Simulator for Big Data Processing Using MapReduce and HDFS

Final Year Project - BSc in Computer Science

Student: Ming Ma

Student Number: 116108056

Supervisor: Dr. John Herbert

Date: April 2018

Department of Computer Science

University College Cork

Declaration of Originality

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

·This is all my own work, unless clearly indicated otherwise, with full and proper accreditation;

·With respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;

·With respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed:. . . . .

Date: . . . .

Acknowledgements

I would like to extend my deepest gratitude to my supervisor, Dr. John Herbert, for providing me with the opportunity to work on a project in which I have great interest and for his continued support, guidance and professionalism throughout the college year.

I would also like to thank Dr. Dapeng Dong for his technical help, opinions shared and advice provided throughout this project.

Abstract

Big data are being increasingly developed with higher requirements on scalability and reliability. On this occasion, many frameworks and architectures are published to provide solutions on big data, Hadoop is one of the most widely used one presented by Apache. HDFS (Hadoop Distributed File System) and MapReduce are two core components in Hadoop system, being responsible for file storage and data processing respectively. This report will give a brief introduction on the ideas of HDFS and MapReduce, and the development of a simulator tool to analyze Hadoop processing performance mainly based on there two key modules.

Contents

Chapter 1

Introduction

1.1 The Problem

As more and more applications and services needs big data processing, which requires a distributed model. Big data is substituting traditional methods on both data storage and process, aiming to aggregate available all resources and physical infrastructures to provide reliable services any time and any where. As of now, numerous data is being uploaded or accessed through Internet by billions of laptops, smartphones, tablet computers and many other types of computers.

Virtualization is a significant technique. It focuses on transforming utilities, such as servers, memories and hard disks to a virtual form, which can be easier managed and accessed, and are not limited by differ platforms and locations.

With the amount of data grows rapidly, traditional structures for storing data are not able to satisfy the requirements any more. Therefore, the idea of distributed storage are put forward. It is a distributed system where data are stored on more than one node rather than a single machine, usually with several replications.

Another core problem in a distributed system is distributed computation. Normally the idea is to divide a complicated task, which may require too strong capability of computation to process on a single machine, into more than one simpler tasks. The final computation result is generated from multiple machines, which each of them process one partition of the task.

1.2 Relevant Techniques

Hadoop is one of the most widely used distributed computation framework to deal with big data in cloud computing. Hadoop uses a layered architecture where HDFS in its lower level to provide reliable data storage, while MapReduce, the mentioned computation model working in the higher level to process huge data volumes.

Not like the traditional hierarchical file storage, HDFS is based on a certain nodes' structure. It includes one and only NameNode, providing metadata services inside HDFS system, and multiple DataNode, providing data storing blocks. All data is divided into blocks and put into DataNodes, with a certain number of replications on other DataNodes. When a operation on a data partition is requested, every DataNodes with the data block is accessible. MapReduce is the key data processing component based on the conceptions of "Map" and "Reduce". Mapper splits a task and deploys on different nodes to process, when workers complete the local task, the results would be returned to Reducer.

1.3 Project Goal

This project aims to create a simulator for predicting performance of applications running on a distributed environment. It follows the architecture of Hadoop framework, simulating two core components, HDFS and MapReduce. It shows how HDFS deals with an input file, dividing them into data blocks, and storing to nodes with a certain number of replications. It also simulates main steps of MapReduce when processing an application with a known time complexity.

The simulator should take information about input files, basically their sizes, as an input, as well as a model, which reflects the relationship between performance and problem size. The model is built up from a set of results of a sample program. Some factors can be defined by users to provide more accurate results in different cases.

The simulator can then give a predicted performance, indicating how long time it may cost to process the data in a Hadoop system. It shows the trend of the changes of performance along with the increase of the input sizes. The simulator is also able to give the information about the configuration that is most likely to produce the best performance if the input is fixed.

1.4 Language

Java is the chosen language to develop the application. Some reasons include it's the same language as the implementation language of Hadoop and some personal advantages as I have done some projects mainly depending on Java and relevant techniques.

Chapter 2

Analysis

In order to design and implement a system that would address the problem correctly, analysis on current solutions and their positive and negative points was important. This section includes the analysis on the objective of the project and some relevant context.

2.1 Big Data

Big data is data sets that are so voluminous and complex that traditional data processing application software are inadequate to deal with them. Big data refers information that requires new processing patterns to suit its rapid growth. [1]

Big data can be mainly described by the following characteristics:

Volume: the quantity of generated and stored data. Generally the quantity of a big data set is in TB level, but in actual use cases, enterprises users generate many data sets together, which makes it reach PB level.

Variety: the type and nature of data. Data comes from many sources, and its variety has broken the traditional limited scopes, containing semi-structured and non-structured data.

Velocity: the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development.

Variability: inconsistency of the data set can hamper processes to handle and manage it.

Veracity: the data quality of captured data can vary greatly, affecting the accurate analysis. With the increasing interest in new data sources, such as social data, enterprise content, business trades, and application data, enterprises need more effective information to guarantee its veracity and security. [2]

2.2 Challenge

Big data can not be managed or analyzed with traditional technological flows or tools. It refers data sets which beyond the normal scope and size, which requires users using a non-traditional methods. The biggest challenge is which methods can use and manage data in a more effective way.

2.3 Big Data Analysis

It is well known that big data is not simply with big volume, and analysis on big data is more important. Through big data analysis, more intelligent, deep, and valuable information can be acquired. More and more applications are relevant to big data, and the characteristics are presenting an increasing complexity. Therefore, big data analysis can be significant to determine whether the acquired information is valuable.

Some basic aspects of big data analysis includes analytic visualizations, data mining algorithms, predictive analytic capabilities, semantic engines, and data quality and master data management.

2.4 Distributed computing

Distributed computing, in contrast to centralized computing, is a method to process data based on networked computers. It focuses on how to divide a big problem which needs huge computing capabilities into small parts, then distributes them to separate computers for processing and gathering the results.

A distributed system may have a common goal, such as solving a large computational problem; the user then perceives the collection of autonomous processors as a unit. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users. [3]

Other typical properties of distributed systems include the following:

The system has to tolerate failures in individual computers.

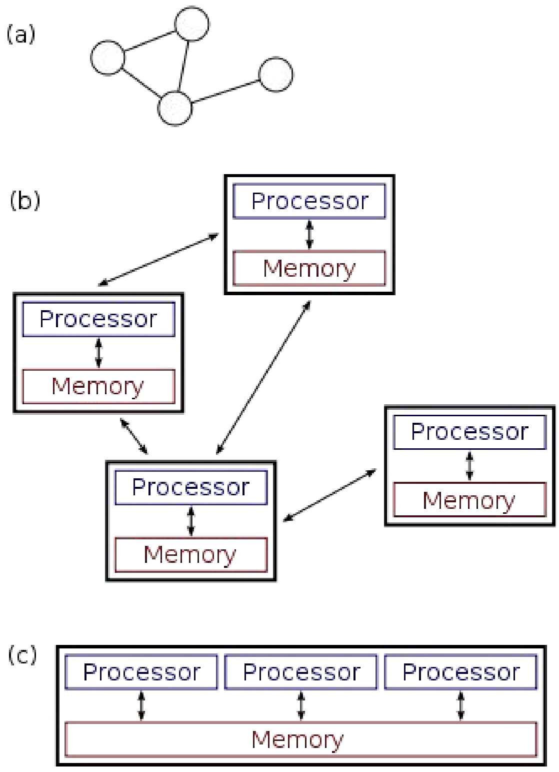
The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.

Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.

2.4.1 Parallel and distributed computing

In parallel computing, all processors may have access to a shared memory to exchange information between processors. [4]

In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.



(a), (b): a distributed system
(c): a parallel system

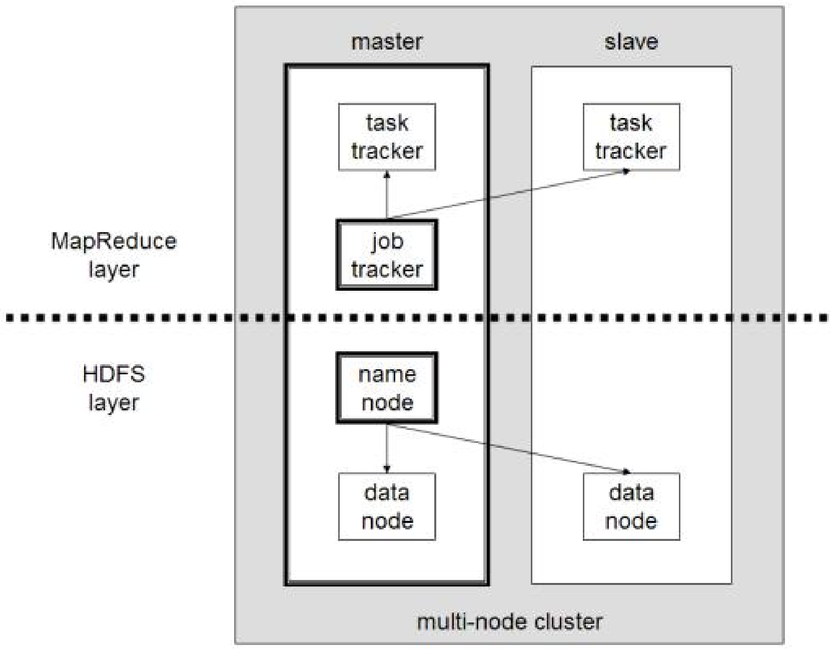
2.5 Hadoop

Apache Hadoop is an open-source software framework used for distributed storage and processing of data sets of big data. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework. [5]

The core modules of Hadoop consists Hadoop Distributed File System (HDFS), a distributed storage part, and MapReduce programming model, a processing part.

2.5.1 Architecture

A Hadoop small cluster includes one master and multiple worker nodes. The master node consists of a Job Tacker, Task Tracker, NameNode, and DataNode. A slave node acts as both a DataNode and Task Tracker. [6]



A multi-node cluster

In a larger cluster, HDFS nodes are managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the NameNode's memory structures, thereby preventing file-system corruption and loss of data. Similarly, a standalone Job Tracker server can manage job scheduling across nodes. When Hadoop MapReduce is used with an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents. [7]

2.5.2 HDFS

Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

HDFS are designed to meet the requirements including:

Hardware Failure: an HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

Streaming Data Access: applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users.

Large Data Sets: applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

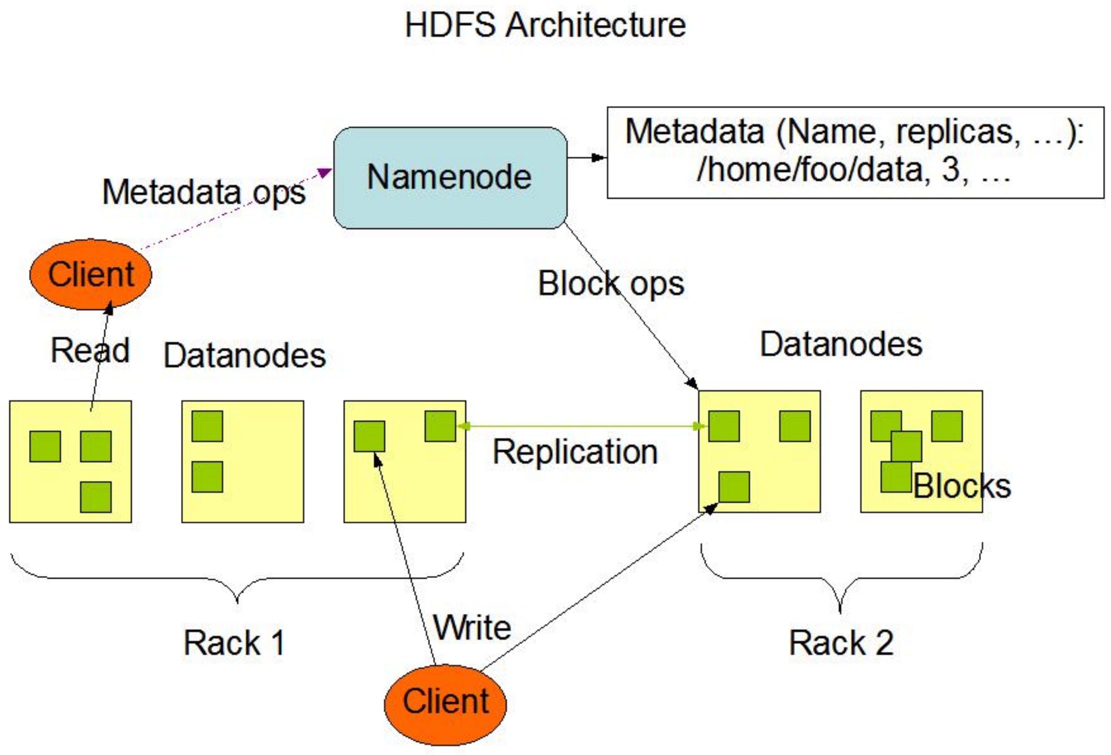
Simple Coherency Model: HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.
HDFS has a master/slave architecture. An HDFS cluster consists of:

A single NameNode: a master server that manages the file system namespace and regulates access to files by clients.

A number of DataNodes: manage storage attached to the nodes that they run on.

HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



The NameNode and DataNode are pieces of software designed to run on commodity machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. [8]

The File System Namespace

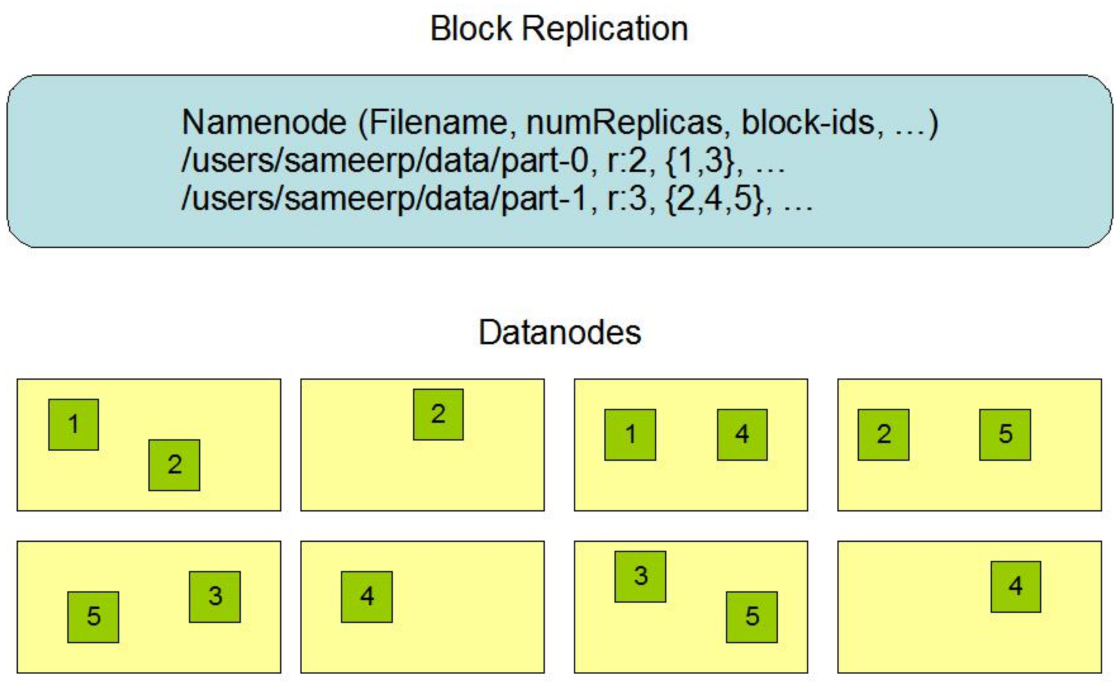
HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file.

The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode.

Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.


Block Replication
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

To minimize global bandwidth consumption and read latency, HDFS tries to satisfy a read request from a replica that is closest to the reader. If there exists a replica on the same rack as the reader node, then that replica is preferred to satisfy the read request.

Data Blocks

HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more times and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 64 MB. Thus, an HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode. [8]

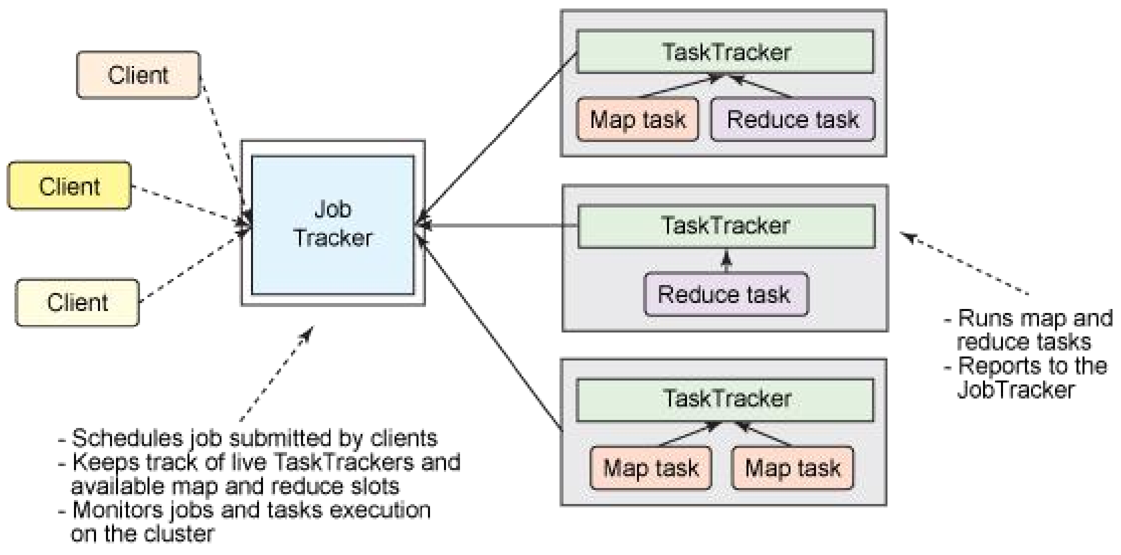
2.5.3 MapReduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data in-parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and HDFS are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master. [9]


Client
Client
Client
Job Tracker
TaskTracker
Map task
Reduce task
TaskTracker
Reduce task
TaskTracker
Map task
Map task
- Runs map and reduce tasks
- Reports to the JobTracker
- Schedules job submitted by clients
- Keeps track of live TaskTrackers and available map and reduce slots
- Monitors jobs and tasks execution on the cluster

Map-Reduce programs transform lists of input data elements into lists of output data elements. A Map-Reduce program will do this twice, using two different list processing idioms, map and reduce.

In between Map and Reduce, there is small phase called Shuffle and Sort in MapReduce.

Mapper writes the output to the local disk of the machine it is working. This is the temporary data. An output of mapper is also called intermediate output. All mappers are writing the output to the local disk. As First mapper finishes, data (output of the mapper) is traveling from mapper node to reducer node. Hence, this movement of output from mapper node to reducer node is called shuffle.

Reducer is also deployed on any one of the datanode only. An output from all the mappers goes to the reducer. All these outputs from different mappers are merged to form input for the reducer. This input is also on local disk. Reducer is another processor where you can write custom business logic. It is the second stage of the processing. Usually to reducer we write aggregation, summation etc. type of functionalities. Hence, Reducer gives the final output which it writes on HDFS.

2.6 Hadoop based-software

The Hadoop community is fast evolving to include businesses that offer support, rent time on managed clusters, build sophisticated enhancements to the open source core, or add their own tools to the mix. Some tools and code can run together under the collective heading “Hadoop”.

Ambari
The Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its APIs. [10]

Cloudera
Cloudera, the commercial Hadoop company, develops and distributes Hadoop, the open source software that powers the data processing engines of the world's largest and most popular web sites.

Founded by leading experts on big data from Facebook, Google, Oracle and Yahoo, Cloudera's mission is to bring the power of Hadoop, MapReduce, and distributed storage to companies of all sizes in the enterprise, Internet and government sectors. [11]

GIS tools for Hadoop
The GIS Tools for Hadoop are a collection of GIS tools that leverage the Spatial Framework for Hadoop for spatial analysis of big data. The tools make use of the Geoprocessing Tools for Hadoop toolbox, to provide access to the Hadoop system from the ArcGIS Geoprocessing environment. [12]

Chapter 3

Design

3.1 Objective

While the goal is to analyse the performance of Hadoop in different situations and configuration parameters, this will be done through developing a simulator based on Hadoop architecture. But the whole system is too complex and hard to simulate in all details, the simulator only focuses on the main modules in Hadoop architecture and some secondary and irregular factors are not taken into account, like hardware and network exceptions, though they could influence system performance in actual cases.

3.2 Chosen Development Method

Having studied common development models for design of applications and products in software engineering, the decision was made to use an Iterative and Incremental Development Model for this simulator. This decision was made to ensure the designed system would not be too complex to implement and consider too many secondary factors.

3.3 Architecture

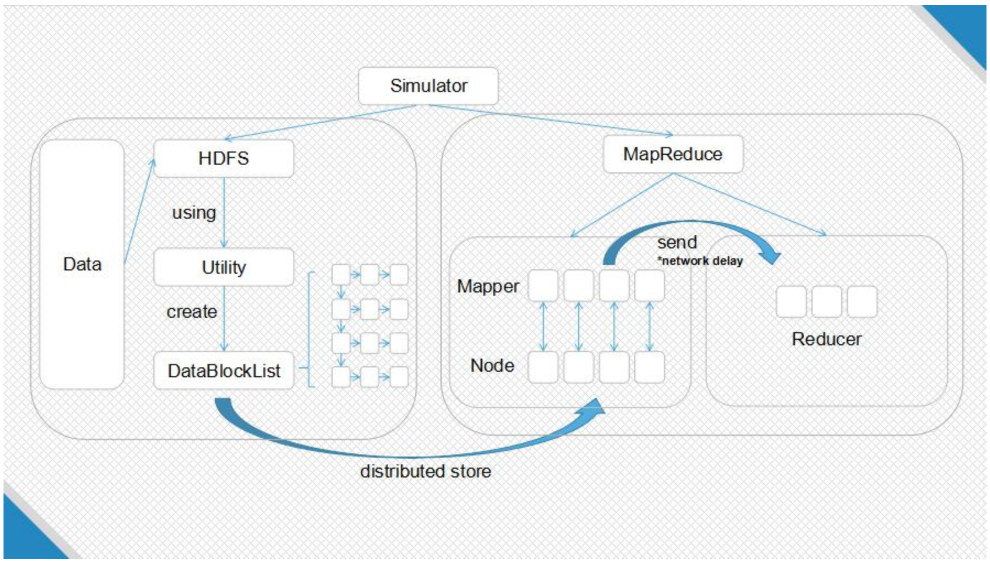
The simulator focuses on performance analysis of two core modules in Hadoop architecture, HDFS and MapReduce. The design planned would have several components based on the different functions provided by the two modules. At the first stage these components were designed separately for the functions in each module, and this is advantageous for future feature additions and the improvement and maintenance.

The design also indicates the interactions between the two core modules.

Recording all the messages changed would be useful for the performance analysis and presentation.

Servers are designed to simulate nodes in a Hadoop cluster, along with some types of physical resources. These resources could be influential to the performance of the system, and designing them provides the flexibility to extend the function of the simulator for putting more factors or different points of focus when considering the performance analysis.

The simulator is intended to give configuration parameters with the most possibilities to reach the best performance on the provided data volume. To design this, the information of the processed data is defined in an input file. HDFS should be able to retrieve the file and then read the information into the system.



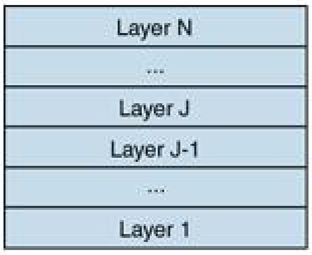
The overall architecture of the simulator

3.3.1 N-tier Model

A layered architecture, also known as a n-tier architecture is considered in this project to provide a more clear logical connection between different modules and components. In this model, components are separated into several layers. The components in each layer should be cohesive and at roughly the same level of abstraction. Each layer should be loosely coupled to the layers underneath.

Pattern-Oriented Software Architecture, Vol 1 [Buschmann96] describes the layering process as follows: [13]

Start at the lowest level of abstraction - call it Layer 1. This is the base of your system. Work your way up the abstraction ladder by putting Layer J on top of Layer J-1 until you reach the top level of functionality - call it Layer N.

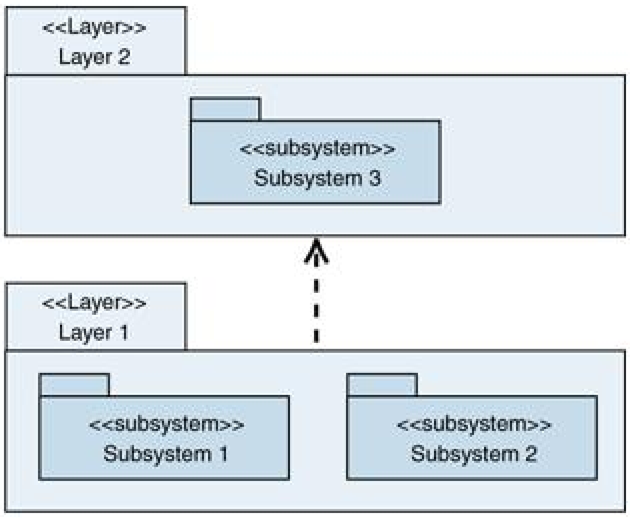


How the layering scheme would look

An important advantage of n-tier architecture is that components in each layer can be managed independently. This helps to analyze the influences of HDFS and MapReduce separately and makes it easier to reach the data flows in each steps, including the submitted application and physical resources.

Components in one layer can interact only with peers in the same level or components from lower levels. This is designed to reduce the dependencies between components on different levels.

Hadoop has a large number of components at some certain levels of abstraction that are not cohesive. In this kind of case, each layer may be further decomposed into one or more cohesive subsystems. [14]



3.4 Experiment

For the aim of predicting computing times on each nodes accurately, experiments based on a simple Java program are done before any operation and initiation on the simulator.

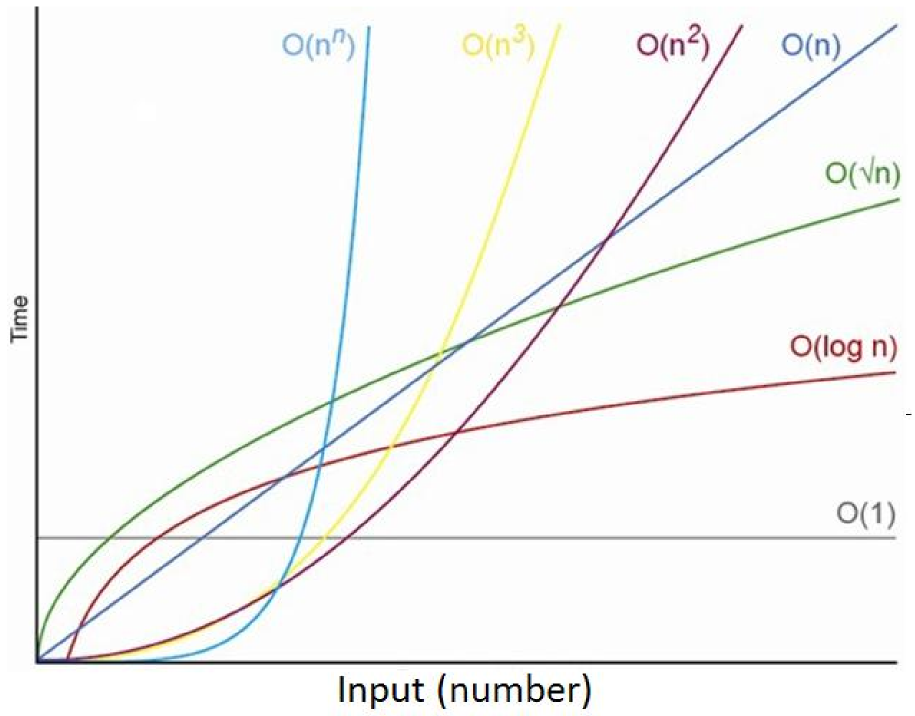
The experiments are chosen as a Word Count application, which retrieves text information in the input file in time complexity of O(n). To record the running time and size of the input file, a model can be built up according to a set of experiment results of the same application sample but different input sizes. The model reflects the relationship between sizes and performances, described by the running times generated from Java programs.

The experiment provides mathematic basic to the simulator, by outputing a model which would be used as an input factor when the simulator initiating in the next step. Once the model is established, it can predict the performance based on any input size. The predicted result is considered to be the performance on a single node.

3.4.1 WordCount application

A WordCount is usually used as a testing sample in Hadoop. It has the time complexity of O(n), which is actually a linear relationship.

A linear function is easy for reference when consider the predictions of the performances with huge input data sets. It is also easy to build a linear model only by a few experiments on testing data sets, though there would be errors.



The time complexity is the computational complexity that measures or estimates the time taken for running an algorithm. An algorithm is said to take linear time, or O(n) time, if its time complexity is O(n). Informally, this means that for large enough input sizes the running time increases linearly with the size of the input.

Linear time is the best possible time complexity in situations where the algorithm has to sequentially read its entire input. Any application with O(n) time complexity can be predicted properly by a simulator taken the model from a WordCount experiment, because it also runs with a linear time. [15]

But the simulator can predict the performance of much more application with other time complexities, rather than only linear time. This can be done by doing experiments with sample programs running in the corresponding time complexity and as long as a model is built up from a set of results, it can be taken as an input to a new simulator. And then the simulator can be used on predictions.

3.5 Architecture

3.5.1 Presentation Layer

Presentation layer is the top-layer in the overall architecture design. All GUI components and test classes are belonged to the same level. Presentation layer does not provide any logical function for predicting. It only manages the user interfaces, deciding how the calculated results are presented to user in a clear and easy way.

Presentation layer is responsible for the delivery and formatting of information to the application layer for further processing or display. It also decides how the user should input their desired configuration parameters, like number of nodes or problem sizes, into the simulator before its initiation.

The major concern of Presentation Layer is the way where all data is showed. As many factors can lead to varying in the predicted results, it is supposed to present the trend that the performance changing along with the growth of all main factors. [16]

In this case, using line charts are considered to be the best option. A line chart can clearly show the relationship between two elements with one of them increases as an independent variable. This model suits the changes of performance and problem size, while the input size growing based on some parameters from users and effecting the ascending trend of performance.

Another factor that is important when deciding the performance is the number of nodes, which also indicates the number of Mappers. A Mapper is designed to be created when a node containing required data blocks, so the number of nodes can influence the performance, by deciding how many Mappers will be created to process the whole data sets. A line chart with a single line is hard to show the effect by the number of nodes, as it only indicates the tendency between two variables. But by adding more lines into the chart, it can be clearly shown how the increase on the number of nodes can improve the performance.

The line chart model is still limited for displaying more factors, such as number of reducers, which can also actually lead to changes on performance. To deal with this, the solution is to display more than one charts at the same time. This is a bit complex to provide extreme clear browse on some factors, but it can describe all information from the simulated results and provide flexibility to analyse through different aspects, which improve the availability of the system.

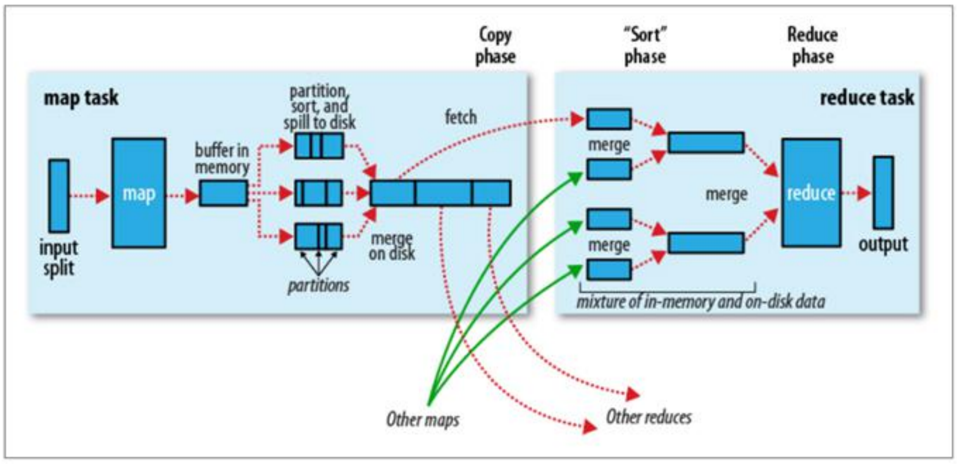
3.5.2 MapReduce Layer

Map-Reduce divides the work into small parts. Input data given to mapper is processed through user defined function written at mapper. All the required complex business logic is implemented at the mapper level so that heavy processing is done by the mapper in parallel as the number of mappers is much more than the number of reducers. Mapper generates an output which is intermediate data and this output goes as input to reducer.

An input to a mapper are created by processing data blocks in local machines. A mapper is firstly responsible for creating the input splits and dividing them into records.

When a map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantages of buffering writes in memory and doing some presorting for efficiency reasons.

MapReduce makes the guarantee that the input to every reducer is sorted by key. The process by which the system performs the sort - and transfers the map outputs to the reducers as inputs - is known as the shuffle. The shuffle is an area of the codebase where refinements and improvements are continually being made. [17]



The following steps are defined in a mapper:

Read data from local disk

This is the first step before any data is processed by a mapper. All data blocks in local disk will be read into memory. The time consumed in this period is relevant to the IO time of the machine, which is declaimed by user.

Read records

The involved data blocks are divided into a set of records for processing. The time complexity is O(n), but the number of records divided is decided by both the size of local data, and a bias, which defines how many records there will be in 1MB data. The bias can change in different cases as the input data sets can be in different type.

Compute

This is where the data is processed in local machine. The running time is predicted by the model which is already built up from the experiment. Each mapper only processes the data in one node, and the involved problem size is decided by the number of records.

Sorting

Before send it to reducers, a mapper does a sorting on its local results set. Generally the used sorting algorithm is Quick Sort, so the time complexity is O(nlogn) at this stage.

Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to. Within each partition, the background thread performs an in-memory sort by key, and if there is a combiner function, it is run on the output of the output. Running the combiner function makes for a more compact map output, so there is less data to write to local disk and to transfer to the reducer.

It is often a good idea to compress the map output as it is written to disk, because doing so makes it faster write to disk, save disk space, and reduces the amount of data to transfer to the reducer. By default, the output is not compressed in Hadoop, but it is easy to enable this.

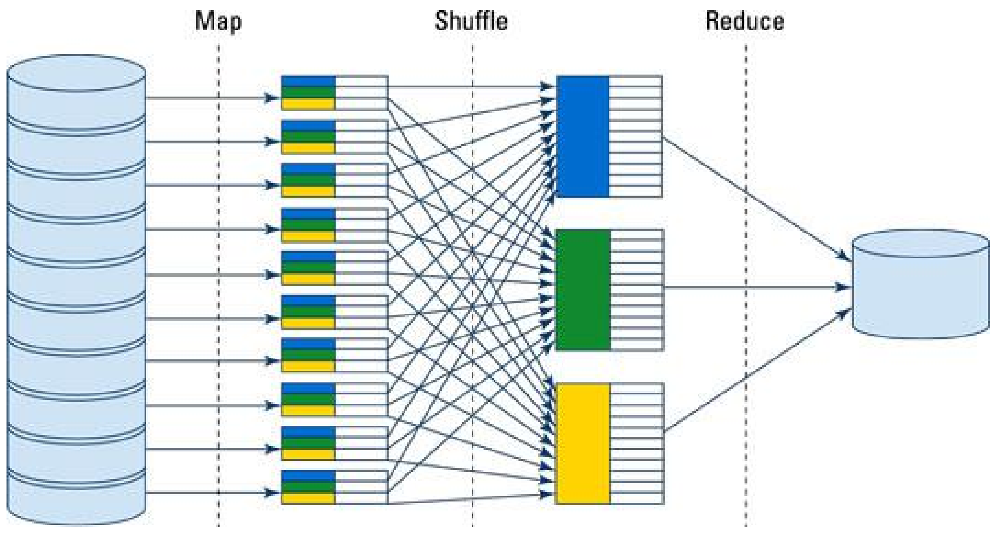
As the amount of data will change before it is sent to reducer, so another bias is used to define the margin when a mapper outputs its data in each local node. The total amount sent to reducer is decided by the input data to mapper and the bias.

The data transfers from mappers to reducers are made available over HTTP. There will be a network transfer delay, which is predicted by input network bandwidth and the amount of data.

In Hadoop, the map output file is sitting on the local disk of the machine that ran the map task (although map outputs always get written to local task, reduce outputs may not be), but now it is needed by the machine that is about to run the reduce task for the partition. Moreover, the reduce task needs the map output for its particular partition from several map tasks across the cluster. The map tasks may finish at different time, so the reduce task starts copying their outputs as soon as each completes.

When all the map outputs have been received, the reduce task moves into the sort phase (which should properly be called the merge phase, as the sorting was carried out on the map side), which merges the map outputs, maintaining their sort ordering.

During the reduce phase, the reduce function is invoked for each key in the sorted output. The output of this phase is written directly to the output file system, typically HDFS. In the case of HDFS, because the node manager is also running a datanode, the first block replica will be written to the local disk.



3.5.3 HDFS Layer

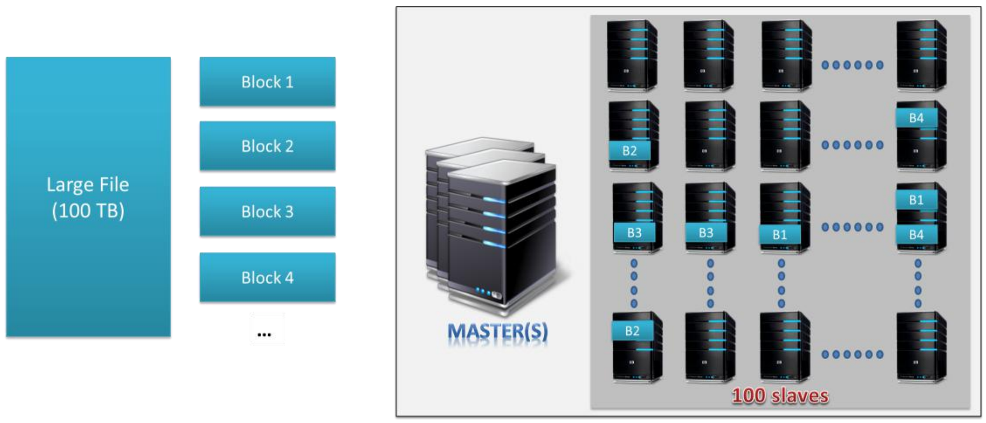
HDFS is layered below MapReduce to provide data access operations and storing algorithms. HDFS is the world's most reliable storage system. HDFS is a file system of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on principle of storage of less number of large files rather than the huge number of small files. [18]

HDFS works in master- slave fashion, where there are namenodes and datanodes in the cluster. Namenodes regulates and manages the slave datanodes and assign tasks to them. It also maintains its slave nodes to handle any storing failure. Namenodes executes file system namespace operations like opening, closing and renaming files and directories.

Datanodes are where the big data is actually stored. They work to do the storing tasks and serve read and write requests from the file system's clients. They also perform block creation, deletion, and replication upon instruction from the namenode.

In the simulator, nodes are considered to be Datanodes, where each contains a set of data block which will be processed by MapReduce. Namenodes are not taken into account because the simulator aims majorly to predict performance of an application, rather than the reliable failure tolerate mechanism provided by HDFS.

Whenever any file has to be written in HDFS, it is broken into small pieces of data known as data blocks. The size of data block can be defined by users, while it’s set to 256 MB by default. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for MapReduce to process the data in parallel in the cluster.



Multiple copies of each block are stored across the cluster on different nodes. There is a replication number of data, which can also be defined by users. By default, the replication is 3.

In the simulator, HDFS stores a list of all data blocks. When a file is input, new data blocks will be created according to its fixed size, and then put into the data blocks list in HDFS. But no replication data block is created by this stage, this is done in the next step, when HDFS calls an utility, which decides how the data blocks are allocated into nodes. The defined utility takes the lists of node and data blocks in HDFS as inputs, then create a linked list using a certain algorithm and replication data blocks are created here.

Each element in the linked list claims one data block, either the original one or a replication, and a node that will receive the data block, based on the decision made by utility. In each data block, a flag is defined, which can be assigned a value either 0 or 1, where 1 presenting this data block is an original one, created by HDFS, while 0 for replications.

When MapReduce processing data, only data blocks with flag 1 will be considered, but all data blocks, including replications are all stored in nodes' disks.

HDFS uses an Utility instance to create the list of data blocks and decide the storage. The Utility works using a data structure of a LinkedList, each element in the LinkedList is a vector, which length is the number of replication setting to 3 by default. The elements in the vector claims a data block, and its corresponding node. After the list is fully created, HDFS will distribute all data blocks, including their copies, to nodes and put it into the node's data list.

3.5.4 Physical layer

Servers, also known as nodes, work as physical infrastructure in Hadoop architecture. Physical factors also have influence on the performance. HDFS also needs to consider physical disk spaces to distribute data blocks, or handle any exception and error.

In the simulator, the most important factor taken into account is network bandwidth. This is because network bandwidth is strongly related to delay time, occurring when data is transferred to reducer over network after mapping operations.

3.6 Functions

The simulator focuses on two main functions for prediction and analysis. The first one is to present the trends of the growth of performance with the increasing problem size using diagrams. In one diagram, there can be more lines representing a certain number of nodes, therefore, also reflecting the number of mappers.

It is also possible to compare the effects by other factors, i.e. number of reducers, network bandwidth, and I/O or operation time, by presenting two or more diagrams at the same time and compare the performance - the data in the Y-axis, while X-axis showing the input size.

User interface of the simulator provides flexible parameters for user to input, aiming to design the diagram as their wishes. For input size, the factors of minimum data size, maximum data size, and the increment, which is the gap between two neighbour point in X-axis, can be defined. And for number of nodes, similar, the minimum number, the maximum number, and the increment can be changed flexibly. But the setting for these numbers will be reflected by different lines in the diagram, not the data or points in axis. For reference, a cutline will be displayed in the diagram to introduce the number of nodes representing by each line. Lines will be distinguished by different colors.

Another function, ignoring a set of continuous input but focusing on a single input of data and nodes number, is to predict the best performance that can be achieved by the present configuration. But the variable is considered to be the size of data block in HDFS. So in this case, the simulator will predict the performance using different size of data block, and record the results of each experiment. To report the result to user, the simulator will find out the best performance and the data block size that leads to it. So the predicted size can be regarded as the one that is most likely to produce a better performance.

3.7 Data bias

In the simulator, two biases are defined to provide more accurate predicted results. The first one, called “record bias”, means the number of records come from 1 MB data. Basically, this decides the data amount input to the mapping operation. The other one is “size bias”, as mappers will produce intermediate data, and this bias decide the relationship between original data in the data block and intermediate data.

These two factors are defined because for a same application, different type of input data can result in different performance. For example, a WordCount program will present different running times on English text and other language text, like Chinese. So the biases provide flexible interfaces to adapt the possible difference bought by types of data.

3.8 Other factors

Network bandwidth
For large data set, network bandwidth can strongly effect the performance. An average network bandwidth is one of the input factors for the simulator, and also can be changed to simulate different network environments.

I/O time
The time for inputing/outputing data block into memory. This is useful in both mapping operation, reading data from HDFS, and reducing operation, simulating the time to output.

Compute time per record
This is useful for simulating the performance in records split, sorting. and merging operations. For a initiated Hadoop, the time complexity for these operations are usually fixed and can be known. So the time for pre-processing and such operations can be predicted. But the computing time on each single node is not related to this factor. Instead, it is decided by the model produced by the experiment based on a sample program and small testing inputs.

Chapter 4

Implementation

4.1 Experiment

A WordCount program is used as the application in the experiment. No complicated algorithm is required in this part, but a very clear implementation of simple time complexity is important. The program will take files as input, so BufferedReader is used to go through the files.

BufferedReader provides a readLine function, by calling this, file can be read by each line. The results will be temporarily copied into a String object and then store into a list of String for processing and sorting.

As only word is considered in the application, so when retrieving each single line, non-alphabetical characters are not put into account.

```
//Retrieve the file
 while((readLine = br.readLine()) != null){
 //Count alphabetic character
        String[] wordsArr1 = readLine.split("[^a-zA-Z]");
        for (String word : wordsArr1) {
           //Remove empty word
            if(word.length() != 0){
               //Put valid word into list
                lists.add(word);
            }
        }
    }
```

Code for retrieving text file

After putting all valid words in the list, the results will be used to create a Map instance. A TreeMap object can store data in <key, value> format. In the WordCount, it is defined as <String, Integer> to record the statistical number for all different words.

The test input files for the sample program are in very small size, but the purpose is to create a model reflecting the relationship between size and performance. The model will be taken as an input factor to the simulator.

```java
//Create the TreeMap
Map<String, Integer> wordsCount = new TreeMap<String,Integer>();

for (String li : lists) {
    if(wordsCount.get(li) != null){
        wordsCount.put(li,wordsCount.get(li) + 1);
    }else{
        wordsCount.put(li,1);
    }
}
```

Code for counting words

4.2 HDFS

HDFS is one of the most important part in the simulator. It needs to provide distributed storage for MapReduce to process data. To simulate the real use case in HDFS, it is challenging to demonstrate the data structure, the strategy, and the storing process, and implement them with visualization. I did research on some materials and tried some possible ways to implement this part. Some core effort are presented in this section.

4.2.1 Data Block

HDFS stores data by using data blocks. Each data block will need to contain some information for tracking and processing. One required records, as HDFS uses a certain number of replication on each data block, is a flag to mark if the data block is a copy which will not be taken into account when MapReduce search local data. The flag is defined to be an integer, though available assigned value is either 0, representing this data block is a copy, or

1, for the original one. It is possible that all data blocks are with the flag of 1, when the replication number is set to 1 when initiating HDFS.

For one simulator, all data blocks are in the same size. But it is still needed to record the size in each data block. And in case it is not particularly declared, the value of size should be assigned a default value. 256 MB is used as the default value, but in actual HDFS use case, the number is usually 128 MB.

There should be no other difference between an original data blocks and its copies, except the flag field. So HDFS needs to guarantee the replication for the one data block contain same information when it is created. My first attempt was to create a new one, and set all its values to the same values as the original data block, then change its flag to 0 for declaring it is a copy one. But a better way was found, which is implementing the data block with Cloneable interface. The interface provides a clone function to make a copy of this object.

Once the data block implements the interface and overrides the clone function, it can be called from HDFS for one or more times, depending on the number of replication.

```java
//The function for cloning a data block
@Override
public Object clone(){

    //Declare a new object
    DataBlock block = null;

    try{
        //The super class is Object
        //Cast the cloned object into DataBlock
        block = (DataBlock)super.clone();
    }catch(CloneNotSupportedException e){
        e.printStackTrace();
    }

    return block;
}
```

Code for implementing clone function

4.2.2 Data List

After HDFS takes its input, it creates data blocks and adds the newly created ones into a data list. But the replication has not been considered yet.

The amount of created data block in this stage is based on the input information, basically the size of file. It is also relevant to the fixed size of data block. HDFS will keep create new data block until the total size of all blocks meets the requirement of input size, which means all data have already been simulated in the form of data blocks.

```
//Break new file into data blocks
private List<DataBlock> distribute(File file){
    //Temporary total size
    int sum=0;
    double size = file.getSize();

    //Create new data blocks until the size is larger than input
    while(sum<size){
        DataBlock newBlock = new DataBlock();
        //Data block size is defined in HDFS
        newBlock.setSize(HDFS.getDataBlockSize());
        sum += newBlock.getSize();
        //Add the new data block to data list
        HDFS.getDataList().add(newBlock);
    }

    return HDFS.getDataList();
}
```

Code for creating data list

This function will be used when any file is input. So once HDFS receives an input, a set of data blocks will be created based on its size. There are two ways for inputing file information to HDFS, as it can be set through user input or using a json file. The details for this part will be explained in latter part.

4.2.3 Utility

In the simulator, HDFS uses an utility instance for distributing storing. The aim of using utility is that in actual use cases, there could be different strategies for data block assignment, and different algorithms can be referred.

The utility is created in the process when HDFS is initiated. HDFS sends parameters to utility, and the functions defined in utility can make up a data block list for distributing. These input parameters include the data list in HDFS, as introduced above, and a node list with all available nodes, and the replication number.

The most difficult part of utility is how it should store the results which points out the data blocks and their corresponding nodes. A data block list is considered to be a good solution. The data block is in a data structure of LinkedList, with a set of sub-lists. Each sub-list represents one data block and its replication, so the length of a sub-list is equal to the replication number.

The element in the list is specially designed. Except data block and a node, it also shows the flag for reference in HDFS.

```
public class UtilityElement {
    //The node where the data block is going to be stored
    private Node node;

    //One data block in HDFS data list
    private DataBlock dataBlock;

    //Will be set as 1 for original data block
    private int flag = 0;
}
```

Code for defining utility element

The utility will first create the data block list. For each data block, it stores itself and replication in a vector. The vector contains a set of utility element shown above. Then the vector will be added into the LinkedList.

```
public void distribute(){
    int nodeNum=0;

    for(int DataBlockNum=0;DataBlockNum<dataList.size();DataBlockNum++){
        //Create new vector for a new data block
        Vector<UtilityElement> newVector = new Vector<UtilityElement>();
          for(int newDataBlock=0;newDataBlock<replication;newDataBlock++,nodeNum++){
            if(nodeNum>=nodeList.size()) nodeNum=0;

            //Create new element
            UtilityElement element = new UtilityElement();
            DataBlock dataBlock = new DataBlock();

            //Use clone function in data block to copy replication
          dataBlock =(DataBlock)dataList.get(DataBlockNum).clone();

            //For original data block, set the flag to 1
            if(newDataBlock==0) {
                dataBlock.setFlag(1);
            }
            element.setDataBlock(dataBlock);
            element.setNode(nodeList.get(DataBlockNum));

            //Add element to vector
            newVector.add(element);
        }

        //Add vector to data block list
        list.add(newVector);
    }
}
```

Code for creating data block list

The utility will then add all data block in the created list to their assigned nodes' data lists. The information about the node can be found from the records in utility elements.

```
public void allocate(List<Node> nodeList){
    for(int dataBlcokNum=0;dataBlcokNum<list.size();dataBlcokNum++){
        for(int replicationNum=0;replicationNum<list.get(dataBlcokNum).size();replicationNum++){

            //Get the information about data block and node
            UtilityElement element = new UtilityElement();
            element = list.get(dataBlcokNum).get(replicationNum);
            int id = nodeList.indexOf(element.getNode());

            //Add the data block the data list in nodes
            nodeList.get(id).getDataList().add(element.getDataBlock());
        }
    }
}
```

Code for adding data blocks into nodes

4.2.4 HDFS input

HDFS needs inputs of the size of simulated data. In the simulator, the inputs will be records as files. Files can be created in either typing information in user interface, or reading from a json file.

The former method is easy to implement as it only takes text data from user interface. The implementation of user interface will be introduced in latter section.

A json file is considered because it is easy to load and simulate the input as a set of files. The input json file needs to define the file name, and the data size.

```
{"files":[

    {
        "Filename": "file1",
        "Size": 10240
    }

]
}
```

Sample json file

HDFS reads the input json file using a JSONParser. Each file defined in the json file will be declared as a JSONObject, and then the information will be loaded by setting the corresponding data in a file instance in the simulator. The new instance will then be added into the file list of HDFS which will be referred in the distribution process.

```
public void readJson(String str) throws FileNotFoundException, IOException, ParseException{
    //Create new JSONParser
    JSONParser parser = new JSONParser();
    Object obj = parser.parse(new FileReader(str));

    JSONObject jsonObject = (JSONObject) obj;
      JSONArray files = (JSONArray) jsonObject.get("files");

      //Retrieve all files defined in json
      for(Object file : files){
        File newFile = new File();
          JSONObject jsonFile = (JSONObject) file;

          String newFilename = (String)jsonFile.get("Filename");
          newFile.setFileName(newFilename);
          String newSize = jsonFile.get("Size").toString();
          newFile.setSize(Double.parseDouble(newSize));
```

```

            //Add new file into file list
            HDFS.fileList.add(newFile);
            distribute(newFile);
        }
    }
```
Code for reading Json file

4.3 MapReduce

MapReduce is another core component in the simulator. MapReduce provides main information for the results of prediction. The functions defined in MapReduce are almost used for computing a simulated time in each step when MapReduce handles data in HDFS.

To provide flexible customized prediction, more factors are needed in MapReduce which all have effects to the predicting results to a certain extent.

MapReduce works in four main steps with different factors are required:
Initiation
The initiation is done before MapReduce takes any operation. In this process, a defined number of mappers and reducers are created and added to the list of MapReduce. And biases, both the record bias and the size bias, are set in the initiation part.

Map
This is when data is process in each single node. The mapping is considered to work synchronously in all nodes, as well as all mappers. Operations in mappers include reading data, breaking data blocks into records, computing, and local sorting. The defined I/O time and compute time can effect the results in this part.

Transfer
To simulate the real case, when intermediate is produced and sent to reducers from mappers, this process can compute a possible network delay time. This time is influenced by the average network bandwidth, the data

amount, and the number of mappers and reducers.

Reduce
Each reducer does a merging operation on all the data it receives in the transferring process. Reducers can also put the outputs into HDFS, so there is an I/O time taken up in the process.

MapReduce has LinkedList for recording all nodes, mappers, and reducers. The initiation part needs to be processed when any new factors is input. The other important parameters required by MapReduce is the model created by the experiment which points out the relationship between size and performance.

Other factors, including biases, number of reducers, I/O time, compute time for each operation, and network bandwidth are also needed by MapReduce.

4.3.1 Initiation implementation

Values of some basic factors are set in the initiation. But the most important part of initiation is to create mappers and reducers as all the following steps are based on mapping and reducing functions.

The creations are simply done by using a loop. But different factors are referred. For mappers, as the simulator assumes there is one mapper on one node, the creation follows the size of nodes list in MapReduce. When a node is created, it is added into the list. So this guarantees the number of mappers is same as the number of nodes. For reducers, it is simply based on the input reducers number.

```
public void initiate(Model model, double sizeBias, double recordBias){
        //Set basic factors
        MapReduce.sizeBias = sizeBias;
        MapReduce.recordBias = recordBias;
        this.model = model;

        //Create mappers according to the size of nodes list
        for(int i=0;i<MapReduce.getNodeList().size();i++){
            if(MapReduce.getNodeList().get(i).getDataList().size() != 0){
                Mapper newMapper = new Mapper(this);
                newMapper.setNode(MapReduce.getNodeList().get(i));
                //Add new mappers
                MapReduce.mappers.add(newMapper);
            }
        }

        //Create reducers accoding to the set number
        for(int i=0;i<MapReduce.getReducersNum();i++){
            Reducer newReducer = new Reducer();
            newReducer.setName("Reducer " + i);
            //Add new reducers
            MapReduce.reducers.add(newReducer);
        }
    }
```

Code for MapReduce initiation

4.3.2 Map implementation

The special part in a mapper is the actual prediction of computing time. The model from the previous experiment is needed here.

```
//Time - Processing data
public double compute(){
    double computeTime = 0;

    computeTime += mr.getModel().model(this.recordsNum);
    //Transfer the unit to second
    computeTime = computeTime / 1000 ;

    return computeTime;
}
```

Code for using model to predict computing time

The other parts of mapping are based on the number of data or records, and the time complexity of the algorithm. For the local sorting part, it is possible that the used sorting algorithm is changed. But the simulator uses the time complexity of O(nlogn), which fits the situation in most use case.

4.3.3 Transfer implementation

The implementation of transfer process needs to simulate each reducer takes one partition of the data in all mappers. In the same word, mappers divide their local data into partitions, where the number is based on the number of reducers, and then send each partition to a reducer. So the operations in this part decide the size of input data in the next part when reducers do merging and output.

The network delay time is also relevant to the size of transferred. All mappers and reducers are also considered to work synchronously to send or receive data. An overall delay time is predicted, and this will be regarded the network delay when the simulator produces the final predicting results.

```
public double transfer(){
    double delayTime = 0, overAllDelay = 0;
     for(int mapperNum=0;mapperNum<MapReduce.getMappers().size();mapperNum++){
         for(int reducerNum=0;reducerNum<MapReduce.getReducers().size();reducerNum++){
            //Send data partitions to reducers
             MapReduce.getReducers().get(reducerNum).addRecordsNum(MapReduce.getMappers().get(mapperNum).getRecordsNum() / MapReduce.getReducersNum());

             double data = MapReduce.getMappers().get(mapperNum).getDataSize() / MapReduce.getReducersNum();
            //Multiply size bias
            data *= MapReduce.getSizeBias();

            //Compute network delay
            delayTime = data / MapReduce.getNetworkBW();
            //Find the longest delay time
            if(delayTime > overAllDelay) overAllDelay = delayTime;
            MapReduce.getReducers().get(reducerNum).setDataSize(MapReduce.getReducers().get(reducerNum).getDataSize() + data);
        }
    }
    //System.out.println("*** Overall delay time: " + overAllDelay + " ***");

    return overAllDelay;
}
```

Code for simulating transferring process

4.3.4 Reduce implementation

Each reducer does a merge operation on the data it receives from mappers. The algorithm in this part is also considered to be in time complexity of O(nlogn) to fit almost cases.

Then the data is output, taking times for I/O operation. So the input I/O time is relevant to the predicted results in this part. Also, according to the data bias, the total size of data is decided by the data from mappers multiplying the data bias.

4.4 GUI implementation

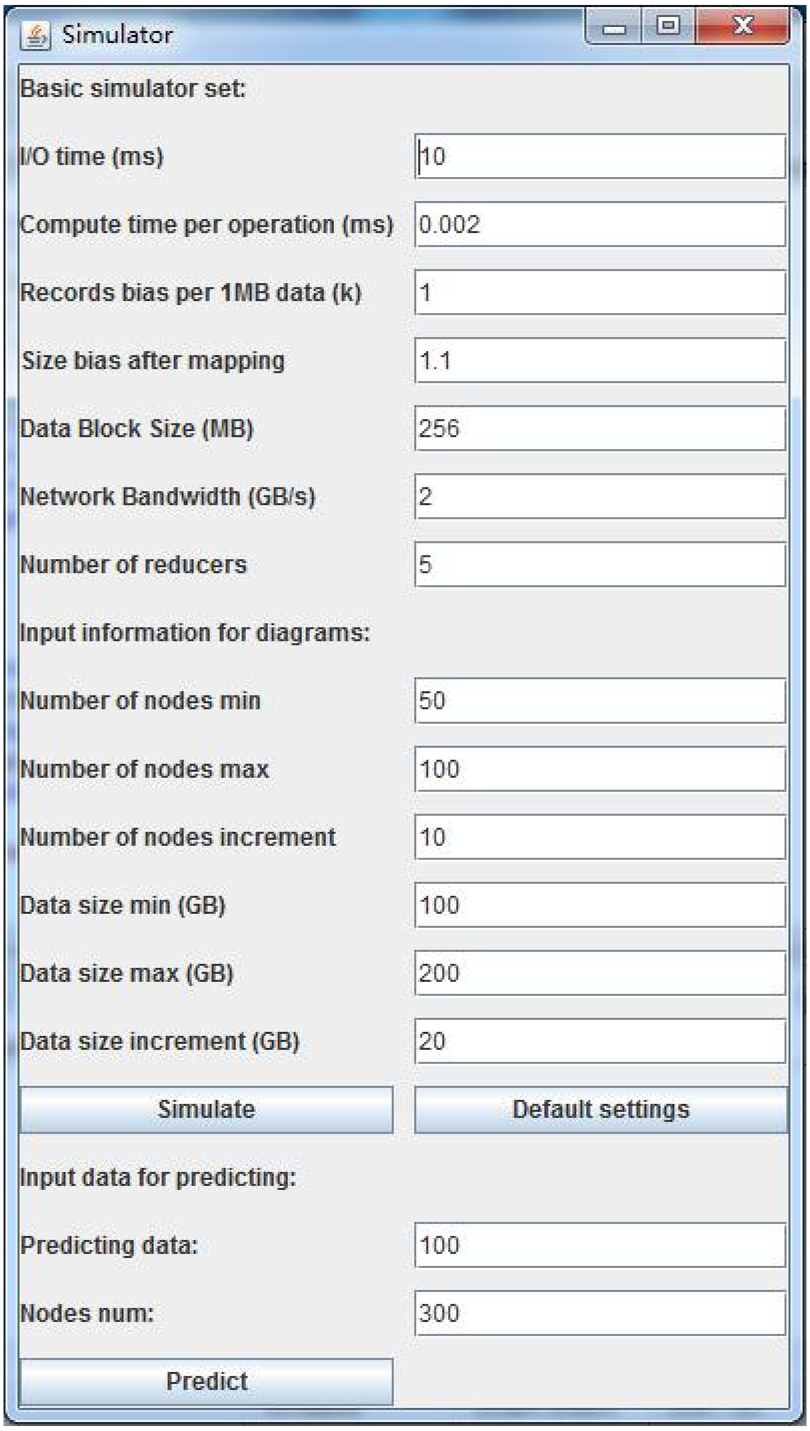
Java frame is used to provide user interface for inputing data and factors in the simulator. And the results are presented by line diagram to show the trends of changing performance.

When initiating the simulator, a frame is shown to user with labels and textfields, and some buttons for different functions. The frame uses a layout so that all components are displayed in correct order.

Each field is assigned with a default value, and the simulator initiates all factors to the same default number in case any input exception occurs. The units of all fields are also presented in the labels to guarantee the data is input in united format.

There are three main parts in the main frame for inputing data. Firstly, the basic simulator factors, define the initiating information for the simulator. Then there are some factors for deciding how the diagram shows the data. This provides users a more flexible mechanism to see the diagram according to their requirements, and the scale of visible data. The corresponding results from this part show continuous changing on the performance. And in the last part of the frame, a single input configuration can provide the function to predict the possible better setting of data block size, and the predicted performance the predicted configuration may reach.

Buttons for each functions are also displayed. The **Simulate** button can show the diagram of changing performance based on all configurations above. By changing data in the user interface and clicking this button for several times, two or more diagrams can be presented. For easily comparing the effects of factors, the **Default settings** button recovers all fields to the default values which as shown in the picture below. The **Predict** button shows a pop-up window with predicting results on it.



Simulator

Basic simulator set:	
I/O time (ms)	10
Compute time per operation (ms)	0.002
Records bias per 1MB data (k)	1
Size bias after mapping	1.1
Data Block Size (MB)	256
Network Bandwidth (GB/s)	2
Number of reducers	5
Input information for diagrams:	
Number of nodes min	50
Number of nodes max	100
Number of nodes increment	10
Data size min (GB)	100
Data size max (GB)	200
Data size increment (GB)	20
Input data for predicting:	
Predicting data:	100
Nodes num:	300

Screenshot for user interface

The diagrams use java chart components. It extends java ApplicationFrame class. The configuration data is received from the frame, and is added to the data list for the chart.

```
public Chart(String title, LinkedList<LinkedList<DataPlot>> dataset) {
    super(title);
    Chart.setData(dataset);
    // TODO Auto-generated constructor stub
    setContentPane(createDemoLine());
}
```

Code for initiating chart

```
public static JFreeChart createChart(DefaultCategoryDataset linedataset){
    //Create line chart
     JFreeChart chart = ChartFactory.createLineChart("Performance", "Size / GBs",
"Time / mins", linedataset, PlotOrientation.VERTICAL, true, true, true);
    CategoryPlot plot = chart.getCategoryPlot();
    plot.setRangeGridlinesVisible(true);
    //Define axis
    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
    rangeAxis.setAutoRangeIncludesZero(true);

    return chart;
}
```

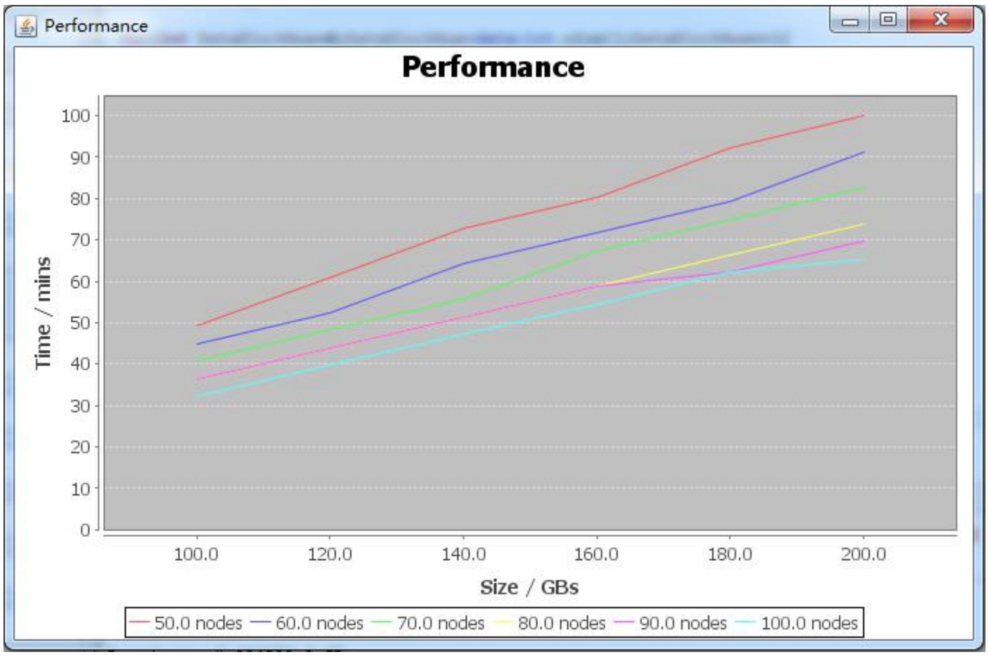
Code for creating line chart

To add plots in the line chart, a dataset is created to store all temporary data of the predicted results. A LinkedList with sub-lists are also referred in the chart, as the chart is supposed to present data in the circumstance of using different number of nodes. The values of plots for a certain number of nodes are stored in a sub-list. The chart can then set the plots in different lines according to the data in the list.

In the line chart, lines are distinguished by different colors and a brief cutline is also displayed in the chart for reference.

```java
public static DefaultCategoryDataset createDataset(LinkedList<LinkedList<Data
Plot>> data){
    //Create dataset
    DefaultCategoryDataset linedataset = new DefaultCategoryDataset();

    LinkedList<String> series = new LinkedList<String>();

    //Add lines for different number of nodes
    for(int i=0;i<data.size();i++){
        String newSeries = data.get(i).get(0).getNodeNum() + " nodes";
        series.add(newSeries);
    }

    //Add plots to each line
    for(int i=0;i<series.size();i++){
        for(int j=0;j<data.get(i).size();j++){
            String type = Double.toString(data.get(i).get(j).getSize() / 1024);
            double time = data.get(i).get(j).getTime();

            //Set the plot
            linedataset.addValue(time , series.get(i), type);
        }
    }

    return linedataset;
}
```

Code for adding lines and plots to the chart



Screenshot for line chart

Chapter 5

Evaluation

5.1 Introduction

This section will give details of the testing that was carried out during and after the development of the simulator. Although the duration of the simulator development prevented me from carrying out more extensive user testing, the feedback collected allowed improvements to be made before project demonstration.

5.2 HDFS testing

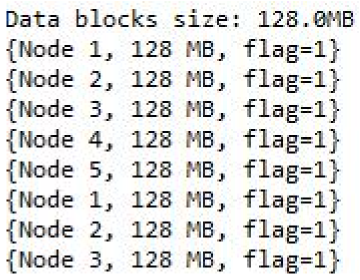
As the basic module to provide distribute data storage for the simulator, the testing on HDFS is carried out before MapReduce to guarantee the data can be correctly loaded into HDFS.

The testing focuses on the creation of data block and data lists. In the simulator, it can be easily accessed and checked out by print the whole created list with all data blocks in it. According to the functional requirements for HDFS, the testing carried out should cover the structure of data list. And different factors, including data block size and replication number should be considered when making testing configurations.

Some testing results are shown below:

Test 1:
Data size: 1 GB
Data block size: 128 MB
Number of nodes: 5
Replication: 1

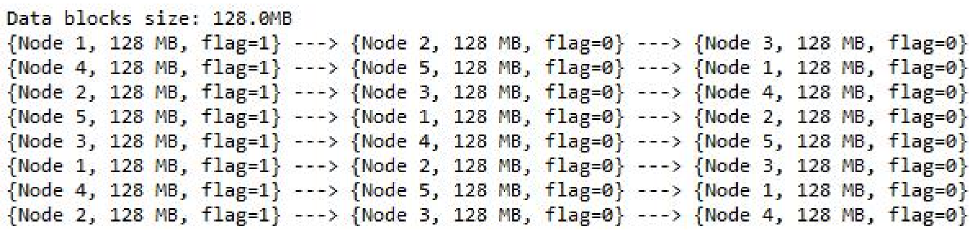
Created data block list:



```
Data blocks size: 128.0MB
{Node 1, 128 MB, flag=1}
{Node 2, 128 MB, flag=1}
{Node 3, 128 MB, flag=1}
{Node 4, 128 MB, flag=1}
{Node 5, 128 MB, flag=1}
{Node 1, 128 MB, flag=1}
{Node 2, 128 MB, flag=1}
{Node 3, 128 MB, flag=1}
```

Test 2:
Data size: 1 GB
Data block size: 128 MB
Number of nodes: 5
Replication: 3

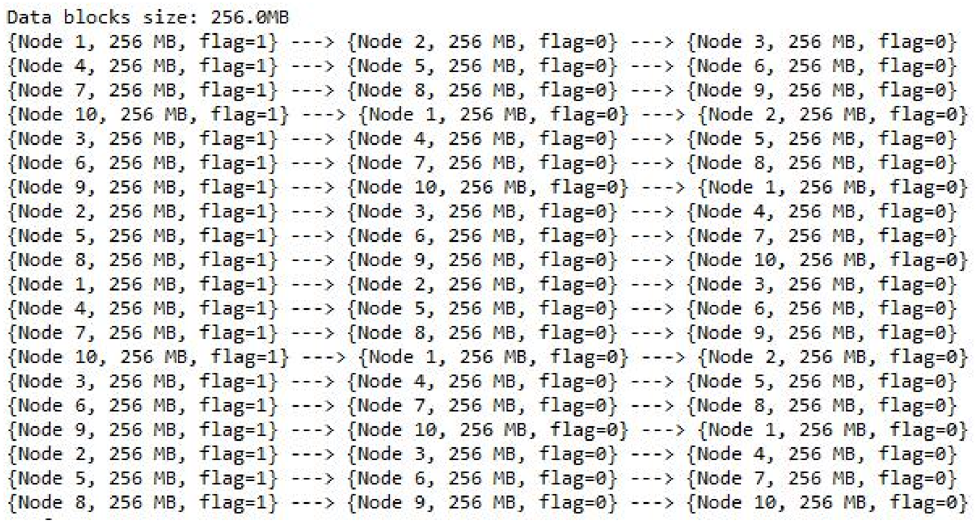
Created data block list:



```
Data blocks size: 128.0MB
{Node 1, 128 MB, flag=1} ---> {Node 2, 128 MB, flag=0} ---> {Node 3, 128 MB, flag=0}
{Node 4, 128 MB, flag=1} ---> {Node 5, 128 MB, flag=0} ---> {Node 1, 128 MB, flag=0}
{Node 2, 128 MB, flag=1} ---> {Node 3, 128 MB, flag=0} ---> {Node 4, 128 MB, flag=0}
{Node 5, 128 MB, flag=1} ---> {Node 1, 128 MB, flag=0} ---> {Node 2, 128 MB, flag=0}
{Node 3, 128 MB, flag=1} ---> {Node 4, 128 MB, flag=0} ---> {Node 5, 128 MB, flag=0}
{Node 1, 128 MB, flag=1} ---> {Node 2, 128 MB, flag=0} ---> {Node 3, 128 MB, flag=0}
{Node 4, 128 MB, flag=1} ---> {Node 5, 128 MB, flag=0} ---> {Node 1, 128 MB, flag=0}
{Node 2, 128 MB, flag=1} ---> {Node 3, 128 MB, flag=0} ---> {Node 4, 128 MB, flag=0}
```

Test 3:
Data size: 5 GB
Data block size: 256 MB
Number of nodes: 10
Replication: 3

Created data block list:



```
Data blocks size: 256.0MB
{Node 1, 256 MB, flag=1} ---> {Node 2, 256 MB, flag=0} ---> {Node 3, 256 MB, flag=0}
{Node 4, 256 MB, flag=1} ---> {Node 5, 256 MB, flag=0} ---> {Node 6, 256 MB, flag=0}
{Node 7, 256 MB, flag=1} ---> {Node 8, 256 MB, flag=0} ---> {Node 9, 256 MB, flag=0}
{Node 10, 256 MB, flag=1} ---> {Node 1, 256 MB, flag=0} ---> {Node 2, 256 MB, flag=0}
{Node 3, 256 MB, flag=1} ---> {Node 4, 256 MB, flag=0} ---> {Node 5, 256 MB, flag=0}
{Node 6, 256 MB, flag=1} ---> {Node 7, 256 MB, flag=0} ---> {Node 8, 256 MB, flag=0}
{Node 9, 256 MB, flag=1} ---> {Node 10, 256 MB, flag=0} ---> {Node 1, 256 MB, flag=0}
{Node 2, 256 MB, flag=1} ---> {Node 3, 256 MB, flag=0} ---> {Node 4, 256 MB, flag=0}
{Node 5, 256 MB, flag=1} ---> {Node 6, 256 MB, flag=0} ---> {Node 7, 256 MB, flag=0}
{Node 8, 256 MB, flag=1} ---> {Node 9, 256 MB, flag=0} ---> {Node 10, 256 MB, flag=0}
{Node 1, 256 MB, flag=1} ---> {Node 2, 256 MB, flag=0} ---> {Node 3, 256 MB, flag=0}
{Node 4, 256 MB, flag=1} ---> {Node 5, 256 MB, flag=0} ---> {Node 6, 256 MB, flag=0}
{Node 7, 256 MB, flag=1} ---> {Node 8, 256 MB, flag=0} ---> {Node 9, 256 MB, flag=0}
{Node 10, 256 MB, flag=1} ---> {Node 1, 256 MB, flag=0} ---> {Node 2, 256 MB, flag=0}
{Node 3, 256 MB, flag=1} ---> {Node 4, 256 MB, flag=0} ---> {Node 5, 256 MB, flag=0}
{Node 6, 256 MB, flag=1} ---> {Node 7, 256 MB, flag=0} ---> {Node 8, 256 MB, flag=0}
{Node 9, 256 MB, flag=1} ---> {Node 10, 256 MB, flag=0} ---> {Node 1, 256 MB, flag=0}
{Node 2, 256 MB, flag=1} ---> {Node 3, 256 MB, flag=0} ---> {Node 4, 256 MB, flag=0}
{Node 5, 256 MB, flag=1} ---> {Node 6, 256 MB, flag=0} ---> {Node 7, 256 MB, flag=0}
{Node 8, 256 MB, flag=1} ---> {Node 9, 256 MB, flag=0} ---> {Node 10, 256 MB, flag=0}
```

The details of data block list can be shown. In the simulator, HDFS handles input data correctly. For testing, the used utility only defines the algorithm of ordered sequence distribution. It is also possible to implement more distribution strategies by creating other utility instance.

It can be observed that the amount of created is relevant to input data size, data block size, and the number of nodes. But when MapReduce processes the data, only one replication of each data block will be taken into account, which are those have flag value of 1.

As the data block size has effect on data block amount, it influence the degree of data distribution which can actually lead to different performance of the system. So it is proved that data block size can change the performance, and it is also shown that the simulator had better give a configuration of data block size that is most likely leading to the best performance.

5.3 MapReduce testing

5.3.1 Diagram and performance

This part of testing consists of practical tasks with different configurations being processed by the simulator. Three groups of configurations parameters are used. More data can also be tested but three would be suffice to do the basic testing and more attempts will be shown in the evaluation sector.

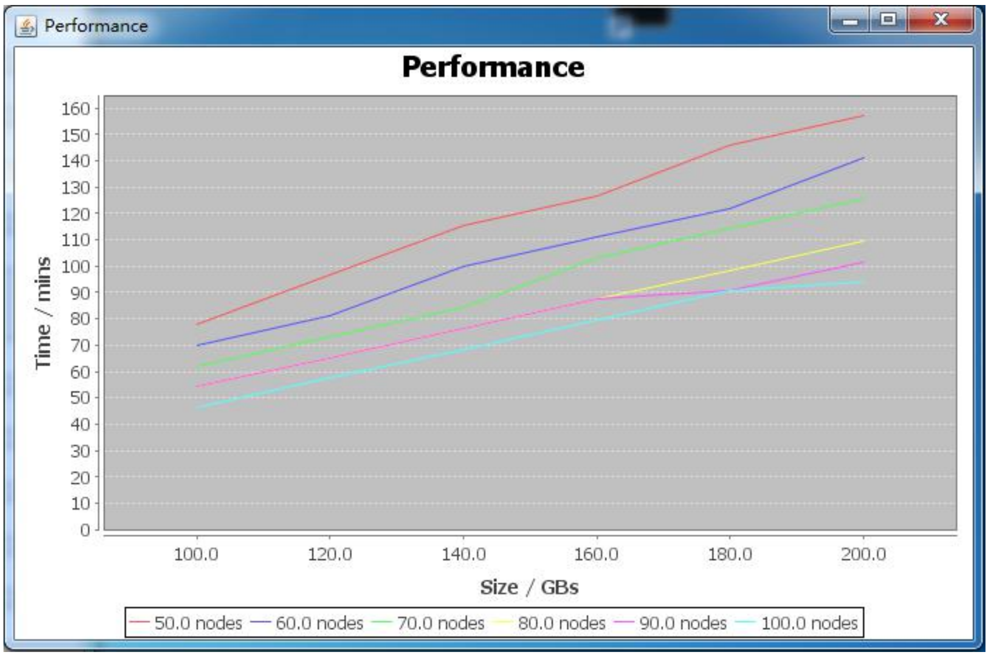
Testing configuration 1: (default settings)
I/O time: 10 ms
Compute time per operation: 0.002 ms
Records bias: 1k
Size bias: 1.1
Data block size: 256 MB
Network bandwidth: 2 GB/s
Number of reducers: 5
Number of nodes: 50 - 100 (increment: 10)
Data size: 100 GB - 200 GB (increment: 20)

Testing configuration 2:
I/O time: 1 ms
Compute time per operation: 0.0001 ms
Records bias: 800
Size bias: 1.1
Data block size: 128MB
Network bandwidth: 3 GB/s
Number of reducers: 20
Number of nodes: 100 - 120 (increment: 5)
Data size: 150 GB - 250 GB (increment: 20)

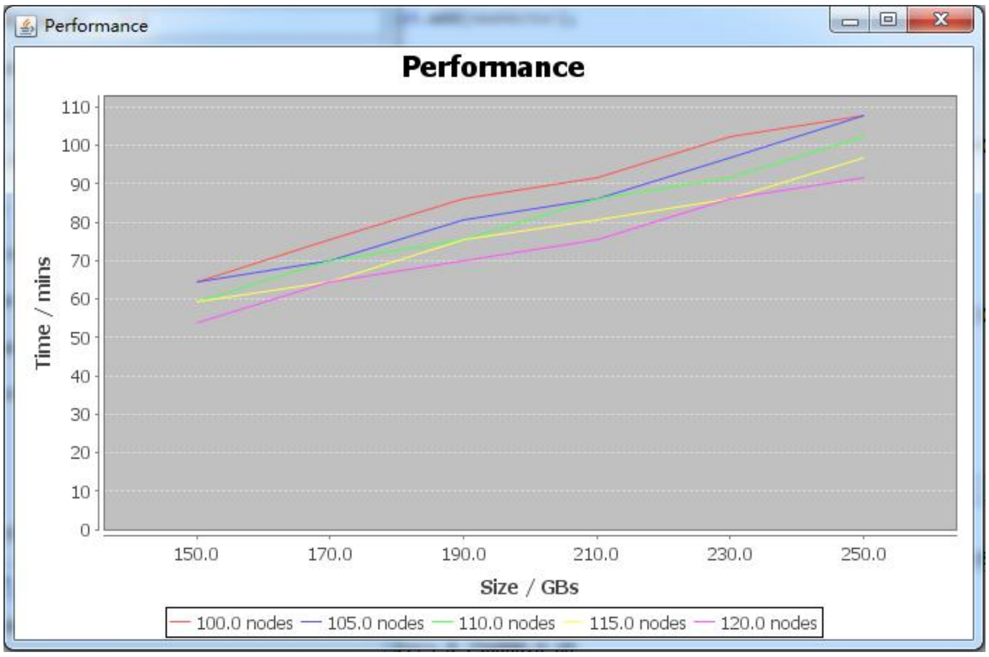
Testing configuration 3:
I/O time: 50 ms
Compute time per operation: 0.01 ms
Records bias: 1200
Size bias: 1.5
Data block size: 256 MB
Network bandwidth: 1 GB/s
Number of reducers: 5
Number of nodes: 40 - 80 (increment: 10)
Data size: 300 GB - 400 GB (increment: 20)

Since the prediction on the performance is the most important part of the simulator, and the performance is presented in diagrams, the testing on showing correct diagrams is firstly carried out.

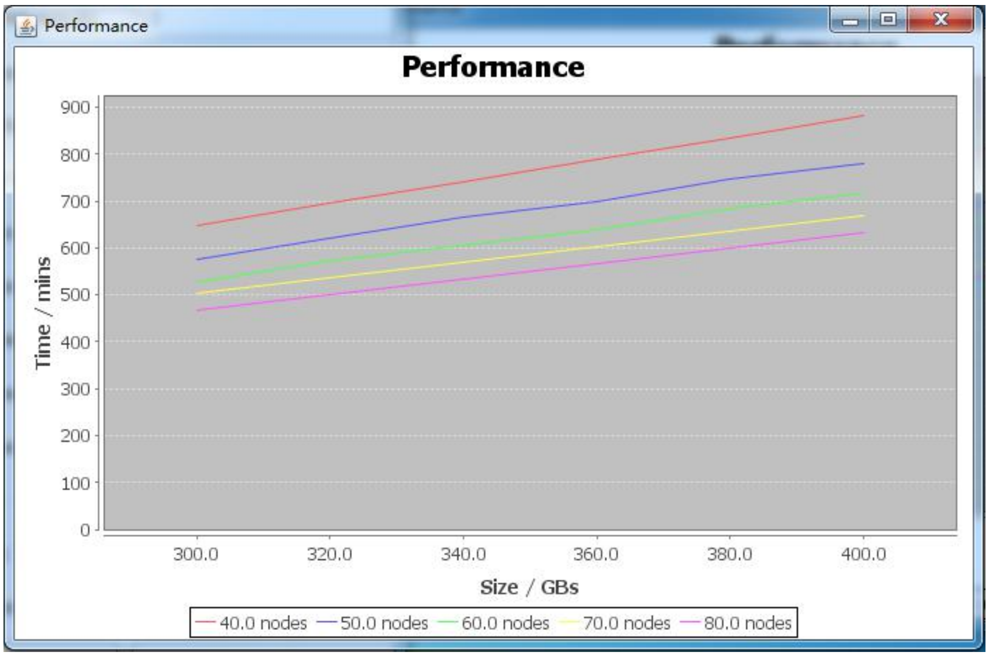
The results collected from the three groups of testing configurations are as below:



Screenshot for diagram on configuration 1



Screenshot for diagram on configuration 2



Screenshot for diagram on configuration 3

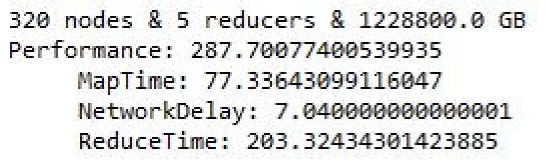
Configuration 1 is the default setting, also considered as a set of normal input parameters. For testing purpose, all other attempts can be compared the predicted performance of configuration 1. The second testing is improved with better I/O and compute time, more reducers, better network bandwidth. According to the results, the performance is improved comparing to the first screenshot, though the total input data set is larger.

But the performance results decrease when testing configuration 3, where most factors are set to be in a worse condition, and processed data is also much larger. This proves the simulator can show changeable performance as input factors changing, and the results obey the approximate expectations. So the simulator can be used to estimate the performance and help analyse the effects of different factors.

It can also be found from the diagram that the performance shows the current configurations have some inadequate points. For example, there are some points of intersection between two lines in the diagram, which means by changing the number of nodes, the performance stays in the same level.

In other words, there is no improvements when adding more nodes into the system. So to figure out the reasons and solve the problem, more attempts can be done to collect more results and see what the performance is under different configurations.

Except the diagrams, the simulator also records all logs information for all the computations it does, so more problems and possible improvements can be found out by looking through the details. For instance, if the simulator reports information like the below screenshot shows, it can be known that the reduce time is the main reason why the performance is poor, while the map time and network delay are in acceptable scales. So to improve the performance, some changes should be done by effectively increasing the performance of the reducer part to make the reduce time shorter. With this idea, other simulations, like using more reducers, can be done to see the effects.



```
320 nodes & 5 reducers & 1228800.0 GB
Performance: 287.70077400539935
     MapTime: 77.33643099116047
     NetworkDelay: 7.040000000000001
     ReduceTime: 203.32434301423885
```

Screenshot for results of inadequate configuration

5.3.2 Prediction for data block size

Some factors are easily observed to be in an inadequate configuration from the diagrams and logs information text, like number of reducers or network bandwidth, as the results are directly presented to users. But some others can be hardly caught for leading to a poor performance, one of which is the size of data block. So the testing on the prediction for data block size is also carried out to see if the simulator can correctly give the advice to improve the performance by changing data block size.

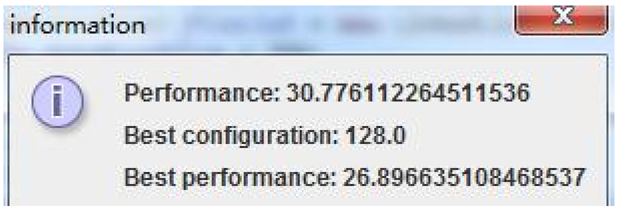
One testing is done by giving the following configuration to the simulator:

Basic simulator set:
I/O time: 10 ms
Compute time per operation: 0.002 ms

Records bias: 1000
Size bias: 1.1
Data block size: 256
Network bandwidth: 2 GB/s
Number of reducers: 5

Input data for predicting:
Predicting data: 100 GB
Number of nodes: 300

The simulator can then give the results as the below screenshot shows. The first performance is based on the current default data block size, which is 256 MB. Then it suggests the best configuration is 128 MB and the expected performance by setting that number is also presented.



```
Performance: 30.776112264511536
Best configuration: 128.0
Best performance: 26.896635108468537
```

Screenshot for prediction best data block size

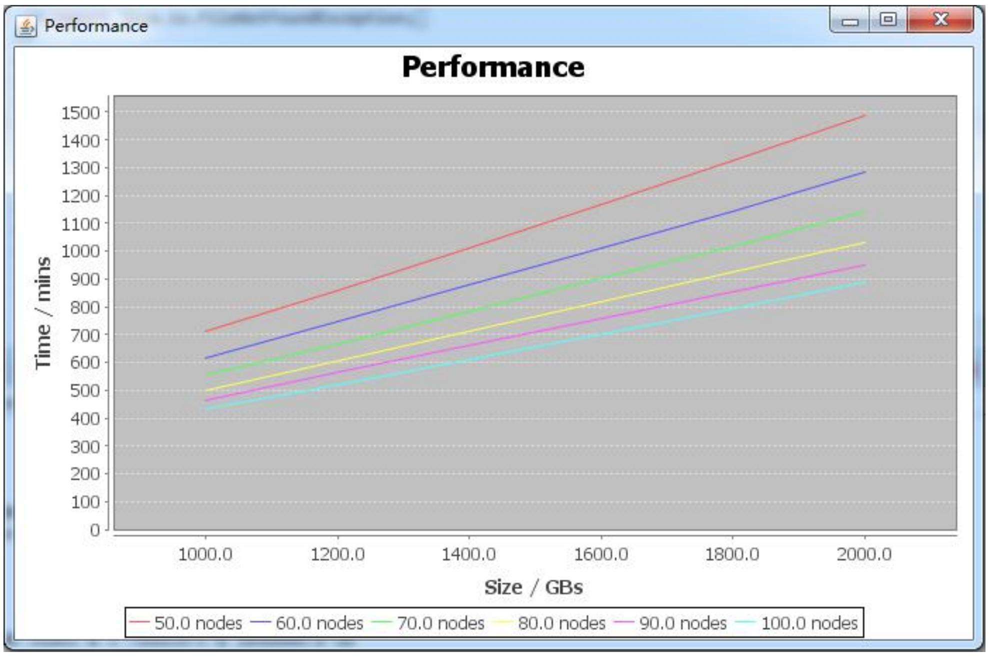
5.4 Evaluation

To evaluate the system, a set of continuous simulations can be done to collect predicted results and give some conclusions. This section will discuss the experiment results and corresponding diagrams.

For finding a control variable, the first simulation is done and all following simulations are compared by changing one or a few similar factors so that the effects can be analyzed. To begin with the evaluation, the first configuration and its predicted result are as below:

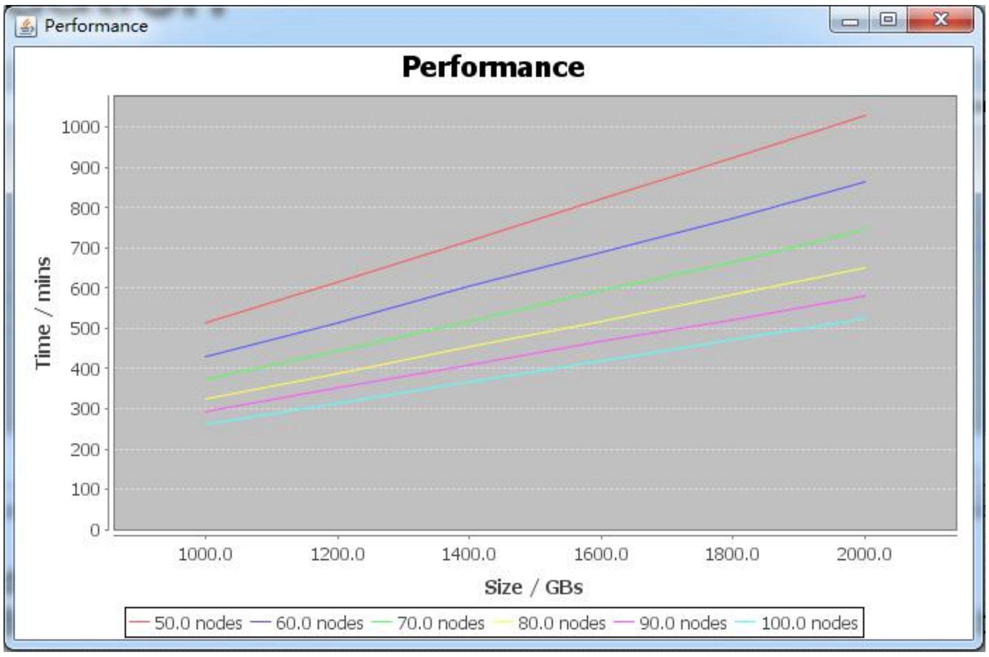
I/O time: 10 ms
Compute time per operation: 0.002 ms
Records bias: 1k
Size bias: 1.1

Data block size: 256 MB
Network bandwidth: 2 GB/s
Number of reducers: 5
Number of nodes: 50 - 100 (increment: 10)
Data size: 1000 GB - 2000 GB (increment: 200)



This simulation runs on a very large input data set, so the performance is very poor at this stage. But it can show the improvements brought by any changes more easily. The first attempt to improve the performance can be started with CPU performance, basically the I/O time and the compute time for each operation.

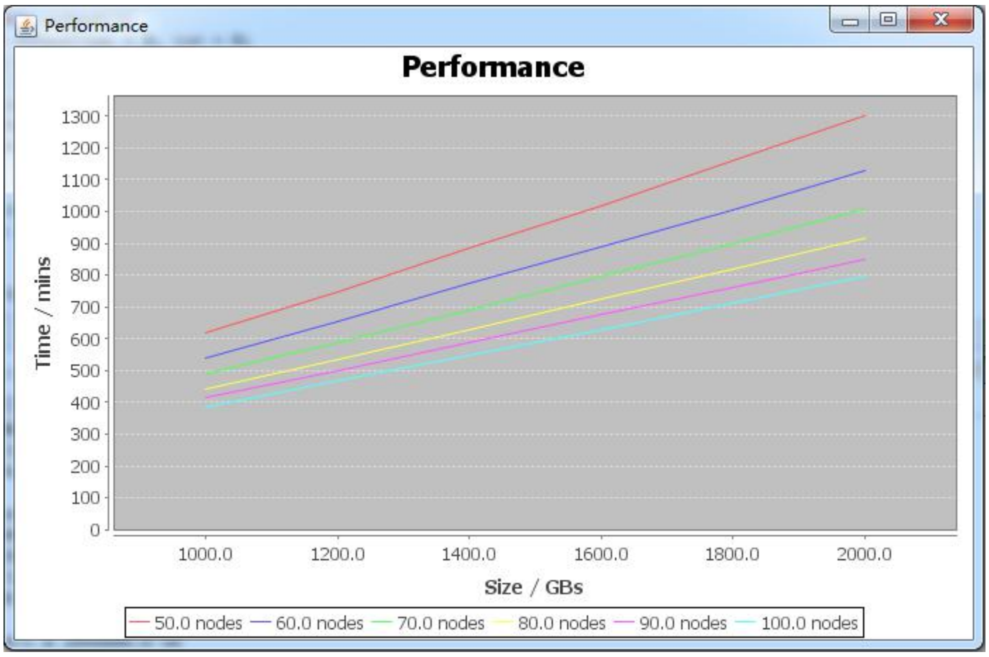
I/O time: 10 ms ---> 1 ms
Compute time per operation: 0.002 ms ---> 0.0001 ms



It can be seen that the performance is improved as CPU performance coming better. Faster I/O operations can improve both the map and reduce part, for mappers reading data from local node and reducers outputing results data into HDFS. And by improving compute time, the time of breaking data blocks into records, sorting, and merging can all be reduced, although the problem size and time complexity remain same.

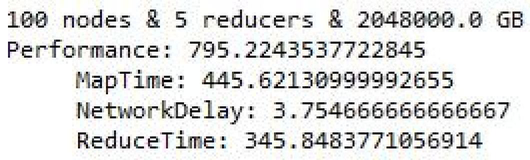
But the network delay time is not shortened. So to improve the performance in this part. The simulation with better network quality can be tried while recovering the I/O time and compute time to the original configuration as we want to see the effect by network bandwidth.

Network bandwidth: 2 GB/s ---> 20 GB/s



Due to faster network data transferring, the performance is improved a little bit. But the effect is far less than the simulation we did by improving CPU performance. Because in the previous simulations, the time on network delay does not contribute to large part. But network bandwidth is actually a very important part in a Hadoop cluster, especially when the processed data set is extreme large, poor network can weaken the performance greatly.

So to move forward on improving the performance, there is little space to improve on the network delay, as even though the network bandwidth is increased a lot, the effect it brings is not remarkable. In this situation, the logs information can help to understand the current performance better.

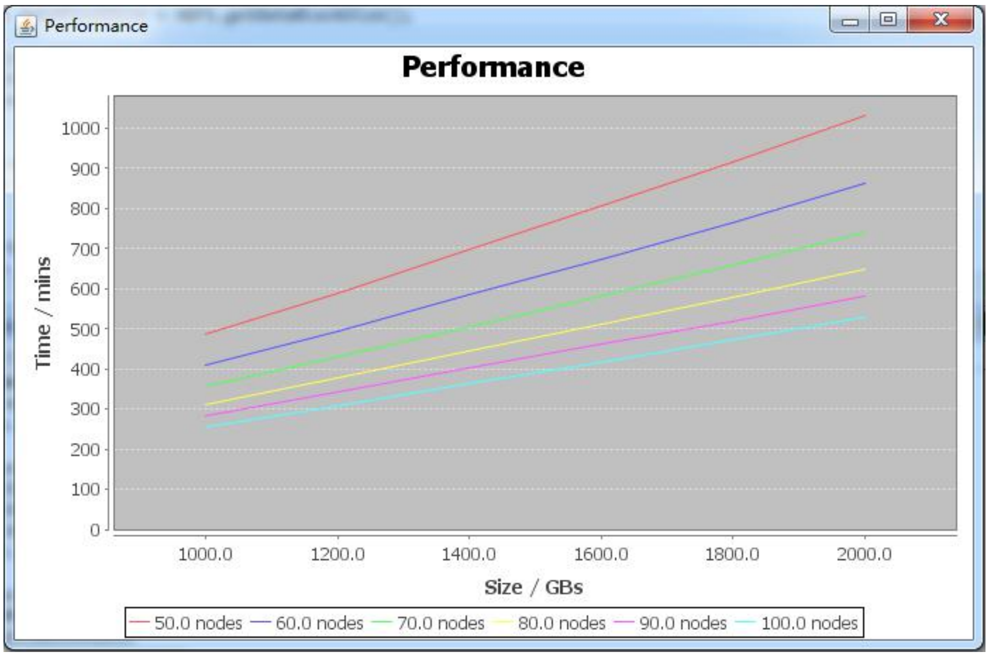


```
100 nodes & 5 reducers & 2048000.0 GB
Performance: 795.2243537722845
     MapTime: 445.62130999992655
     NetworkDelay: 3.7546666666666667
     ReduceTime: 345.8483771056914
```

From the screenshot, the network delay time is indeed very short. But both map time and reduce time is too long. So to effectively improve the performance, some changes should be done on mapping and reducing part. We can firstly try to shorten the reduce time.

The main factor that effects the reduce time is the number of reducers. Not the simulation is done with 100 nodes, which means the number of mappers is 100, while the number of reducers is only 5. Obviously this is an inadequate configuration as reducers are few. So for more reasonable simulation, more reducers should be added.

Number of reducers: 5 ---> 20

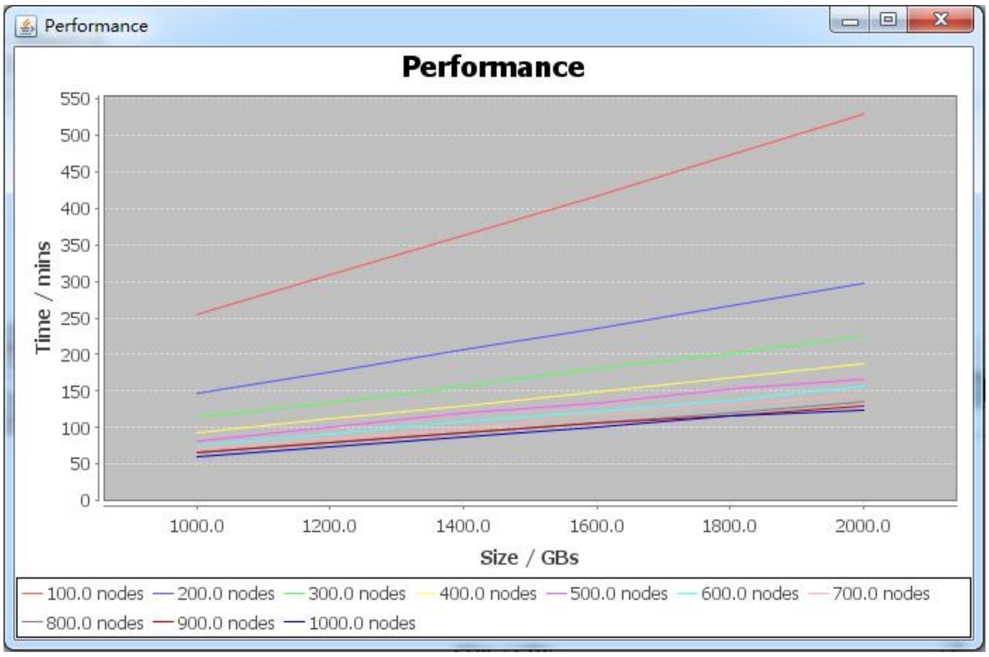


Obviously it gets better. And if we look the log again, it can be noticed that by adding more reducers, the network delay is also shortened. This is because more reducers means there are more receivers in the data transferring process, when the intermediate data is sent from mappers to reducers, and as they work synchronously, the transfer is processed faster.

```
100 nodes & 20 reducers & 2048000.0 GB
Performance: 528.290186664951
     MapTime: 445.62130999992655
     NetworkDelay: 0.9386666666666668
     ReduceTime: 81.73020999835776
```

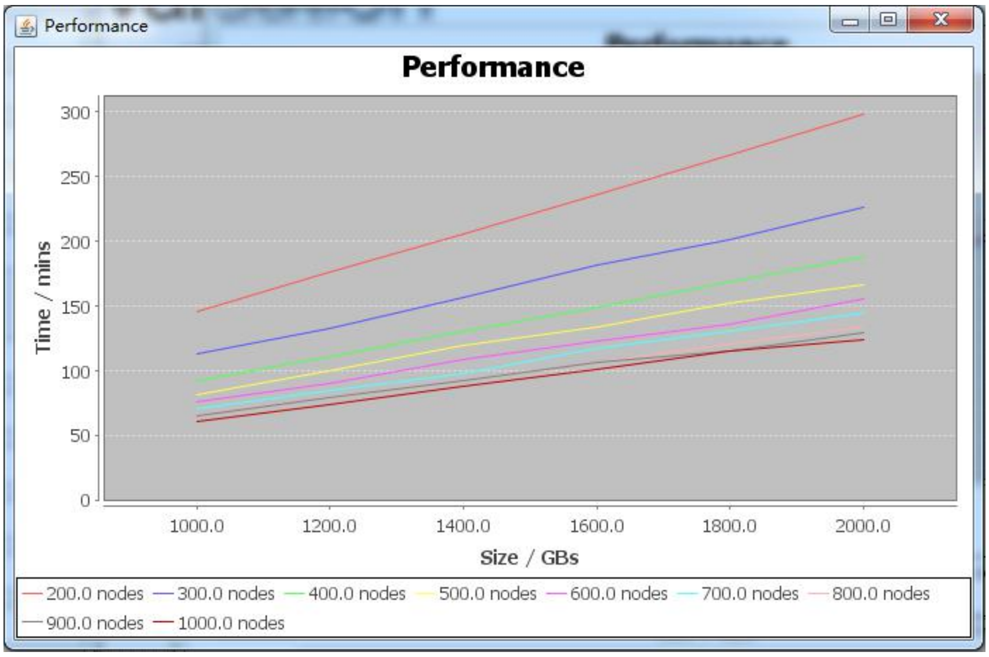
Now the main issue is the map time. Of course it can be improved by fastening I/O and compute time as we did before, but normally it is not possible to improve the CPU according to our will. So the adequate method is to try add more nodes in the system.

Number of nodes: 50 - 100 (increment: 10) ---> 100 - 1000 (increment: 100)



This brings the biggest improvements we have so far, as more nodes means more mappers being created when MapReduce initiates. And for clearly seeing the improved results, we may ignore the performance of 100 nodes and focus on it when using 200 to 1000 nodes.

Number of nodes: 100 - 1000 (increment: 100) ---> 200 - 1000 (increment: 100)

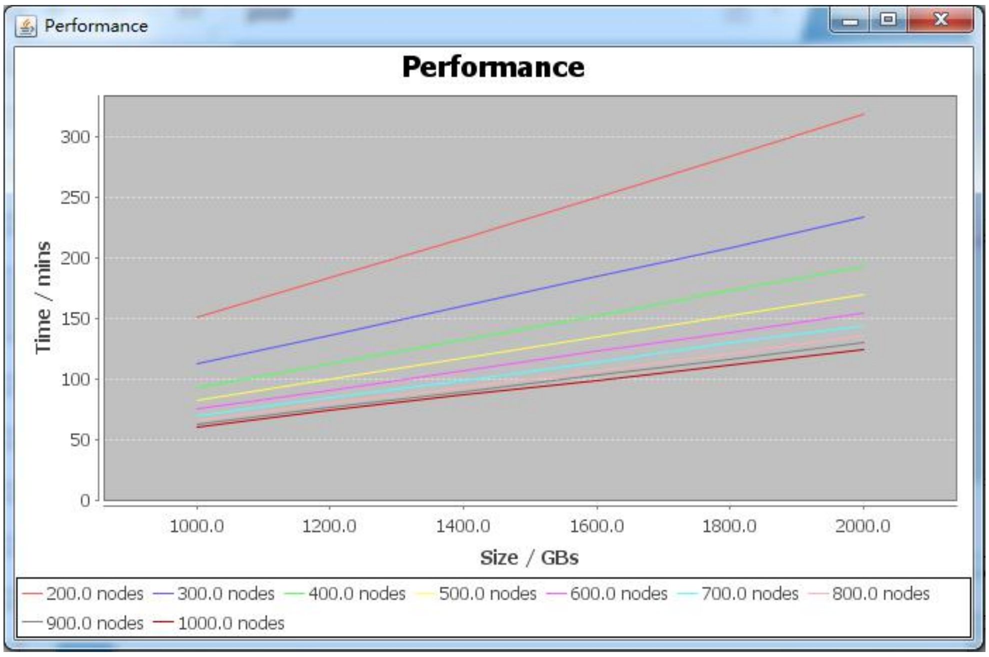


```
1000 nodes & 20 reducers & 2048000.0 GB
Performance: 123.77141784991912
     MapTime: 41.94734118489469
     NetworkDelay: 0.09386666666666668
     ReduceTime: 81.73020999835776
```

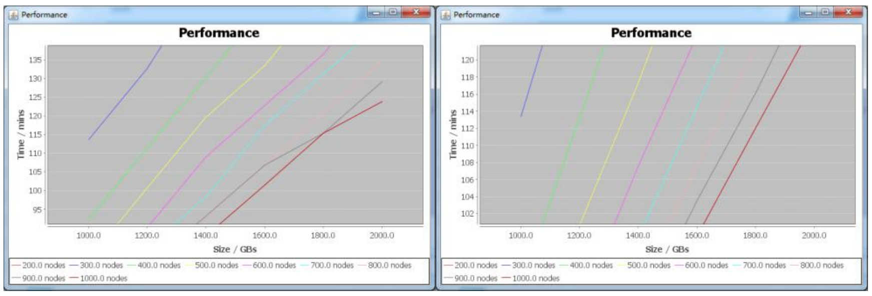
With more nodes, the map time is now shorter than reduce time. So if we want to improve the performance more, again, one effective way is to add more reducers to the system.

Now the diagram is more clear to find out some existing problems, though the performance is already very good. Like mentioned before, there are some point of intersection, such as when testing with 1800 GB data, the performance of 900 nodes and 1000 nodes are same. So the extra 100 nodes play no part in improving the performance. One possible reason is the data block is too large, so the low degree of distribution makes poorly using of the added nodes. By proving this and improving the issue, a testing with smaller data block can be carried out.

Data block size: 256 MB ---> 64MB



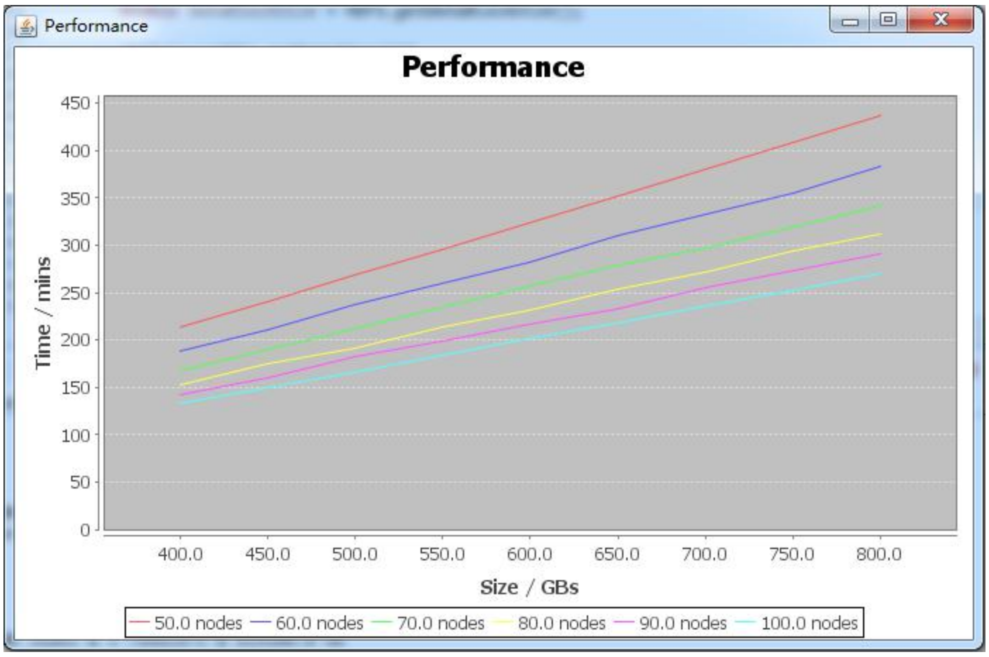
There is no any point of intersection with small data block. To see it clearly, the two screenshots below can be compared. The left one is part of the diagram of performance with 256 MB data block, while the right one uses 64 MB. As more distribute the system becoming, the added nodes can be actually used and improve the performance.



Other factors are used to define the data set, including data size, records bias, and size bias. Obviously they can effect the performance but they are decided by the data set itself. So when testing to improve the performance it is not reasonable to change those factors because it means the simulation is done with different input data.

But we can change it and see the performance of the same configuration processing smaller data set. For example, with the same configuration as we did at first, the performance can be better if the input data is smaller.

Records bias: 1 ---> 0.8
Size bias: 1.1 ---> 1
Data size: 1000 GB - 2000 GB (increment: 200 GB) ---> 400 GB - 800 GB (increment: 50 GB)



Chapter 6

Conclusion

6.1 Project conclusion

Big data has been demonstrated as a viable solution for increasing demands for data storing and processing, based on some existing framework, while Hadoop is one of the most widely used.

As Hadoop combining MapReduce and HDFS, it provides ideas of a new method for data computing. It generates distributed available resources and puts them into a cluster where their infrastructure can be shared with others nodes. This is going to be the new hotspot in data technology, so to effectively and correctively estimate the performance has become an important issue, and will be very useful if a tool is developed to solve this problem. This outcome was the projects initial goal and reaching it has proved the viability of such a simulator.

At the beginning of this project, the goal was to demonstrate the mechanism that MapReduce and HDFS work. This requires numerous studying on materials and researches. All aspects also needs to be re-considered and designed to create the first version of the simulator architecture.

Hadoop provides flexible interfaces which makes it hard to design the simulator in a fixed pattern. So at the first, a very simple model is developed and more components are added into it afterwards. To decide some certain algorithms and strategies is still hard, as there are too many options that can all be referred by Hadoop framework. So the simulator also provides some flexible interfaces to make it possible to change some modules and make prediction with new configuration. The currently work can give the basic results, but more models should still be taken into account and provide more accurate for users.

6.2 Personal conclusion

Throughout this project, I learned many valuable new skills and also applied the theory learned throughout my time in UCC. I gained experience with developing for Java application and have knowledge of big data processing. Critically thinking about design and implementation, along with troubleshooting issues that arose, has made me a stronger software developer.

I personally feel this project is successful, both in honing my planning and development skills, and also my investigative and research skills. I realize that there are many new ideas can come along with this project, which can be done in my future studying. Gaining this motivation helps to keep me learning new things and developing my scope, and also let me have a positive effect on personal growth as a Computer Science student.

On a personal level, I am very happy with this project. Although there are still features to be improved, the project creates tools that can be useful in some actual use cases. The experience of developing the simulator is a driving force behind my motivation to continue working on big data.

6.3 Future work

As has already been mentioned, there are several features which I would like to work on for future version of the simulator.

Firstly, though using an sample programme for experiment, the simulator assumes all models are in linear model, where there is a linear relationship between size and performance. But in real case, it is not always linear, especially when it comes to extreme large data set. So addition method to compute the processing time on each node or generate the model can be put into the simulator.

There also should be more testing to improve the predicted results. It would be better to test using a real cluster with large data set, so the inaccurate results can be found out and improved to provide more adequate prediction.

The simulator considers many factors, but it can not cover all possible situations that may happen in real case. So the simulator can adopt more features and more detailed configuration, like some physical factors have not been counted yet, e.g. memory and disk space. This is another possibility that I am looking at to further improved accuracy.

Bibliography

[1] Big data. URL: https://en.wikipedia.org/wiki/Big_data

[2] The 5 Vs of Big Data. “INTRODUCTION TO BIG DATA: INFRASTRUCTURE AND NETWORKING CONSIDERATIONS” by Shoban Babu Sriramoju

[3] Distributed computing.
URL: https://en.wikipedia.org/wiki/Distributed_computing

[4] Papadimitriou (1994), Chapter 15. Keidar (2008).

[5] Hadoop. Hemsoth, Nicole (2014-10-15). "Cray Launches Hadoop into HPC Airspace". hpcwire.com.

[6] Hadoop architecture.
URL:http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-n ode-cluster

[7] HDFS. URL: https://en.wikipedia.org/wiki/Apache_Hadoop

[8] HDFS architecture.
URL: https://hadoop.apache.org/docs/r1.0.4/hdfs_design.html

[9] MapReduce overview.
URL: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[10] Ambari. URL: https://ambari.apache.org/

[11] Clourdera.
URL: https://www.crunchbase.com/organization/cloudera#section-overview

[11] GIS tools for Hadoop. URL: https://github.com/Esri/gis-tools-for-hadoop

[12] Software architecture. Pattern-Oriented Software Architecture, Vol 1 [Buschmann96]

[13] N-tier architecture. URL: https://en.wikipedia.org/wiki/Multitier_architecture

[14] Time complexity. URL: https://en.wikipedia.org/wiki/Time_complexity

[15] Presentation layer definition. URL: http://www.linfo.org/presentation_layer.html

[16] MapReduce sorting. “Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale” by Tom White

[17] HDFS definition.
URL: https://www.quora.com/What-is-the-difference-between-Hadoop-and-HDFS