito, S. & Matsuo, H. (2012), Dynamic processing slots scheduling for i/o intensive jobs of hadoop mapreduce, *in* 'Networking and Computing (ICNC), 2012 Third International Conference on', Okinawa, pp. 288–292. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6424579&isnumber=6424528 [Accessed 7 December 2013].

Nguyen, P., Simon, T., Halem, M., Chapman, D. & Le, Q. (2012), A hybrid scheduling algorithm for data intensive workloads in a mapreduce environment, *in* 'Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on', Chicago, IL, pp. 161–167. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6424941&isnumber=6424908 [Accessed 6 December 2013].

OpenStack Training Guides (2014). [Online].Available from:http://docs.openstack.org/training-guides/training-guides.pdf [Accessed 6 April 2014].

Shen, H. & Zhu, Y. (2009), 'A proactive low-overhead file replication scheme for structured {P2P} content delivery networks', *Journal of Parallel and Distributed Computing* **69**(5), 429 – 440. [Online]. Available from: http://www.sciencedirect.com/science/article/pii/S0743731509000264 [Accessed 3 December 2013].

Welcome to Apache Hadoop! (2008). [Online].Available from:http://hadoop.apache.org/ [Accessed 5 April 2014].

White, T. (2012), *Hadoop: The Definitive Guide*, O'Reilly Media, Inc.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A. & Qin, X. (2010), Improving mapreduce performance through data placement in heterogeneous hadoop clusters, *in* 'Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on', Atlanta, GA, pp. 1–9. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5470880&isnumber=5470678 [Accessed 2 December 2013].

Ye, X., Huang, M., Zhu, D. & Xu, P. (2012), A novel blocks placement strategy for hadoop, *in* 'Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on', Shanghai, pp. 3–7. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6211069 [Accessed 20 February 2014].

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S. & Stoica, I. (2010), Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling, *in* 'Proceedings of the 5th European Conference on Computer Systems', EuroSys '10, ACM, New York,

NY, USA, pp. 265–278. [Online]. Available from: http://doi.acm.org/10.1145/1755913.1755940 [Accessed 4 December 2013].

Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. & Stoica, I. (2008), Improving mapreduce performance in heterogeneous environments, *in* 'Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation', OSDI'08, USENIX Association, Berkeley, CA, USA, pp. 29–42. [Online]. Available from: http://dl.acm.org/citation.cfm?id=1855741.1855744 [Accessed 6 December 2013].

Zhang, X., Feng, Y., Feng, S., Fan, J. & Ming, Z. (2011), An effective data locality aware task scheduling method for mapreduce framework in heterogeneous environments, *in* 'Proceedings of the 2011 International Conference on Cloud and Service Computing', CSC '11, IEEE Computer Society, Hong Kong, pp. 235–242. [Online]. Available from: http://dx.doi.org/10.1109/CSC.2011.6138527 [Accessed 5 December 2013].

Zhao, Y., Wang, W., Meng, D., Yang, X., Zhang, S., Li, J. & Guan, G. (2012), A data locality optimization algorithm for large-scale data processing in hadoop, *in* 'Computers and Communications (ISCC), 2012 IEEE Symposium on', Cappadocia, pp. 000655–000661. [Online]. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6249372&isnumber=6249257 [Accessed 6 December 2013].

Appendix A

Appendix

A.1 Speed Analyser component

A.1.1 Install script

```
1  usage="Usage: slaves.sh [--config confdir] command..."
2
3  # if no args specified, show usage
4  if [ $# -le 0 ]; then
5    echo $usage
6    exit 1
7  fi
8
9
10 bin=‘dirname "$0"‘
11 bin=‘cd "$bin"; pwd‘
12
13 # If the slaves file is specified in the command line,
14 # then it takes precedence over the definition in
15 # hadoop-env.sh. Save it here.
16 HOSTLIST=$1
17
18 #if [ -e "$bin/../libexec/hadoop-config.sh" ]; then
19 #  . "$bin"/../libexec/hadoop-config.sh
20 #else
21 #  . "$bin/hadoop-config.sh"
22 #fi
23
24
25 if [ "$HOSTLIST" = "" ]; then
26   if [ "$HADOOP_SLAVES" = "" ]; then
```

```
    export HOSTLIST="${HADOOP_CONF_DIR}/slaves"
  fi
fi

for slave in `cat "$HOSTLIST"|sed "s/#.*$//;/^$/d"`; do
if ssh hduser@$slave stat $HOME/speedinstall/rational.jar \> /dev/null 2\>\&1
then
echo "SpeedAnalyser is already installed on $slave..."
else
ssh  hduser@$slave "mkdir -p $HOME/speedinstall; chmod 777 *"
scp -r hduser@master:$HOME/speedinstall/* hduser@$slave:$HOME/speedinstall
fi
done

wait
```

A.1.2 Execute script

```
usage="Usage: slaves.sh [--config confdir] command..."

# if no args specified, show usage
if [ $# -le 0 ]; then
  echo $usage
  exit 1
fi


bin=`dirname "$0"`
bin=`cd "$bin"; pwd`

HOSTLIST=$1


speed="/home/hduser/speedinstall"
if [ "$HOSTLIST" = "" ]; then
  if [ "$HADOOP_SLAVES" = "" ]; then
    export HOSTLIST="${HADOOP_CONF_DIR}/slaves"
  fi
fi

for slave in `cat "$HOSTLIST"|sed "s/#.*$//;/^$/d"`; do
if ssh hduser@$slave stat $HOME/speedinstall \> /dev/null 2\>\&1
then
if ssh hduser@$slave stat $HOME/speedinstall/log \> /dev/null 2\>\&1
then
ssh -q hduser@$slave "cd /home/hduser/speedinstall; rm log"
```

```
fi
files=‘ssh hduser@$slave ’ls -l $HOME/speedinstall | wc -l’‘
if [ $files -ge 5 ]
then
ssh -q hduser@$slave "cd /home/hduser/speedinstall; touch out log response;(/usr/bin ↩
    /time --format="%E" java MatrixMultiplyRA input.txt out) |& tee log"
ssh -q hduser@$slave "scp -r hduser@$slave:/home/hduser/speedinstall/log  ↩
    hduser@master:/home/hduser/speedinstall/log"
fi
else
  echo "Install Speed Analyser on $slave......................."
fi
done

wait
```

A.2 Data-Distribution component

```
DatanodeDescriptor targets[] = replicator.chooseTarget(src,
                                                       replication,
                                                       clientNode,
                                                       excludedNodes,
                                                       blockSize);
  if ( oldtargets == null )
    {
        oldtargets = targets[0];
        i = getcomputingratio(targets[0]);
        i = i -1;
    }
    else
    {
        if ( i == 0 )
        {
        //targets = targets;
        oldtargets = target[0];
        i = getcomputingratio(targets[0]);
        i = i -1;
        }
        else
        {
        targets[0] = oldtargets;
        i = i - 1;
        }
    }
```