

NOVEL APPROACH FOR AUTO TUNING HADOOP CONFIGURATION

*A project report submitted in partial fulfilment of the requirements for
the award of the Degree of*

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

Submitted by

B.MANASWINI
Regd.No.14811A0506

G.SUKANYA
Regd.No.14811A0519

S.NANDINI
Regd.No.14811A0561

P.CHAITANYA
Regd.No.14811A0512

Under the Esteemed Guidance of

Mr. K. VARA PRASAD, M.Tech

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



AVANTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi & Permanently affiliated to JNTU Kakinada)

(Accredited by NAAC, UGC & NBA, AICTE)

MAKAVARAPALEM, NARSIPATNAM,

VISAKHAPATNAM DIST

(2014-2018)

AVANTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

(Approved by AICTE, Permanently affiliated to JNTU Kakinada)

(Accredited by NAAC, UGC & NBA, AICTE)

MAKAVARAPALEM, NARSIPATNAM,

VISAKHAPATNAM-531113



CERTIFICATE

This is to certify that the project entitled **“NOVEL APPROACH FOR AUTO TUNING HADOOP CONFIGURATION”** in partial fulfilment for the of degree of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING**, at **AVANTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY**, MAKAVARAPALEM, VISAKHAPATNAM is an bonafied work carried out by **B.MANASWINI (14811A0506)**, **G.SUKANYA (14811A0519)**, **P.CHAITANYA (14811A0512)** **S.NANDINI (14811A0561)**, under the guidance and supervision during 2017-2018.

PROJECT GUIDE

HEAD OF THE DEPARTMENT

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We consider it as a privilege to thank all those people who have helped for the successful completion of the project “**NOVEL APPROACH FOR AUTO TUNING HADOOP CONFIGURATION**”.

We express our deep sense of gratitude to our Guide & Head of the Department **Mr. K. VARA PRASAD** Department of CSE, Avanathi Institute of Engineering and Technology, Visakhapatnam for his guidance and assistance. We always cherish our association with him for his encouragement, approachability and freedom of thought and action we had enjoyed during this work.

We also express my gratitude to the Principal of Avanathi Institute of Engineering and Technology **Dr. C.P.V.N.J MOHAN RAO** who has given a lot of support and freedom during our project work.

We would also like to thank our College Management for providing various resources like Systems, Internet facilities and other needed requirements to complete our project work successfully.

We would like to thank the faculty, lab staff and all our friends for their good wishes and constructive criticism, which led to the successful completion of this project.

Submitted By

B. MANASWINI (14811A0506)

G. SUKANYA (14811A0519)

P. CHAITANYA (14811A0512)

S. NANDINI DEVI (14811A0561)

DECLARATION

We are hereby declare that the project entitled “**NOVEL APPROACH FOR AUTO TUNING HADOOP CONFIGURATION**” is a bonafied work done by us and submitted for the partial fulfilment of the requirements for the award of degree of **Bachelors of Technology in Computer Science & Engineering** from Jawaharlal Nehru Technological University Kakinada & approved by AICTE is my original work in the year 2017-2018 under the esteemed guidance of **K.VARAPRASAD** in the stream of computer science and engineering department and it is not previously formed. The basis for any degree or diploma or any other similar titled submitted to any university.

Submitted By

B. MANASWINI (14811A0506)

G. SUKANYA (14811A0519)

P. CHAITANYA (14811A0512)

S. NANDINI DEVI (14811A0561)

ABSTRACT

Hadoop is a widely-used implementation framework of the Map Reduce programming model for large-scale data processing. Hadoop performance however is significantly affected by the settings of the Hadoop configuration parameters. Unfortunately, manually tuning these parameters is very time-consuming, if at all practical.

In this project, an approach called RFHOC to automatically tune the Hadoop configuration parameters for optimized performance for a given application running on a given cluster. RFHOC constructs two ensembles of performance models using a random-forest approach for the map and reduce stage respectively. Leveraging these models, RFHOC employs a genetic algorithm to automatically search the Hadoop configuration space. The evaluation of RFHOC using five typical Hadoop programs, each with five different input data sets, shows that it achieves a performance speedup by a factor of 2.11_ on average and up to 7.4_ over the recently proposed cost-based optimization (CBO) approach. In addition, RFHOC's performance benefit increases with input data set size.

Map Reduce is a widely used programming model for processing and generation vast data sets on large scale compute clusters. The Hadoop framework has up to 190 configuration parameters, and overall performance is highly sensitive to the settings of these parameters.

INDEX

SNO	CONTENT	PAGE
1.	INTRODUCTION	1
	1.1 Big Data	
	1.2 Existing System	
	1.3 Proposed System	
2.	LITERATURE SURVEY	12
3.	SYSTEM ANALYSIS	18
	3.1 Feasibility study	
	3.2 Software Requirement Specification	
	3.3 System Requirements	
4.	SYSTEM DESIGN	26
	4.1 System Architecture	
	4.2 Data Flow Diagram	
	4.3 UML Diagram	
	4.3.1 Use cases	
	4.3.2 Class Diagram	
	4.3.3 Sequence Diagram	
	4.3.4 Activity Diagram	
5.	LANGUAGE SPECIFICATION	26

6.	METHODOLOGY	39
	6.1 Modules	
	6.2 Language Implementation	
7.	RESULTS	50
8.	SYSTEM TESTING	59
	8.1 Types of tesing	
	8.2 Test Cases	
9.	CONCLUSION	65
10.	FUTURE ENHANCEMENT	67
11.	BIBLIOGRAPHY	69

INTRODUCTION

INTRODUCTION

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a map procedure (or method), which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a reduce method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

The model is a specialization of the split-apply-combine strategy for data analysis. It is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. The key contributions of the MapReduce framework are not the actual map and reduce functions but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine. As such, a single-threaded implementation of MapReduce will usually not be faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with multi-threaded implementations. The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play. Optimizing the communication cost is essential to a good MapReduce algorithm.

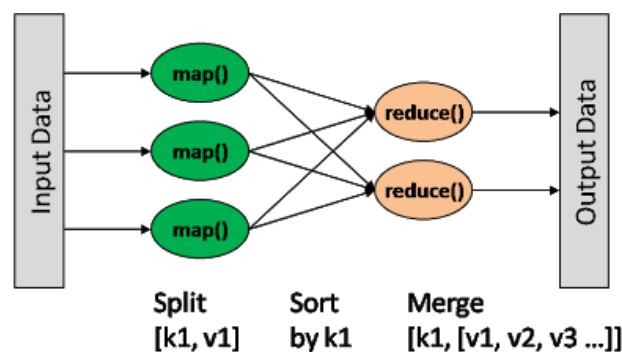


Figure 1.1: MODEL OF MAP REDUCE

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop. The name MapReduce originally referred to the proprietary Google technology, but has since been genericized. By 2014, Google was no longer using MapReduce as their primary *big data* processing model, and development on Apache Mahout had moved on to more capable and less disk-oriented mechanisms that incorporated full map and reduce capabilities.

MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to minimize communication overhead.

A MapReduce framework (or system) is usually composed of three operations (or steps):

1. **Map:** each worker node applies the `map` function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
2. **Shuffle:** worker nodes redistribute data based on the output keys (produced by the `map` function), such that all data belonging to one key is located on the same worker node.
3. **Reduce:** worker nodes now process each group of output data, per key, in parallel.

MapReduce allows for distributed processing of the map and reduction operations. Maps can be performed in parallel, provided that each mapping operation is independent of the others; in practice, this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential (because multiple instances of the reduction process must be run), MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours.^[13] The

parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

1. **Prepare the Map() input** – the "MapReduce system" designates Map processors, assigns the input key value $K1$ that each processor would work on, and provides that processor with all the input data associated with that key value.
2. **Run the user-provided Map() code** – Map() is run exactly once for each $K1$ key value, generating output organized by key values $K2$.
3. **"Shuffle" the Map output to the Reduce processors** – the MapReduce system designates Reduce processors, assigns the $K2$ key value each processor should work on, and provides that processor with all the Map-generated data associated with that key value.
4. **Run the user-provided Reduce() code** – Reduce() is run exactly once for each $K2$ key value produced by the Map step.
5. **Produce the final output** – the MapReduce system collects all the Reduce output, and sorts it by $K2$ to produce the final outcome.

These five steps can be logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected.

In many situations, the input data might already be distributed ("sharded") among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the Map-generated data they need to process.

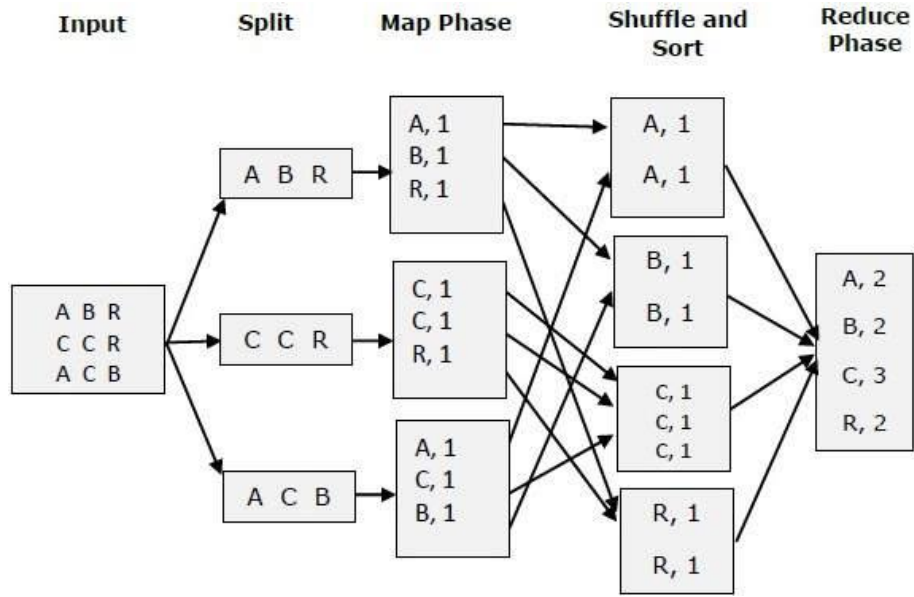


Figure1.2: PHASES OF MAP REDUCE

The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. *Map* takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

$$\text{Map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$$

The Map function is applied in parallel to every pair (keyed by k_1) in the input dataset. This produces a list of pairs (keyed by k_2) for each call. After that, the MapReduce framework collects all pairs with the same key (k_2) from all lists and groups them together, creating one group for each key.

The *Reduce* function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$$\text{Reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}(v_3)$$

Each *Reduce* call typically produces either one value v_3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values. This behavior is different from the typical functional programming map and reduce combination, which accepts a list of arbitrary values and returns one single value that combines *all* the values returned by map.

1.2 EXISTING SYSTEM

- A naive approach to find the optimum Hadoop configuration is to try every combination of configuration parameter values and choose the best one. Unfortunately, this is unrealistic because of the huge number of Hadoop configuration parameter combinations.
- To make things even worse, every run of a Hadoop application with a large input data set takes a considerable amount of time, leading to impractically long times if one were to explore the huge Hadoop configuration parameter optimization space exhaustively.
- Performance models can predict the performance of an application with a given configuration much faster than an approach that requires executing the application.
- Herodotou et al. propose fine-grained analytical models to predict the execution time of each phase; putting together per-phase predictions then yields an estimate for the total execution time.
- Lama and Zhou build a support vector machines (SVM) based model to predict the performance of Hadoop jobs.

DISADVANTAGES OF EXISTING SYSTEM:

- These analytical performance models are based on oversimplified assumptions, which limits overall performance.
- Previously proposed analytical models typically assume the execution time of per-byte or per-record processing to be constant, even as Hadoop configurations change, statistical models such as linear regression algorithms typically assume that the relationship between configuration parameters is linear.

1.3 PROPOSED SYSTEM:

- In this project, we propose a novel approach based on random-forest learning, which we call RFHOC, to predict the performance of an application on a given cluster with a given Hadoop configuration.
- Strictly speaking, random-forest is not a machine learning algorithm; instead, it is a robust ensemble model that combines the advantages of statistical reasoning and machine learning approaches.
- We consider a number of observations from a real Hadoop system to train an ensemble model for each phase via random forest learning. The model takes Hadoop configurations as input and outputs a performance prediction. In a subsequent step, we then use the performance prediction models for each phase as part of a genetic algorithm (GA) to search for the optimum Hadoop configuration for the application of interest.

ADVANTAGES OF PROPOSED SYSTEM:

- RFHOC does not make any assumptions on the cost (execution time) of per-byte or per-record processing and the relationship between configuration parameters.
- RFHOC on the other hand does not make these simplifying assumptions and recognizes that Hadoop configuration parameters interact with each other in complex non-linear ways.
- Second, random-forest learning makes predictions based on a set of regression or classification trees, rather than a single tree, which makes the performance prediction not only accurate, but also stable and robust when deployed on previously unseen data sets.

LITERATURE SURVEY

2. LITERATURE SURVEY

1) Heuristic artificial intelligent algorithm for genetic algorithm.

AUTHORS: L. Lie

A genetic algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

2) Jaql: A scripting language for large scale semi structured data analysis.

AUTHORS: K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh,

C.-C. Kanne, F. Ozcan, and E. J. Shekita

This paper describes Jaql, a declarative scripting language for analyzing large semistructured datasets in parallel using Hadoop's MapReduce framework. Jaql is currently used in IBM's InfoSphere BigInsights and Cognos Consumer Insight products. Jaql's design features are: a flexible data model, reusability, varying levels of abstraction, and scalability. Jaql's data model is inspired by JSON and can be used to represent datasets that vary from flat, relational tables to collections of semistructured documents. A Jaql script can start without any schema and evolve over time from a partial to a rigid schema. Reusability is provided through the use of higher-order functions and by packaging related functions into modules. Most Jaql scripts work at a high level of abstraction for concise specification of logical operations (e.g., join), but Jaql's notion of physical transparency also provides a lower level of abstraction if necessary. This allows users to pin down the evaluation plan of a script for greater control or even add new Koperators. The Jaql compiler automatically rewrites Jaql scripts so they can run in parallel on Hadoop. In addition to describing Jaql's design, we present the results of scale-up experiments on Hadoop running Jaql scripts for intranet data analysis and log processing.

3) An integrated home financial investment learning environment applying cloud computing in social network analysis.

AUTHORS: M.-P. Wen, H.-Y. Lin, A.-P. Chen, and C. Yang

This paper tried to apply cloud computing technology in social network analysis for a comprehensive home financial learning environment that individual investors may use as a reference in establishing web-based learning and investment platforms. The major contributions were described in three parts. First, this paper advanced the social network analysis technology to be able to handle millions of nodes and links. Second, we demonstrate how cloud computing can be applied to advanced computing in social network. Third, we performed several intelligent analyses on a very popular social network, IHFILE, to identify some interesting and important features of it. In addition to analyzing a homogeneous social network such as IHFILE, we also propose direction of how cloud computing can be performed on a social network analysis as our future work.

4) Starfish: A self-tuning system for big data analytics.

AUTHORS: H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu

Timely and cost-effective analytics over “Big Data ” is now a key ingredient for success in many businesses, scientific and engineering disciplines, and government endeavors. The Hadoop software stack—which consists of an extensible MapReduce execution engine, pluggable distributed storage engines, and a range of procedural to declarative interfaces is a popular choice for big data analytics. Most practitioners of big data analytics like computational scientists, systems researchers, and business analysts lack the expertise to tune the system to get good performance. Unfortunately, Hadoop’s performance out of the box leaves much to be desired, leading to suboptimal use of resources, time, and money (in pay-as-you-go clouds). We introduce Starfish, a self-tuning system for big data analytics. Starfish builds on Hadoop while adapting to user needs and system workloads to provide good performance automatically, without any need for users to understand and manipulate the many tuning knobs in Hadoop. While Starfish’s system architecture is guided by work

5) Profiling, What-if analysis, and cost-based optimization of MapReduce programs

AUTHORS: H. Herodotou and S. Babu

MapReduce has emerged as a viable competitor to database systems in big data analytics. MapReduce programs are being written for a wide variety of application domains including business data processing, text analysis, natural language processing, Web graph and social network analysis, and computational science. However, MapReduce systems lack a feature that has been key to the historical success of database systems, namely, cost-based optimization. A major challenge here is that, to the MapReduce system, a program consists of black-box map and reduce functions written in some programming language like C++, Java, Python, or Ruby. We introduce, to our knowledge, the first Cost-based Optimizer for simple to arbitrarily complex MapReduce programs. We also introduce a Profiler to collect detailed statistical information from unmodified MapReduce programs, and a What-if Engine for fine-grained cost estimation. The effectiveness of each component is demonstrated through a comprehensive evaluation using representative MapReduce programs from various application domains.

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 FESABILITY STUDY:

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance

by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.2 SOFTWARE REQUIREMENTS SPECIFICATIONS:

- The purpose of Software Requirements Specifications is to reduce the communication gap between the clients and the developers.
- A good SRS should satisfy all the parties involved in the system.
- Having a clear distinction between the needs of the clients, any software can be developed which will satisfy the client's requirement meeting his specifications.

Functional requirements:

Functional requirements describe what the system should do, i.e. the services provided for the users and for the other systems.

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process

and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2. Select methods for presenting information.

3. Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- ❖ Convey information about past activities, current status or projections of the
- ❖ Future.

- ❖ Signal important events, opportunities, problems, or warnings.
- ❖ Trigger an action.
- ❖ Confirm an action.

Non-Functional Requirements:

Scalability:

Scalability, is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged.

Reliability:

Reliability (systemic def.) is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances

Integrity:

Integrity as a concept has to do with perceived consistency of actions, values, methods, measures, principles, expectations and outcome.

Extensibility:

The System should be extendible with other application techniques

Reusability:

The System should be 60% reusable so as to be applied for other datasets.

3.3 SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

- System : i3 Processor
- Hard Disk : 500 GB.
- Monitor : 15” LED
- Input Devices : Keyboard, Mouse
- Ram : 4GB.

SOFTWARE REQUIREMENTS:

- Operating system : Windows 7/UBUNTU.
- Coding Language : Java 1.7 ,Hadoop 0.8.1 (for Mapper and Reducer)
- Back End : Hadoop Cluster
- Tool : Virtual Box Oracle tool
- Evaluation : PHP, Javascript (Intelligent Graph)

SYSTEM DESIGN

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

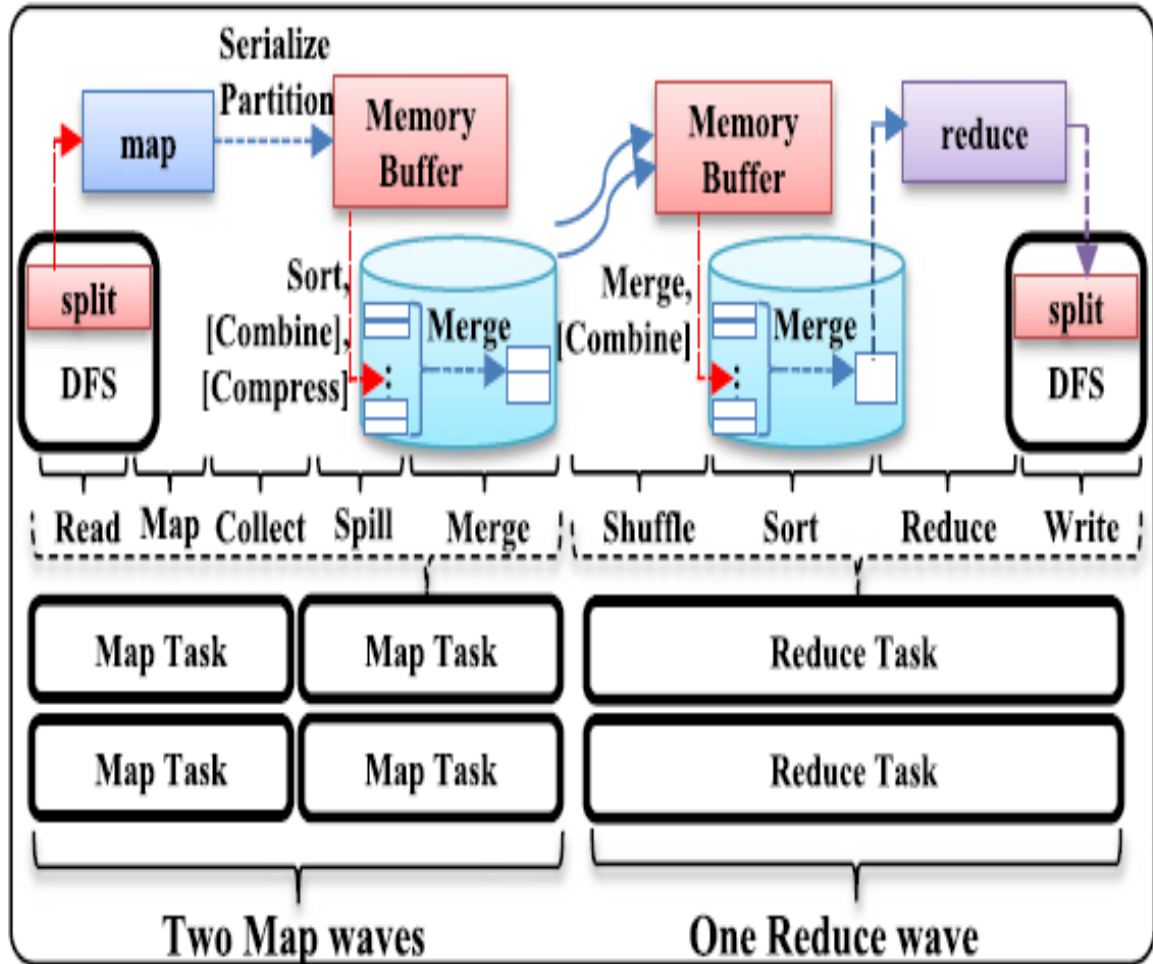
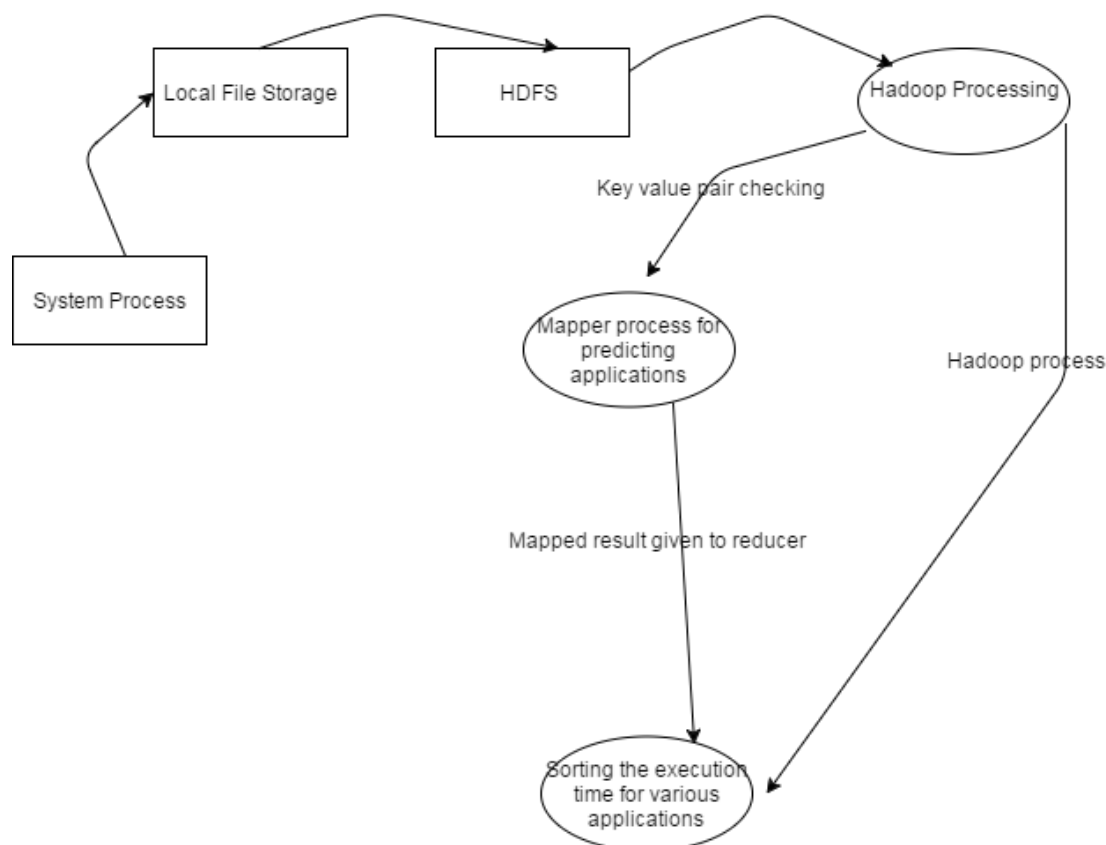


Fig: System Architecture

We first describe the MapReduce programming model and we provide motivation and background for using random forest learning to optimize performance.

4.2 DATA FLOW DIAGRAM

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.



4.3 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks.
7. Integrate best practices.

4.3.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. It gives a simple and precise form of end to end flow of process.

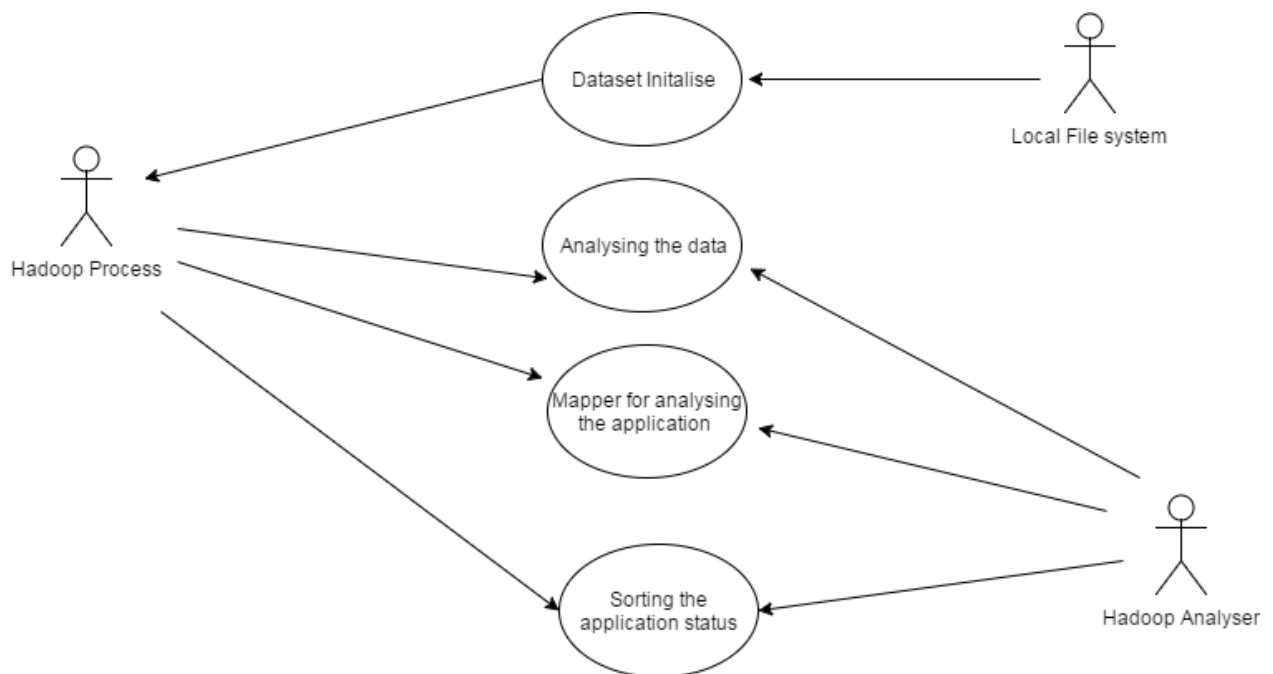


FIG: USECASE DIAGRAM

4.3.2 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

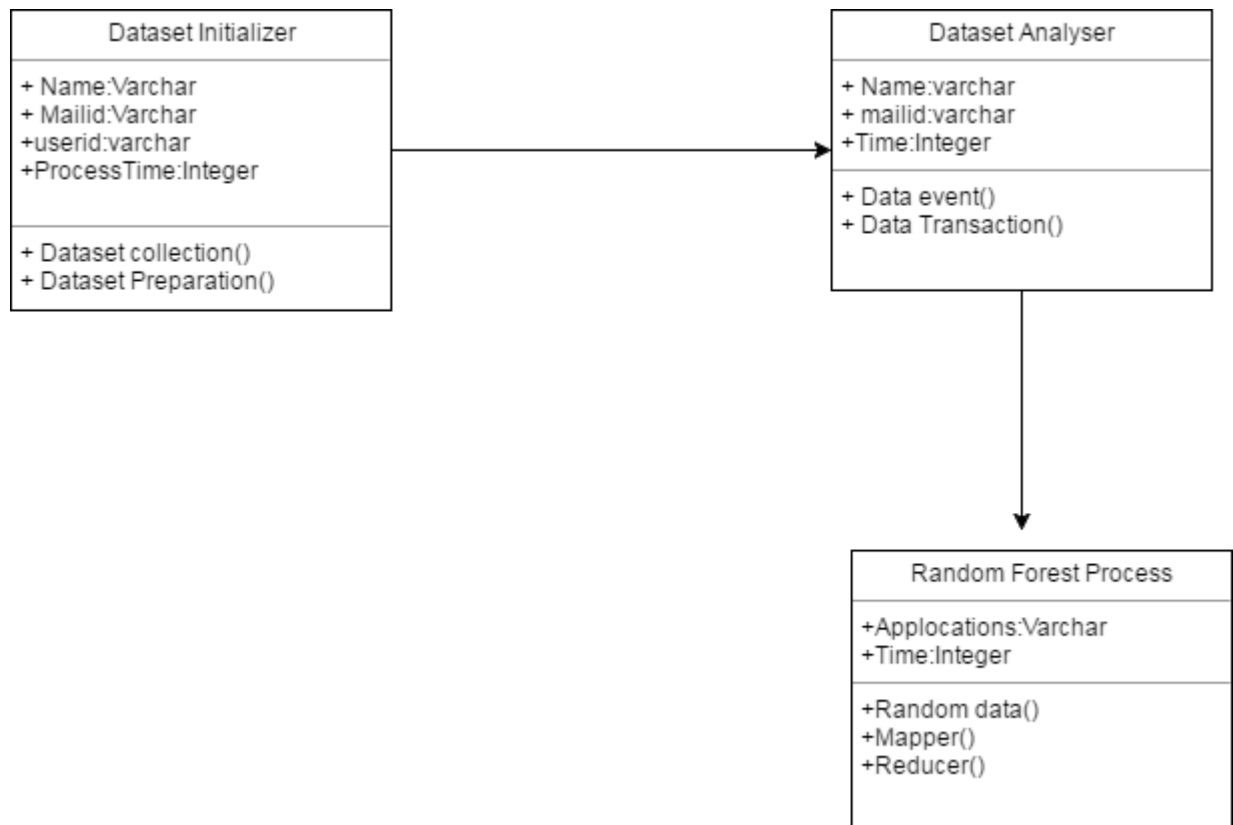


FIG: CLASS DIAGRAM

4.3.3 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

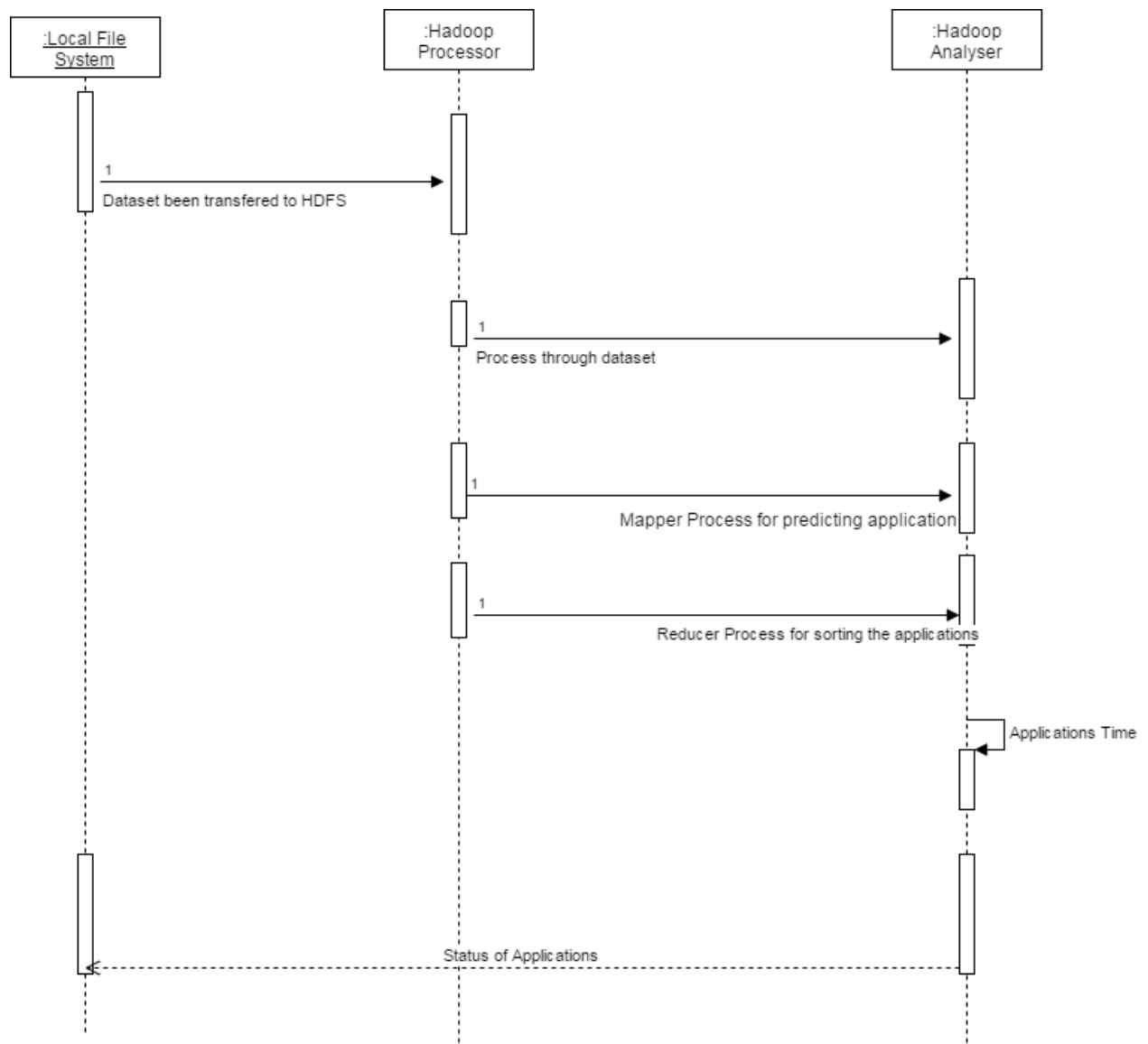


FIG: SEQUENCE DIAGRAM

4.3.4 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

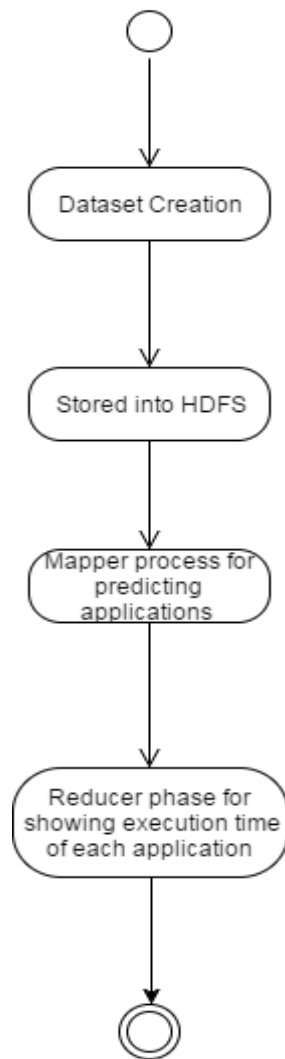


FIG: ACTIVITY DIAGRAM

LANGUAGE SPECIFICATION

5. LANGUAGE SPECIFICATION

Java Technology

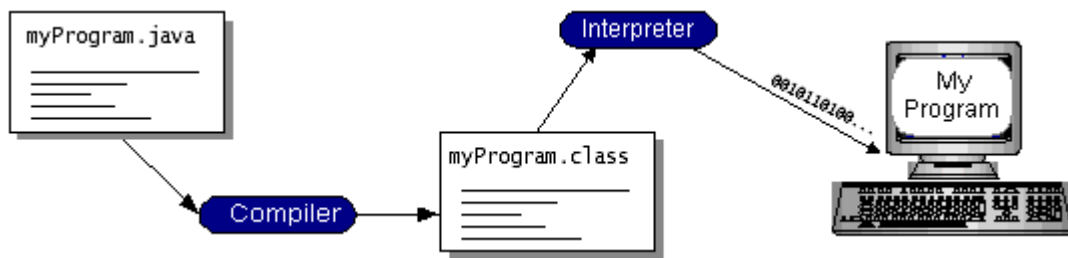
Java technology is both a programming language and a platform.

The Java Programming Language

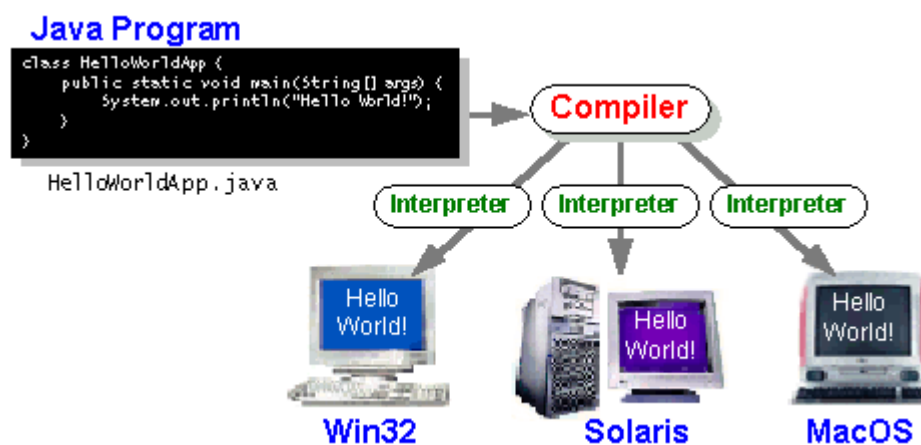
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes*—the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

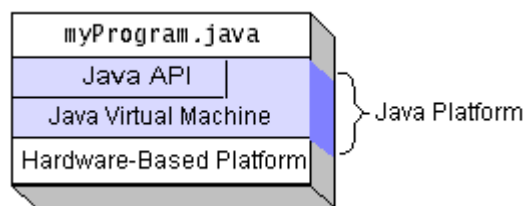
The Java platform has two components:

- *The Java Virtual Machine (Java VM)*
- *The Java Application Programming Interface (Java API)*

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

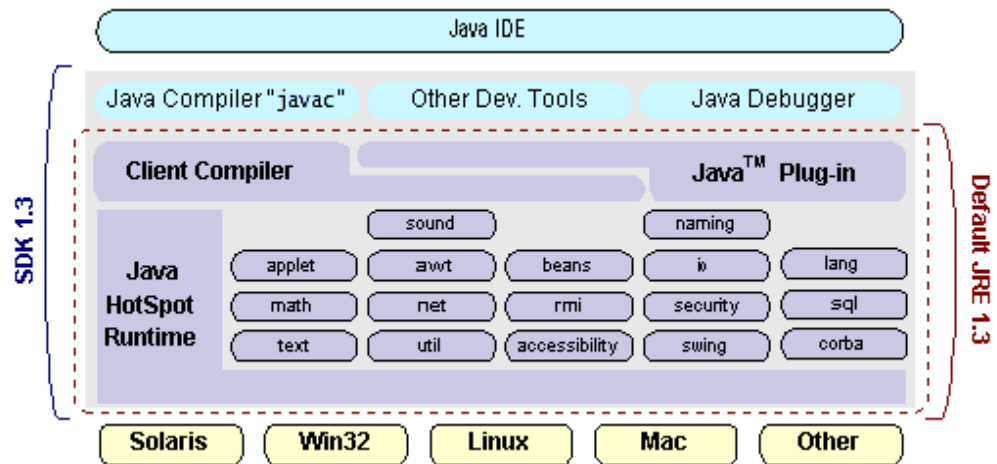
However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials:** Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets:** The set of conventions used by applets.
- **Networking:** URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization:** Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security:** Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components:** Known as JavaBeansTM, can plug into existing component architectures.
- **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBCTM):** Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



How Will Java Technology Change My Life?

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by avoiding the use of libraries written in other

languages. The 100% Pure Java Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.

- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
- **Distribute software more easily:** You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

MIDP contains the following packages, the first three of which are core CLDC packages, plus three MIDP-specific packages.

* java.lang

* java.io

* java.util

* javax.microedition.io

* javax.microedition.lcdui

HADOOP

Introduction

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. 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 relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is part of the Apache Hadoop Core project.

Assumptions and Goals

Hardware Failure

Hardware failure is the norm rather than the exception. 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.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the Name Node or the Data Node software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the Name Node software. Each of the other machines in the cluster runs one instance of the Data Node software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

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. HDFS does not yet implement user quotas or access permissions. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

Data Replication

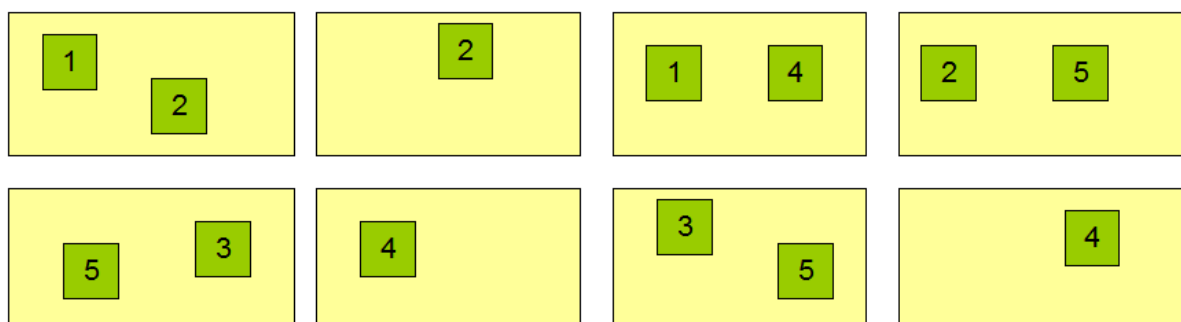
HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable 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 Name Node makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Block report from each of the Data Nodes in the cluster. Receipt of a Heartbeat implies that the Data Node is functioning properly. A Block report contains a list of all blocks on a Data Node.

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



Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead DataNode is not available to HDFS any more. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased.

Cluster Rebalancing

The HDFS architecture is compatible with data rebalancing schemes. A scheme might automatically move data from one DataNode to another if the free space on a DataNode falls below a certain threshold. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

Data Integrity

It is possible that a block of data fetched from a DataNode arrives corrupted. This corruption can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements checksum checking on the contents of HDFS files. When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace.

Metadata Disk Failure

The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the NameNode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of the FsImages and EditLogs to get updated synchronously. This synchronous updating of multiple copies of the FsImage and EditLog may degrade the rate of namespace transactions per second that a NameNode can support. However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive. When a NameNode restarts, it selects the latest consistent FsImage and EditLog to use.

Snapshots

Snapshots support storing a copy of data at a particular instant of time. One usage of the snapshot feature may be to roll back a corrupted HDFS instance to a previously known good point in time. HDFS does not currently support snapshots but will in a future release.

Data Organization

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.

Accessibility

HDFS can be accessed from applications in many different ways. Natively, HDFS provides a FileSystem Java API for applications to use. A C language wrapper for this Java API is also available. In addition, an HTTP browser can also be used to browse the files of an HDFS instance. Work is in progress to expose HDFS through the WebDAV protocol.

FS Shell

HDFS allows user data to be organized in the form of files and directories. It provides a commandline interface called FS shell that lets a user interact with the data in HDFS. The syntax of this command set is similar to other shells (e.g. bash, csh) that users are already familiar with. Here are some sample action/command pairs:

Action	Command
Create a directory named /foodir	bin/hadoop dfs -mkdir /foodir
Remove a directory named /foodir	bin/hadoop fs -rm -R /foodir
View the contents of a file named /foodir/myfile.txt	bin/hadoop dfs -cat /foodir/myfile.txt

FS shell is targeted for applications that need a scripting language to interact with the stored data.

DFS Admin

The DFS Admin command set is used for administering an HDFS cluster. These are commands that are used only by an HDFS administrator. Here are some sample action/command pairs:

Action	Command
Put the cluster in Safemode	bin/hdfs dfsadmin -safemode enter
Generate a list of DataNodes	bin/hdfs dfsadmin -report
Recommission or decommission DataNode(s)	bin/hdfs dfsadmin -refreshNodes

Browser Interface

A typical HDFS install configures a web server to expose the HDFS namespace through a configurable TCP port. This allows a user to navigate the HDFS namespace and view the contents of its files using a web browser.

Space Reclamation

File Deletes and Undeletes

When a file is deleted by a user or an application, it is not immediately removed from HDFS. Instead, HDFS moves it to a trash directory (each user has its own trash directory under `/user/<username>/.Trash`). The file can be restored quickly as long as it remains in trash. Most recent deleted files are moved to the current trash directory (`/user/<username>/.Trash/Current`), and in a configurable interval, HDFS creates checkpoints.

(under `/user/<username>/.Trash/<date>`) for files in current trash directory and deletes old checkpoints when they are expired. After the expiry of its life in trash, the NameNode deletes the file from the HDFS namespace. The deletion of a file causes the blocks associated with the file to be freed. Note that there could be an appreciable time delay between the time a file is deleted by a user and the time of the corresponding increase in free space in HDFS.

Currently, the trash feature is disabled by default (deleting files without storing in trash). User can enable this feature by setting a value greater than zero for parameter `fs.trash.interval` (in `core-site.xml`). This value tells the NameNode how long a checkpoint will be expired and removed from HDFS. In addition, user can configure an appropriate time to tell NameNode how often to create checkpoints in trash (the parameter stored as `fs.trash.checkpoint.interval` in `core-site.xml`), this value should be smaller or equal to `fs.trash.interval`.

METHODOLOGY

6. METHODOLOGY

6.1 MODULES

Random forest is an ensemble model that can be used

- Random Forest
- Analytical Models
- Resource consumption

MODULES DESCRIPTION:

Random Forest

Both classification and regression. It operates by constructing a multitude of decision trees at training time; prediction then combines the outputs by the individual trees to arrive at the final output (e.g., majority voting for classification, and average for regression). A key feature of random forests is that they correct for decision trees' tendency to overfit to their training data.

Analytical Models

Although the analytical performance models are fine-grained at the level of Map Reduce phases, they assume the execution time per operation to depend on the cluster resources only. In addition, they further assume it to be constant across different Hadoop configurations. To verify whether the above assumption holds true, we take the same 10 Hadoop configuration parameters and we assign random values to them, forming six different Hadoop configurations,

Resource Consumption:

Support Vector Machines (SVM) to build a model called AROMA to predict the performance of Hadoop applications with different configurations. AROMA is based on the observation that jobs with similar resource consumption characteristics exhibit similar performance behavior across Hadoop configurations. This observation might hold true if the performance models were constructed for very fine-grained objects. However, the

amount of CPU, I/O, and network consumed at the Hadoop job level to identify the resource consumption characteristics, which is a coarse-grained approach.

AROMA groups the sort and word count benchmarks in the same group because they consume similar cluster resources. To verify this assumption, we conduct an experiment to observe the resource consumption for both benchmarks at the phase level.

6.2 LANGUAGE IMPLEMENTATION

Random forest:

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class RandomForest {
    private static int NUM_THREADS;//=Runtime.getRuntime().availableProcessors();
        public static int C;

        public static int M;

        public static int Ms;//recommended by Breiman:
        =(int)Math.round(Math.log(M)/Math.log(2)+1);

        private long time_o;

        private int numTrees;

        private double update;
        private double progress;
        private int[] importances;
        private HashMap<int[],int[]> estimateOOB;
            private double error;
        private ExecutorService treePool;
        private ArrayList<ArrayList<String>> data;
        private ArrayList<ArrayList<String>> testdata;
        private ArrayList<ArrayList<String>> Prediction;
    public ArrayList<Character> DataAttributes;
        public HashMap<ArrayList<String>, String> FinalPredictions;
```

```
public int corretsPredictions;
```

```
    @SuppressWarnings("static-access")
    public RandomForest(ArrayList<Character> dataLayout,int numTrees,int
numThreads,int M,int Ms,int C, ArrayList<ArrayList<String>>
train,ArrayList<ArrayList<String>> test) {
        // TODO Auto-generated constructor stub
        StartTimer();
        this.numTrees=numTrees;
        this.NUM_THREADS=numThreads;
        this.data=train;
        this.testdata=test;
        this.M=M;
        this.Ms=Ms;
        this.C=C;
        this.DataAttributes = dataLayout;

        trees2 = new ArrayList<DecisionTree>(numTrees);
        update=100/((double)numTrees);
        progress=0;
        corretsPredictions =0;
        System.out.println("creating "+numTrees+" trees in a random Forest. . .");
        System.out.println("total data size is "+train.size());
        System.out.println("number of attributes "+M);
        System.out.println("number of selected attributes "+Ms);

        estimateOOB=new HashMap<int[],int[]>(data.size());
        Prediction = new ArrayList<ArrayList<String>>();
        FinalPredictions = new HashMap<ArrayList<String>, String>();

    }

    @SuppressWarnings("unchecked")
```

```

public void Start(boolean forAccuracy,boolean withThreads) {
    // TODO Auto-generated method stub
    System.out.println("Number of threads started : "+NUM_THREADS);
    System.out.print("Starting trees");
    treePool=Executors.newFixedThreadPool(NUM_THREADS);
    for (int t=0;t<numTrees;t++){
        treePool.execute(new CreateTree(data,this,t+1));
    }treePool.shutdown();
    try {
        treePool.awaitTermination(10,TimeUnit.SECONDS); //effectively
infinity
    } catch (InterruptedException ignored){
        System.out.println("interrupted exception in Random Forests");
    }
    System.out.println("Trees      Production      completed      in
"+TimeElapsed(time_o));

    if(forAccuracy){
        if(withThreads){
            System.out.println("Testing  Forest  for  Accuracy  with
threads");
            ArrayList<DecisionTree>      Tree1      =
(ArrayList<DecisionTree>) trees2.clone();
            TestforAccuracy(Tree1,testdata,data);
        }else{
            System.out.println("Testing  Forest  for  Accuracy  without
threads");
            ArrayList<DecisionTree>      Tree2      =
(ArrayList<DecisionTree>) trees2.clone();
            TestForestForAccuracy(Tree2, data, testdata);
        }
    }else{
        if(withThreads){

```

```

        System.out.println("Testing Forest for Labels with threads");
        ArrayList<DecisionTree> Tree3 =
(ArrayList<DecisionTree>) trees2.clone();
        TestForestForLabelWT(Tree3, data, testdata);
    }else{
        System.out.println("Testing Forest for Labels without
threads");

        ArrayList<DecisionTree> Tree4 =
(ArrayList<DecisionTree>) trees2.clone();
        TestForestForLabel(Tree4, data, testdata);
    }
}

private void TestforAccuracy(ArrayList<DecisionTree>
trees,ArrayList<ArrayList<String>> Testdata,ArrayList<ArrayList<String>> TrainData)
{
    long time2 = System.currentTimeMillis();
    ExecutorService TestthreadPool =
Executors.newFixedThreadPool(NUM_THREADS);

    for(ArrayList<String> TP:Testdata){
        TestthreadPool.execute(new TestTree(TP,trees,TrainData));
    }TestthreadPool.shutdown();
    try{
        TestthreadPool.awaitTermination(10, TimeUnit.SECONDS);
    }catch(InterruptedException ignored){
        System.out.print("Interuption in testing");
    }System.out.println("Testing Complete");

    System.out.println("Results are ...");
    System.out.println("Forest Accuracy is
"+((corretsPredictions*100)/Testdata.size())+"%");
}

```

```

        System.out.println("this test was done in "+TimeElapsed(time2));
        System.out.println("");System.out.println("");

    }

    private void TestForestForLabel(ArrayList<DecisionTree>
trees,ArrayList<ArrayList<String>> traindata,ArrayList<ArrayList<String>> testdata) {
        // TODO Auto-generated method stub
        long time = System.currentTimeMillis();
        int treee=1;
        System.out.println("Predicting Labels now");
        for(DecisionTree DTC : trees){
            DTC.CalculateClasses(traindata, testdata, treee);treee++;
            if(DTC.predictions!=null)
                Prediction.add(DTC.predictions);
        }
        for(int i = 0;i<testdata.size();i++){
            ArrayList<String> Val = new ArrayList<String>();
            for(int j=0;j<trees.size();j++){
                Val.add(Prediction.get(j).get(i));
            }
            String pred = ModeofList(Val);
            System.out.println("[ "+pred+"]: Class predicted for data point:
"+i+1);
        }
        System.out.println("this test was done in "+TimeElapsed(time));
    }

    private void TestForestForLabelWT(ArrayList<DecisionTree>
tree,ArrayList<ArrayList<String>> traindata,ArrayList<ArrayList<String>> testdata) {
        long time = System.currentTimeMillis();
        ExecutorService TestthreadPool =
Executors.newFixedThreadPool(NUM_THREADS);int i=1;
        for(ArrayList<String> TP:testdata){

```

```

        TestthreadPool.execute(new
TestTreeforLabel(TP,tree,traindata,i));i++;
    }TestthreadPool.shutdown();
    try{
        TestthreadPool.awaitTermination(10, TimeUnit.SECONDS);
    }catch(InterruptedException ignored){
        System.out.print("Interuption in testing");
    }
    System.out.println("Testing Complete");
    System.out.println("this test was done in "+TimeElapsed(time));
}

    public      void      TestForestForAccuracy(ArrayList<DecisionTree>
trees,ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test){
        long time = System.currentTimeMillis();
        int      correctness=0;ArrayList<String>      ActualValues      =      new
ArrayList<String>();

        for(ArrayList<String> s:test){
            ActualValues.add(s.get(s.size()-1));
        }int treee=1;
        System.out.println("Testing forest now ");

        for(DecisionTree DTC : trees){
            DTC.CalculateClasses(train, test, treee);treee++;
            if(DTC.predictions!=null)
                Prediction.add(DTC.predictions);
        }
        for(int i = 0;i<test.size();i++){
            ArrayList<String> Val = new ArrayList<String>();
            for(int j=0;j<trees.size();j++){
                Val.add(Prediction.get(j).get(i));
            }
            String pred = ModeofList(Val);

```

```

        if(pred.equalsIgnoreCase(ActualValues.get(i))){
            correctness = correctness +1;
        }
    }
    System.out.println("The Result of Predictions :-");
    System.out.println("Total Cases : "+test.size());
    System.out.println("Total CorrectPredicions : "+correctness);
    System.out.println("Forest Accuracy
:"+(correctness*100/test.size())+"%");
    System.out.println("this test was done in "+TimeElapsed(time));
}

```

```

public String ModeofList(ArrayList<String> predictions) {
    // TODO Auto-generated method stub
    String MaxValue = null; int MaxCount = 0;
    for(int i=0;i<predictions.size();i++){
        int count=0;
        for(int j=0;j<predictions.size();j++){

            if(predictions.get(j).trim().equalsIgnoreCase(predictions.get(i).trim()))
                count++;
            if(count>MaxCount){
                MaxValue=predictions.get(i);
                MaxCount=count;
            }
        }
    }return MaxValue;
}

```

```

private class CreateTree implements Runnable{
    private ArrayList<ArrayList<String>> data;
    private RandomForest forest;
    private int treenum;

```

```

        public CreateTree(ArrayList<ArrayList<String>> data, RandomForest
forest, int num){
            this.data=data;
            this.forest=forest;
            this.treenum=num;
        }
    public class TestTreeforLabel implements Runnable{
        public ArrayList<String> testrecord;
        public ArrayList<DecisionTree> Trees;
        public ArrayList<ArrayList<String>> trainData;
        public int point;
        public TestTreeforLabel(ArrayList<String> dp, ArrayList<DecisionTree>
dtree, ArrayList<ArrayList<String>> data, int i){
            this.testrecord = dp;
            this.Trees = dtree;
            this.trainData = data;
            this.point =i;
        }

        @Override
        public void run() {
            ArrayList<String> predictions = new ArrayList<String>();

            for(DecisionTree DT:Trees){
                String Class = DT.Evaluate(testrecord, trainData);
                if(Class == null)
                    predictions.add("n/a");
                else
                    predictions.add(Class);
            }

            String finalClass = ModeofList(predictions);

```



```

        System.out.println("[ "+finalClass+"]: Class predicted for data point:
"+point);
    }
}

private void StartTimer(){
    time_o=System.currentTimeMillis();
}

private static String TimeElapsed(long timeinms){
    double s=(double)(System.currentTimeMillis()-timeinms)/1000;
    int h=(int)Math.floor(s/((double)3600));
    s-=(h*3600);
    int m=(int)Math.floor(s/((double)60));
    s-=(m*60);
    return ""+h+"hr "+m+"m "+s+"sec";
}

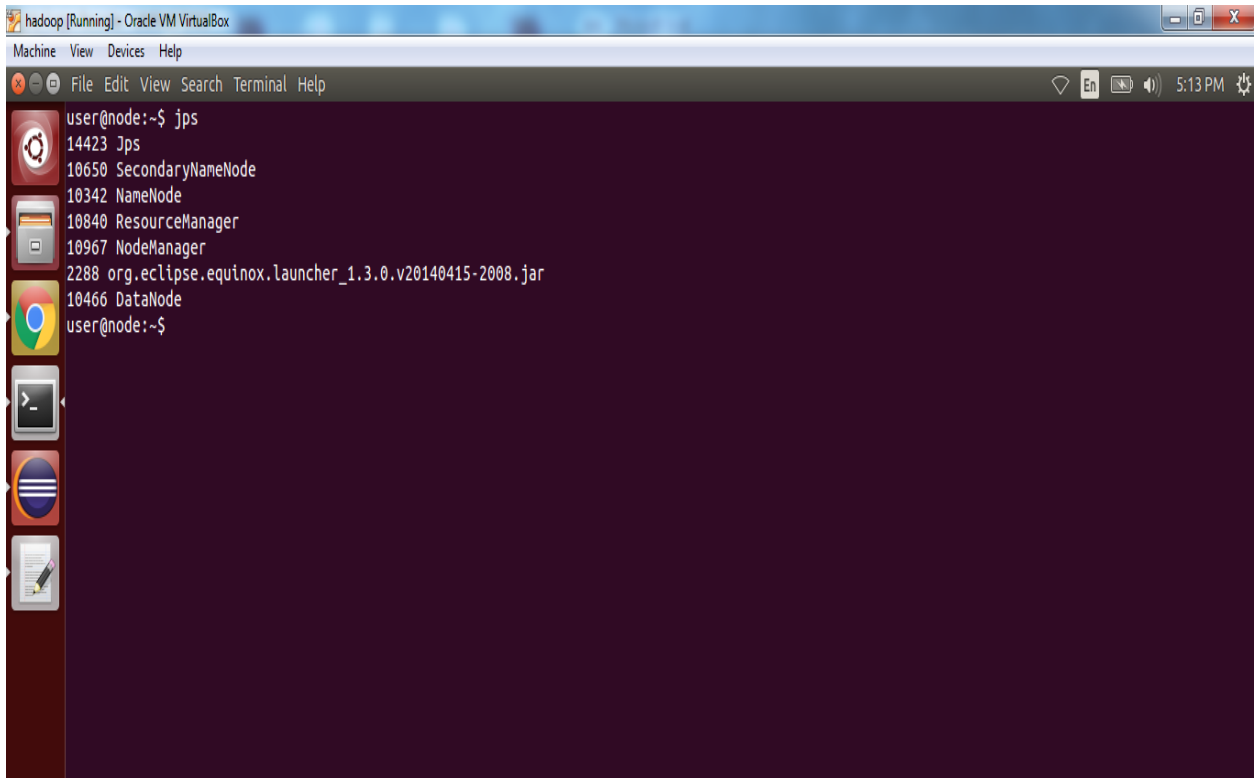
}

```

RESULTS

7. RESULTS

SCREEN -1: Initially we will start all the Hadoop nodes.

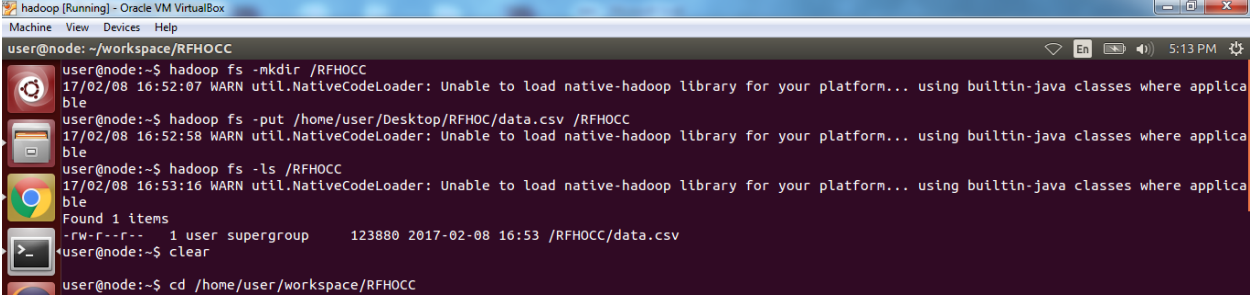


The screenshot shows a terminal window titled "hadoop [Running] - Oracle VM VirtualBox". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The status bar at the bottom right shows "En", a battery icon, a speaker icon, and the time "5:13 PM". The terminal content shows the command "user@node:~\$ jps" and its output:

```
14423 Jps
10650 SecondaryNameNode
10342 NameNode
10840 ResourceManager
10967 NodeManager
2288 org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
10466 DataNode
user@node:~$
```

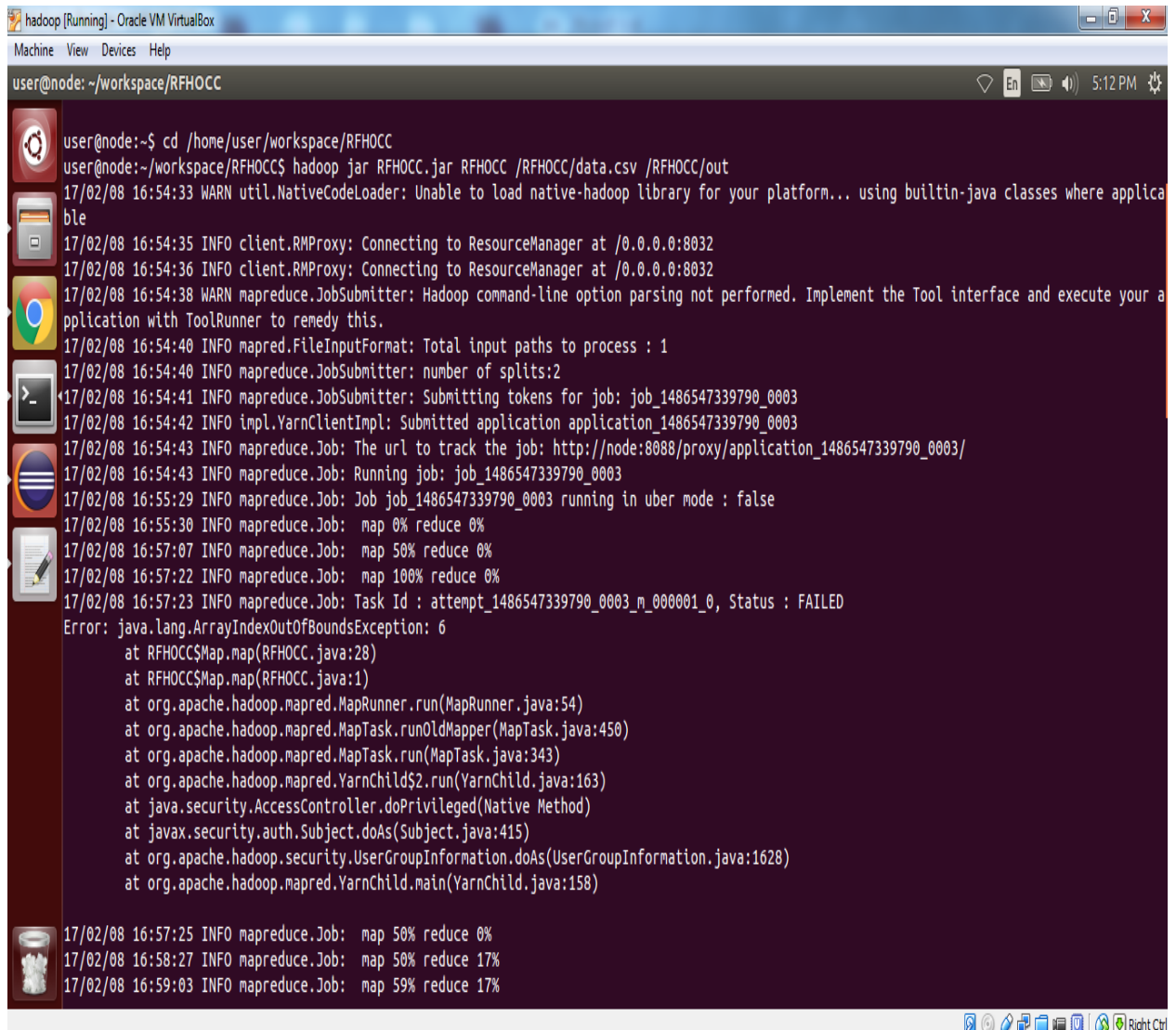
On the left side of the terminal window, there is a vertical dock with several application icons: a gear (Settings), a folder (Files), a document with a magnifying glass (Search), a terminal icon, a blue circle with white lines (Eclipse IDE), and a document with a pencil (Text Editor).

SCREEN -2:creating new directory and uploading dataset from local file system to HDFS



```
hadoop [Running] - Oracle VM VirtualBox
Machine View Devices Help
user@node: ~/workspace/RFHOCC
user@node:~$ hadoop fs -mkdir /RFHOCC
17/02/08 16:52:07 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
user@node:~$ hadoop fs -put /home/user/Desktop/RFHOC/data.csv /RFHOCC
17/02/08 16:52:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
user@node:~$ hadoop fs -ls /RFHOCC
17/02/08 16:53:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 user supergroup 123880 2017-02-08 16:53 /RFHOCC/data.csv
user@node:~$ clear
user@node:~$ cd /home/user/workspace/RFHOCC
```

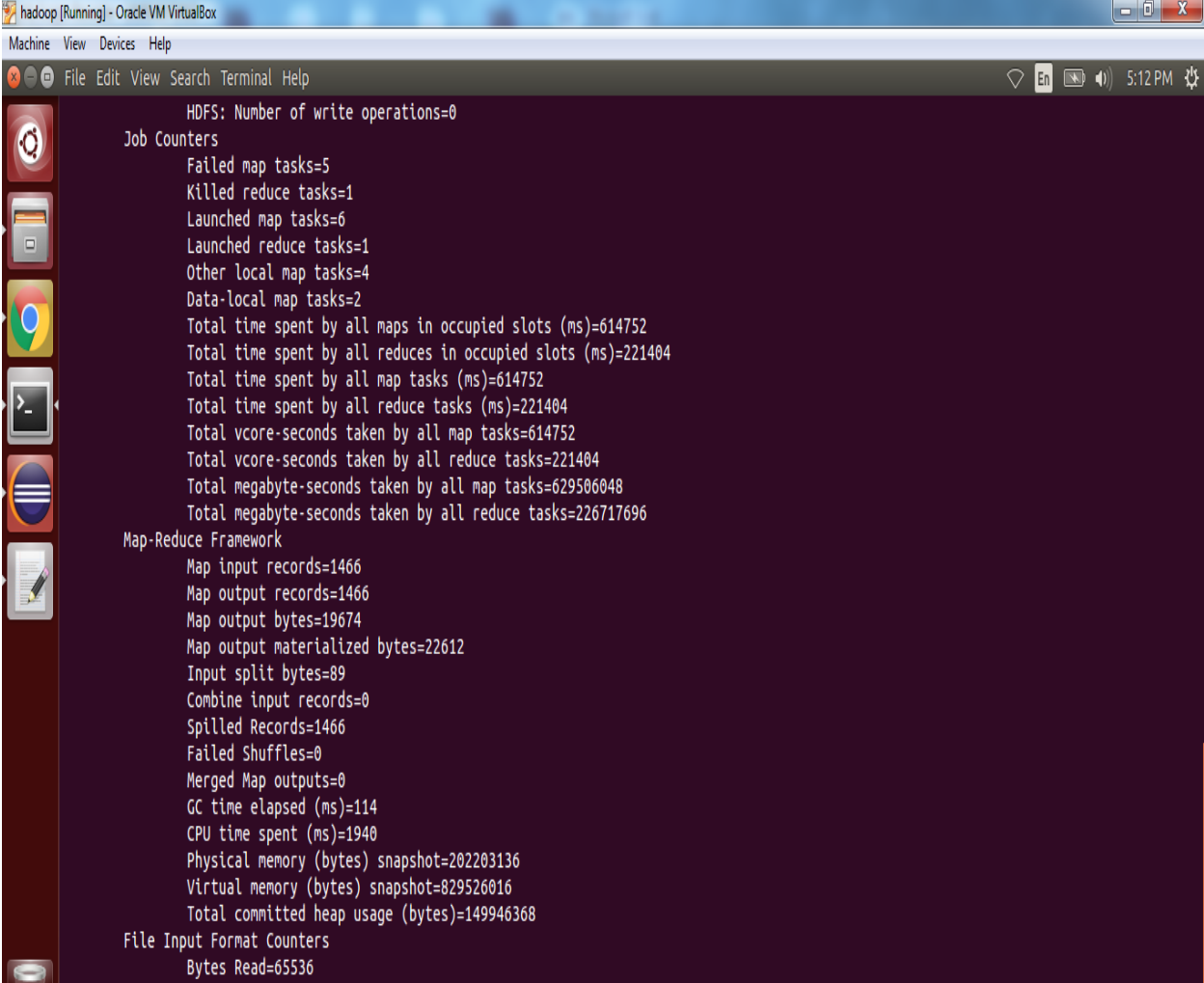
SCREEN-3: Mapreduce process starts



```
hadoop [Running] - Oracle VM VirtualBox
Machine View Devices Help
user@node: ~/workspace/RFHOCC

user@node:~$ cd /home/user/workspace/RFHOCC
user@node:~/workspace/RFHOCC$ hadoop jar RFHOCC.jar RFHOCC /RFHOCC/data.csv /RFHOCC/out
17/02/08 16:54:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/02/08 16:54:35 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/02/08 16:54:36 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/02/08 16:54:38 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/02/08 16:54:40 INFO mapred.FileInputFormat: Total input paths to process : 1
17/02/08 16:54:40 INFO mapreduce.JobSubmitter: number of splits:2
17/02/08 16:54:41 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1486547339790_0003
17/02/08 16:54:42 INFO impl.YarnClientImpl: Submitted application application_1486547339790_0003
17/02/08 16:54:43 INFO mapreduce.Job: The url to track the job: http://node:8088/proxy/application_1486547339790_0003/
17/02/08 16:54:43 INFO mapreduce.Job: Running job: job_1486547339790_0003
17/02/08 16:55:29 INFO mapreduce.Job: Job job_1486547339790_0003 running in uber mode : false
17/02/08 16:55:30 INFO mapreduce.Job: map 0% reduce 0%
17/02/08 16:57:07 INFO mapreduce.Job: map 50% reduce 0%
17/02/08 16:57:22 INFO mapreduce.Job: map 100% reduce 0%
17/02/08 16:57:23 INFO mapreduce.Job: Task Id : attempt_1486547339790_0003_m_000001_0, Status : FAILED
Error: java.lang.ArrayIndexOutOfBoundsException: 6
    at RFHOCC$Map.map(RFHOCC.java:28)
    at RFHOCC$Map.map(RFHOCC.java:1)
    at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:54)
    at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:450)
    at org.apache.hadoop.mapred.MapTask.run(MapTask.java:343)
    at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:163)
    at java.security.AccessController.doPrivileged(Native Method)
    at javax.security.auth.Subject.doAs(Subject.java:415)
    at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1628)
    at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
17/02/08 16:57:25 INFO mapreduce.Job: map 50% reduce 0%
17/02/08 16:58:27 INFO mapreduce.Job: map 50% reduce 17%
17/02/08 16:59:03 INFO mapreduce.Job: map 59% reduce 17%
```

SCREEN-4: Map reduce process completed

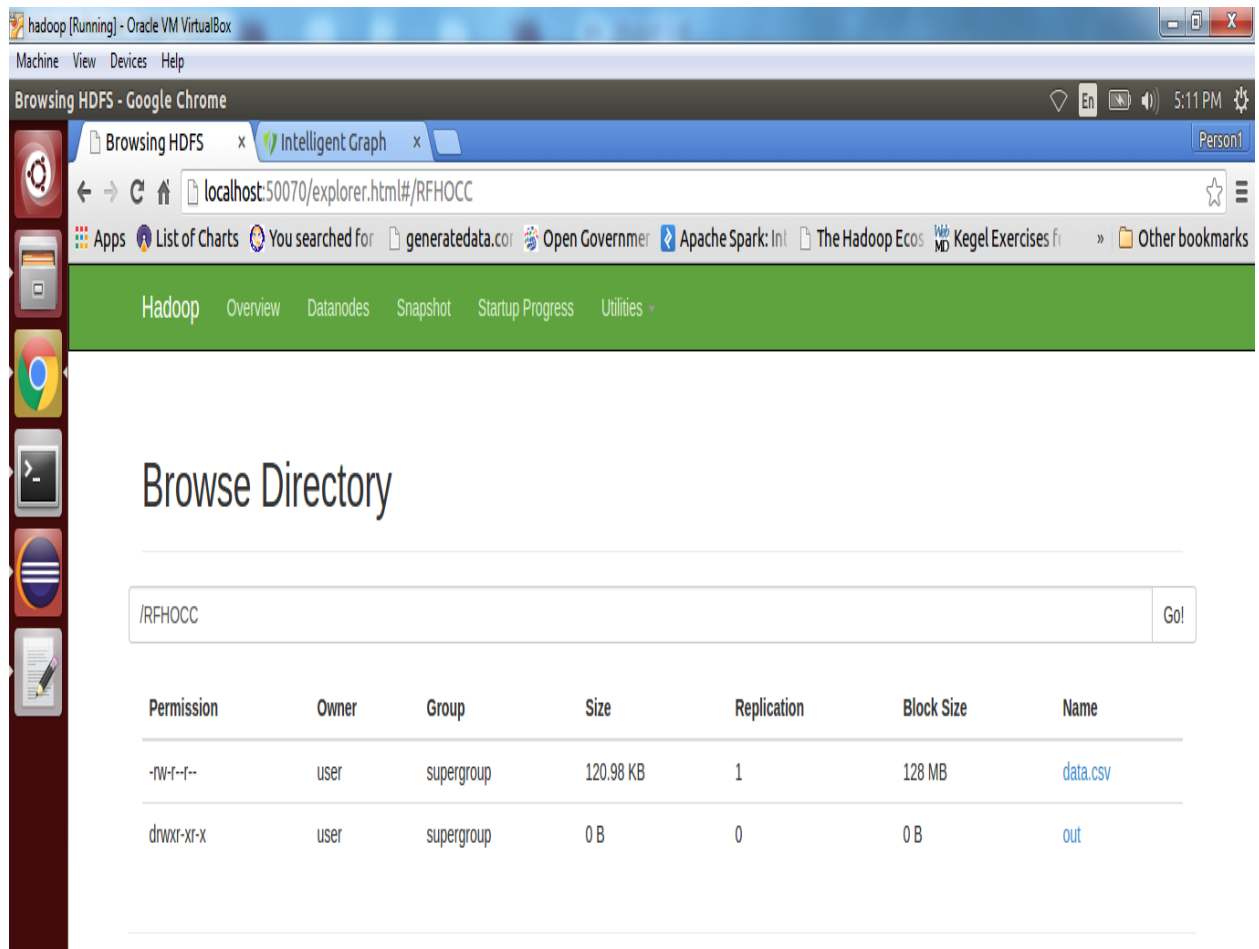


The screenshot shows a terminal window titled "hadoop [Running] - Oracle VM VirtualBox". The terminal displays the output of a Hadoop MapReduce job. The output is organized into sections: HDFS statistics, Job Counters, Map-Reduce Framework, and File Input Format Counters. The job has completed successfully, with all tasks finished and no errors reported.

```
hadoop [Running] - Oracle VM VirtualBox
Machine View Devices Help
File Edit View Search Terminal Help
5:12 PM

HDFS: Number of write operations=0
Job Counters
  Failed map tasks=5
  Killed reduce tasks=1
  Launched map tasks=6
  Launched reduce tasks=1
  Other local map tasks=4
  Data-local map tasks=2
  Total time spent by all maps in occupied slots (ms)=614752
  Total time spent by all reduces in occupied slots (ms)=221404
  Total time spent by all map tasks (ms)=614752
  Total time spent by all reduce tasks (ms)=221404
  Total vcore-seconds taken by all map tasks=614752
  Total vcore-seconds taken by all reduce tasks=221404
  Total megabyte-seconds taken by all map tasks=629506048
  Total megabyte-seconds taken by all reduce tasks=226717696
Map-Reduce Framework
  Map input records=1466
  Map output records=1466
  Map output bytes=19674
  Map output materialized bytes=22612
  Input split bytes=89
  Combine input records=0
  Spilled Records=1466
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=114
  CPU time spent (ms)=1940
  Physical memory (bytes) snapshot=202203136
  Virtual memory (bytes) snapshot=829526016
  Total committed heap usage (bytes)=149946368
File Input Format Counters
  Bytes Read=65536
```

SCREEN -5: Browsing the Directory



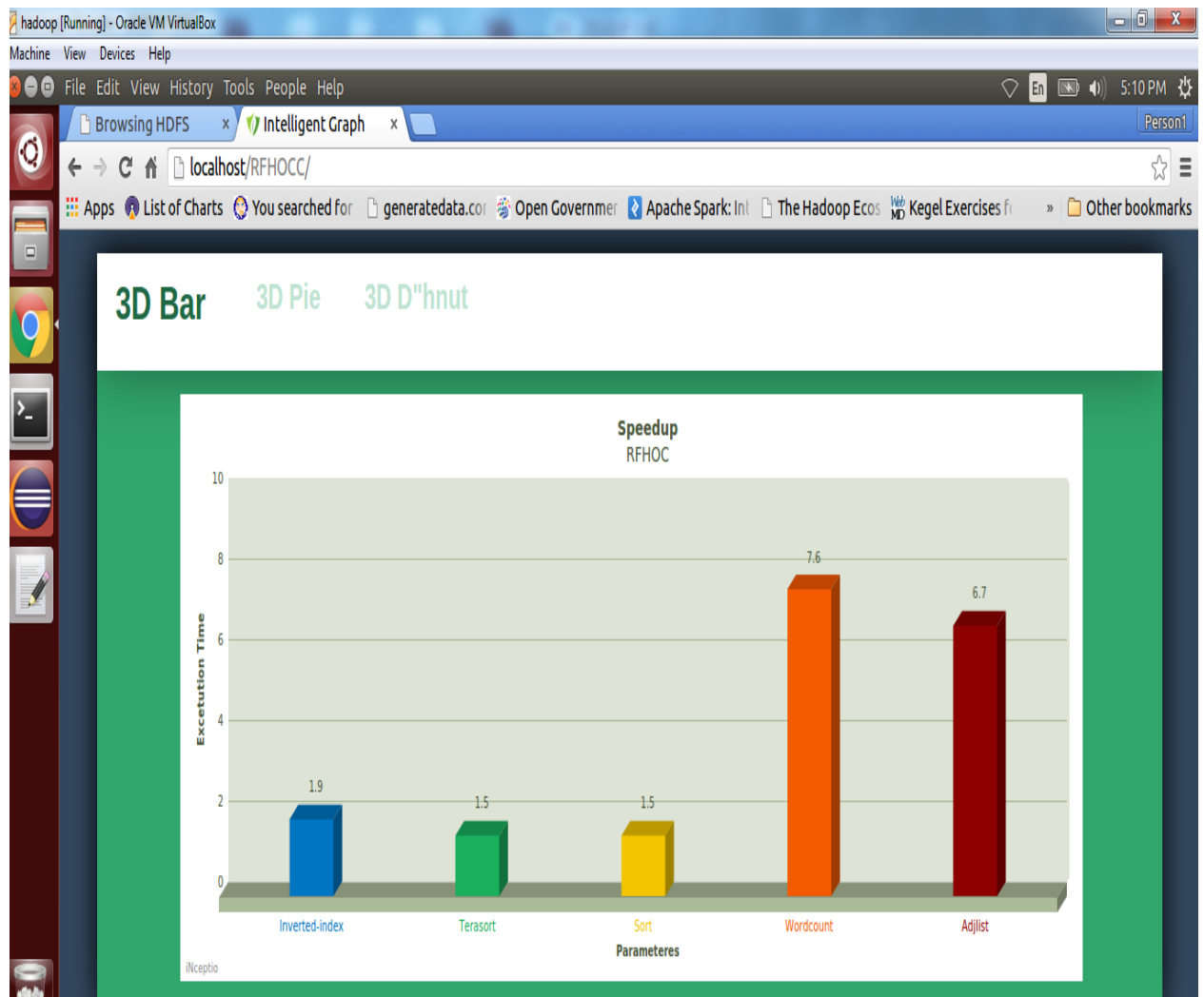
The screenshot shows a web browser window titled "Browsing HDFS - Google Chrome" with the address bar displaying "localhost:50070/explorer.html#/RFHOCC". The browser's bookmark bar includes links like "Apps", "List of Charts", "You searched for", "generatedata.co", "Open Governmer", "Apache Spark: Int", "The Hadoop Ecos", "Kegel Exercises fi", and "Other bookmarks". The page has a green header with navigation links: "Hadoop", "Overview", "Datanodes", "Snapshot", "Startup Progress", and "Utilities".

Browse Directory

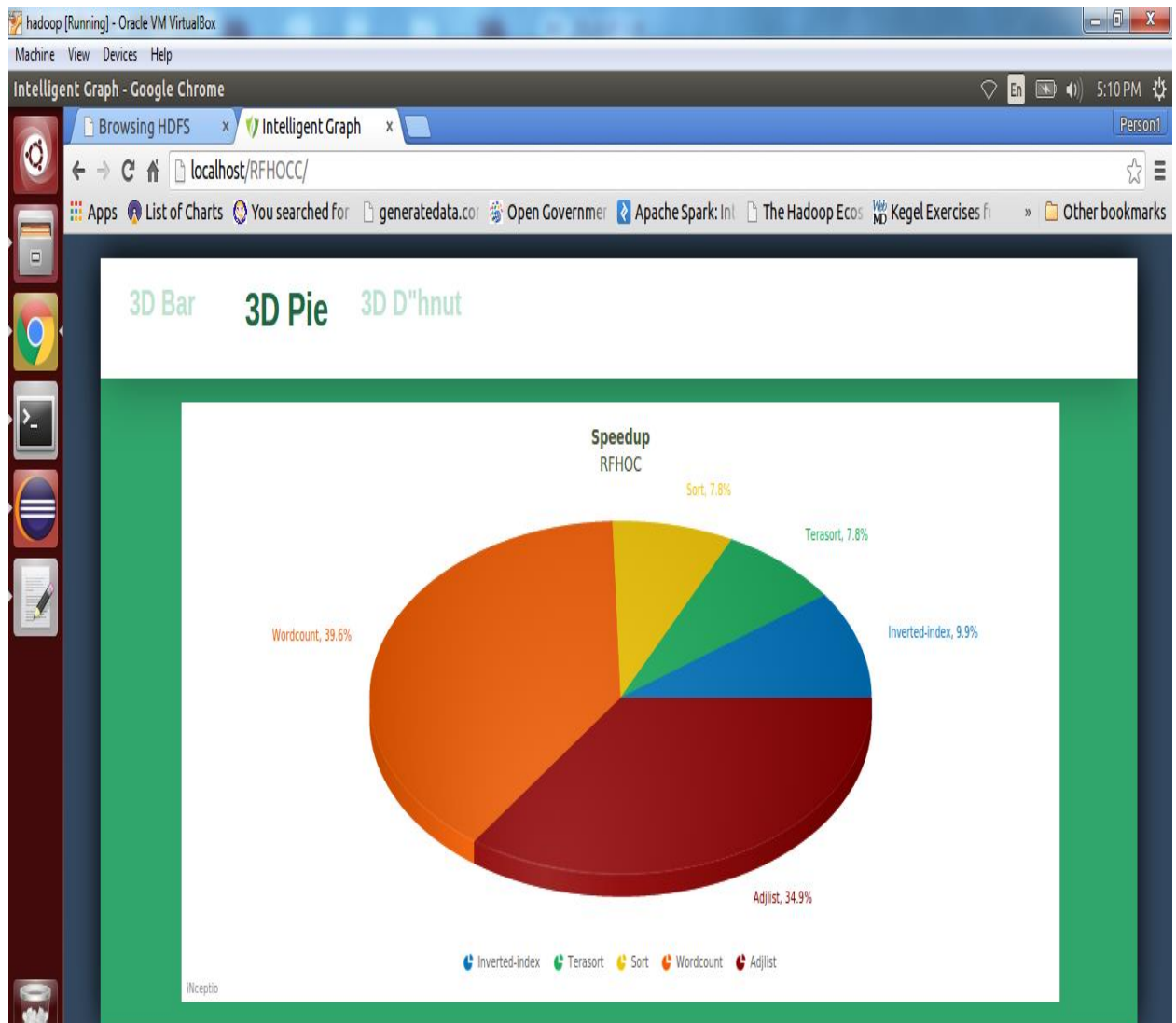
/RFHOCC Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	user	supergroup	120.98 KB	1	128 MB	data.csv
drwxr-xr-x	user	supergroup	0 B	0	0 B	out

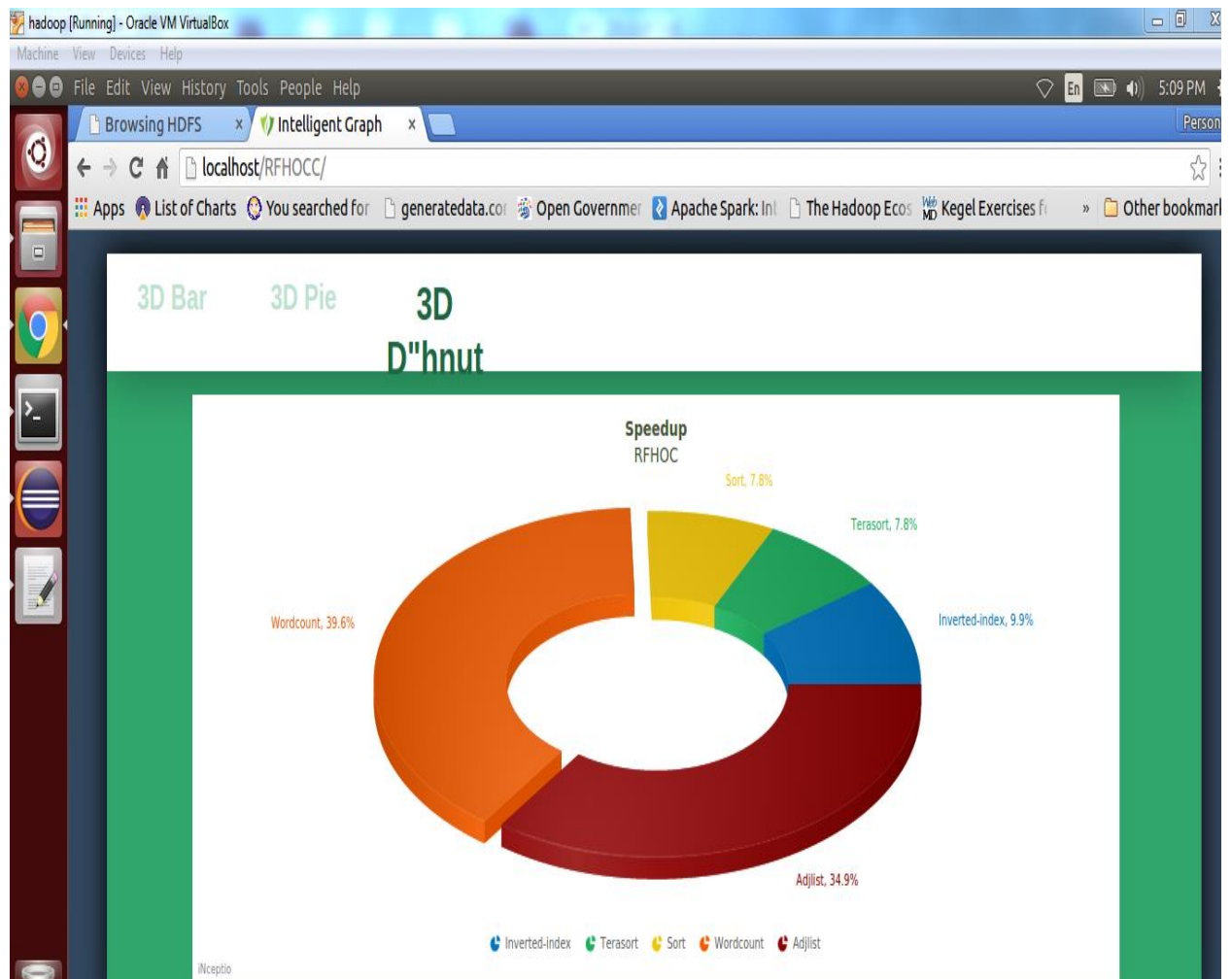
SCREEN -6: Generating the results in 3D Bar



SCREEN -7: Generating results in 3D Pie



SCREEN -8: Generating results in 3D D"hnut



SYSTEM TESTING

8. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

8.1 TYPES OF TESTING

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encounter.

TEST CASES:

Test Case Id	Test Case Name	Test Case Desc	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		
Define Reducers 01	Reducer location details	It defines the reducers particular website details by providing relevance coefficient and similarities.	If I doesn't provide relevance coefficient similarities of each page.	website details will not be saved	Reducers details will not be saved successfully	Fail	High
Define Reducers 02	Reducer location details	It defines the reducers particular website details by providing relevance coefficient and similarities.	provide relevance coefficient similarities of each page.	website details will be saved	Reducers details will be saved successfully	Pass	High
Reducer 1 & 2 03	Run reducers	Start the reducer nodes ,and all	If I not run the application	Reducer don't know the updated details	Reducer node will be started	Pass	High

		details will be updated at reducer node					
Upload 04	Upload the Wrong input data	Data will not be uploaded from shuffle phase	I can't upload the data	I can't reduce the relevance coefficient similarities	Input data will not be loaded successfully	Fail	High
Upload 05	Upload the input data	Data will be uploaded from shuffle phase	If I can't upload the data	I can't reduce the relevance coefficient similarities	Input data loaded successfully	Pass	High
Start Mapreduce For Auto-Tuning 06	Auto-Tuning using Mapreduce	It Auto tunes all the partitioned data	If I not start the Auto-tuning	I can't reduce the relevance coefficient similarities	After processing the Auto-Tuning data, it displays the count result.	Pass	High
Graph 07	Random Forest Auto Tuning graph	Displays the graph better processing time & Technique	If I can't do any Auto-Tuning	Nothing will be displayed	Graph will be displayed using Auto-tuning Processed data	Pass	High

CONCLUSION

9. CONCLUSION

Performance tuning is a challenging problem for Hadoop/Map Reduce workloads because of the large number of Hadoop configuration parameters. Yet, there is a significant opportunity for optimizing performance beyond Hadoop's default runtime configuration parameters. Previously proposed techniques to automatically tune the Hadoop configuration parameters build analytical models based on oversimplified assumptions, affecting the overall model's accuracy and ultimately the achievable performance improvements.

In this project, we propose RFHOC, a novel methodology to optimize Hadoop performance by leveraging the notion of a random forest to build accurate and robust performance prediction models for the phases of the map and reduce stage of a Hadoop program of interest. Taking the output of these models as the input to a genetic algorithm to automatically search the Hadoop configuration space yields a Hadoop configuration setting that leads to optimized application performance. We evaluate RFHOC using five Hadoop benchmarks, each with five input data sets ranging from 50 GB to 1 TB.

The results show that RFHOC speeds up Hadoop programs significantly over the previously proposed cost-based optimization approach. The speedups achieved are significant: by 2.11_ on average and up to 7.4_. Furthermore, we find RFHOC's performance benefits to increase with increasing input data set sizes.

FUTURE ENHANCEMENT

10. FUTURE ENHANCEMENT

Random forest approach for auto tuning Hadoop configuration scheme is to give the best configuration .In future we can represent a enhanced parallel detrended fluctuation analysis. Algorithm (EPDFA) for scalable analytics on massive datasets EPDFA is a enhanced Hadoop platform whose parameters are applied by gene expression programming.

REFERENCES

11. REFERENCES

- [1] L. Lie, “Heuristic artificial intelligent algorithm for genetic algorithm,” *Key Eng. Materials*, vol. 439, pp. 516–521, 2010.
- [2] K. S. Beyer, V. Ercegovic, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan, and E. J. Shekita, “Jaql: A scripting language for large scale semistructured data analysis,” in *Proc. VLDB Conf.*, Sep. 2011, pp. 1272–1283.
- [3] M.-P. Wen, H.-Y. Lin, A.-P. Chen, and C. Yang, “An integrated home financial investment learning environment applying cloud computing in social network analysis,” in *Proc. Int. Conf. Adv. Social Netw. Anal. Mining*, Jul. 2011, pp. 751–754.
- [4] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, “Starfish: A self-tuning system for big data analytics,” in *Proc. Biennial Int. Conf. Innovative Data Syst. Res.*, Jan. 2011, pp. 261–272.
- [5] H. Herodotou and S. Babu, “Profiling, What-if analysis, and cost based optimization of MapReduce programs,” *Proc. VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

BIBLIOGRAPHY:

- 1. <https://en.m.wikipedia.org>
- 2. www.tutorial.com
- 3. <https://hadoop.apache.org>
- 4. www.geeksforgeeks.org
- 5. www.guru99.com