**Teleparadigm**

**Web service management based on Hadoop**

**<Version 1.0.0>**

**Contents**

**Table of Contents**

1. Introduction ………………………………………………………………
   1. Purpose ……………………………………………………………
   2. Scope
   3. Existed system
   4. Proposed system
   5. Application, Objective and Benefits …………………………………
2. Software Requirement Specification (SRS)…………………………………
   1. Overview ……………………………………………………………
   2. Product Perspective ………………………………………………...
   3. System Interfaces …………………………………………………..
      1. User interface ……………………………………………
      2. Software Interfaces ……………………………………………
      3. Memory Constraints …………………………………………..
   4. Product Functions ……………………………………………………
   5. User Characteristics……………………………………………………
   6. Constraints ……………………………………………………………
   7. Logical Database Requirements ………………………………………
   8. Design Constraints ……………………………………………………
   9. Standard Compliance …………………………………………………
   10. Reliability …………………………………………………………...
   11. Availability ………………………………………………………….
   12. Security …………………………………………………………….
   13. Maintainability ……………………………………………………...
   14. Portability …………………………………………………………..
   15. System Mode ……………………………………………………….
   16. Supporting Information ……………………………………………
   17. Document Control …………………………………………………
   18. Specific Requirements………………………………………….
   19. Performance Requirements ………………………………………..
   20. Software System Attribute ………………………………………
   21. Objects……………………………………………………………..
   22. Feature…………………………………………………………..
   23. Stimulus ……………………………………………………………
   24. Response……………………………………………………………
   25. Functional Hierarchy ……………………………………………….
   26. Additional Comments ………………………………………………
3. Modules …………………………………………………………………….
   1. Module 1
   2. Module 2
   3. Module 3
   4. Module 4
4. Technology Features ……………………………………………………….
   1. JAVA……………………………………………….
   2. Hadoop……………………………………………………..
   3. MapReduce……………………………………………………………..
   4. HDFS……………………………………………………………..
   5. HBASE………………………………………………………………
5. Design……………………………………………………………………..
   1. Structural Diagrams
   2. Deployment Diagrams
   3. Behavioral Diagrams
   4. Sequence Diagram
   5. Collaboration Diagram
   6. State Chart Diagram
   7. Activity Diagram
6. Architecture Diagram …………………………………………………….
7. Programming Code …………………………………………………………
8. Testing ………………………………………………………………………
   1. Black Box Testing
   2. White Box Testing
   3. Grey Box Testing
   4. Regression Testing
   5. Test Cases
9. Output screens …………………………………………………………
10. Conclusions ………………………………………………………………
11. Future Enhancements ………………………………………………………

References …………………………………………………………………..

**CHAPTER 1**

**INTRODUCTION**

To solve the disadvantage that huge amount of web service impose on, web service management framework based on Hadoop environment is proposed. In this framework, Hadoop Distributed File System (HDFS) function as a foundation for supporting the HBase and MapReduce. The HBase is assigned to manage the functional and non-functional properties extracted from web service advertisement. To accurately and quickly find the satisfied web service, the HBase table based on request interface is established. In addition, the non-functional property index mechanism based on quality of service (QoS) tree is also designed for further enhancing the performance of non-functional property retrieval. To satisfy the user's complex requirement, the searching optimal solution in all possible paths which depends on MapReduce is proposed.

* 1. **PURPOSE**

This web service management framework based on hadoop is developed to provide High performance   and Security compare with traditional web service management.

High Performance achieved in two ways

– Using HTTP redirection to provide data locality

– File read/write are redirected to the corresponding data nodes

**1.1.1Scope**

In this framework, Hadoop considers the aggregated QoS is necessary in composition Distributed File System (HDFS) function as a foundation for process.

The HBase is assigned to manage the functional and non-functional properties

* 1. **EXISTING SYSTEM**

In existing system as more and more customers delegate their task to web service, one can expect a huge web service repository to be fast and efficiently searched. The web service advertisement information is the loose structure, and it's not convenient to be managed by the relational database. The other problem leads to the decreasing performance of web service selection for arranging several services into a complex one with QoS guaranteed.

* 1. **PROPOSED SYSTEM**

To increase the efficiency of web service management, an efficient web service management framework based on Hadoop is introduced. This framework can overcome the drawback which occurs in the traditional web service management infrastructure.

**1.4 APPLICATION, GOAL AND OBJECTIVE**

**Application**

The application will be utilized by the Users.

**Goal**

The Web Site is intended to avoid server failure, virus attack and decrease the infrastructure costs, maintenance costs occurred to maintain the application.

**Objective**

Design is the first step in moving from problem domain to solution domain. Design is essentially the bridge between requirements specification and the final solution. The goal of design process is to produce a model or representation of a system can be used later to build that system. The produced model is called the ‘Design of the System’.

It is a plan for a solution of the system.

The objective of design phase is to:

* Create User Interface.
* Create Application.

**CHAPTER-2**

**SOFTWARE REQUIREMENT SPECIFICATION**

**Approvals Signature Block**

|  |  |  |
| --- | --- | --- |
| **Organization Responsibility** | **Signature** | **Date** |
| Customer /customer representative |  |  |
| Project Manager |  |  |
| Software Quality Assurance Leader |  |  |
| Software Configuration Management Leader |  |  |
| User Documentation Leader |  |  |
| User Training Leader |  |  |

**2.1 Overall Description**

The web service advertisement information is the loose structure, and it's not convenient to be managed by the relational database. Moreover, the traditional central model, such as UDDI platform, cannot easily scale to support a large number of web service**.** To address these problems, an efficient web service management framework based on Hadoop is introduced. This framework can overcome the drawback which occurs in the traditional web service management infrastructure. In addition, the two components of Hadoop, HBase and MapReduce, is integrated into this framework. To strengthen the retrieving performance of the functional and non-functional properties, the tables in HBase is designed in terms of the interface-based and QoS tree index. After obtaining the qualified web service, the parallel algorithm based on MapReduce supports us to search all possible solutions and select the optimal solution for satisfying the customer's requirement.

**2.2 Product Perspective**

The project is a part of a large system*.*

**Block diagram**

The major components of the larger system are shown as below.

HDFS

Eclipse IDE

Cygwin

Front-end Middleware

Back-end

Middleware

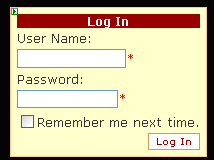
**2.3 System Interfaces**

Description of various interfaces used in the system is described in the subsequent paragraph

**2.3.1 User Interfaces**

Interface between the software product and its users

User friendly interfaces as depicted below will be used.

****

1.Screen formats

These are required to be created with the following features:

1. User friendly.
2. Indicate the Mandatory fields by asterisk (\*).
3. Fill up default values where ever possible
4. Give combo boxes in all input screens.
5. User and data entry personnel details should be stored in the database through “Save” button
6. Authentication of users should be carried out where ever required.

2. Web page

Each screen should have

1. Menu driven facility
2. Uniformity
3. Consistency.

3. Outputs

Analytical outputs should be supported by graphs.

1. For each output there should be provision of printing
2. Emails should be well structured
3. Each paper output formatted to A4 size paper

4. DOs (Input)

1. Each input box to be supported by label
2. Give tool tips where required
3. Give formats (mm/ dd /yy)
4. Provide Tab Index
5. Input elements without visible labels **should** continue to contain text (search, login)

Top of Form

1. Password 
2. radio inputs **should** have one option checked as default

**5. DONTs (Inputs)**

1. Don’t distract users from their goals
2. Don’t use dark background with dark font colors
3. Don’t use too many colors

**6. Error Messages**

Give small error messages like “Incorrect Data” or “Incorrect Date Format” or “Field is required” and supplement it with “Ok” Button.

**2.3.2 Hardware Interfaces**

Following Hardware Interfaces are required:

* Pentium 4 Processor
* Minimum 4GB RAM
* 64 bit Windows 7 Ultimate
* 80 GB HDD

**2.3.3 Software Interfaces**

Following Software is required:-

* Java 6
* Eclipse Indigo(With Hadoop Configuration)
* Cygwin
* Hadoop Appliance
* HBase

**2.3.4 Memory Constraints**

RAM with minimum 4 GB and 250 GB HD space

**2.4 Product Functions: HDFS Functions**

**1**. **HDFS should have the facility to**

1. Create ,
2. Update
3. Edit
4. Delete
5. Search
   1. **Authentication**

Users at all levels should be authenticated before giving them access.

**3. Analysis**

Outputs of all analysis should be in the form of

a) Data in tabular form.

b) Graphical representation of performance.

**2.5 User characteristics**

Following are the characteristics of the intended users:

Educational Level: Level of the users is between Educated to highly educated.

Experience: Experienced in their domain but training in the proposed application.

Technical Expertise: Training required in the proposed application.

**2.6 Constraints**

**Site Adaption Requirements:**

Only system Administrator and DBA are authorized to carry out this task jointly.

Assumptions and Dependencies:

It is assumed that all the systems will have the basic HW, SW and Communication Interfaces available. The users are trained in using the application.

**Apportioning of Requirements:**

Identify requirements that may be delayed until future versions of the system. All requirements will be met.

**2.7 Logical Database Requirements**

Logical requirements connected with the database include:

1. Most of the values are string types but the count is in numbers.
2. Counting of the patients connected with a specific disease is monitored immediately after entering the record of each patient.
3. Accessing rights are limited to authenticated users only.
4. Integrity constraints are maintained by setting the relationships.

**Functions:**

Validity checks on the inputs

**Data Entry Operators:**

a) Responses to abnormal situation, including

* Overflow – Periodic Backups
* Communication facilities : Internet
* Error handling and recovery : Periodic Backup, Error alerts, Maintain Error Logs

b) Effect of parameters

**2.8 Design Constraints**

**Hardware Constraints:**

* Pentium 4 processor,
* RAM Minimum 4 GB memory,
* 250 GB Hard disk.

**Design constraints:**

* Interaction between User Interface and the database should be through MapReduce.

**2.9 Standards Compliance**

Following standards will be maintained:-

* Report format.
* Data naming (As per the Naming Conventions – organization policy)
* Transaction procedures.

**2.10 Reliability**

Software will be handed over to the client after carryout extensive

* Unit Testing
* Integration Testing
* System Testing
* After performing periodic demonstrations to the end users on completion of each module and keep a log of the errors / observations made by the user
* Number of errors during Unit Testing may be more but they should show decreasing trend during Integration and System Testing and should reduce to zero at the time of delivery.

Ensure strict compliance to Project Plan

**2.11 Availability**

Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

**2.12 Security**

* Authentication module will ensure that only authorized users are provided access to do transactions and view their transactions.
* Roles will be defined to impose restrictions on the authorized users.
* Ensure that buffer overflow and integer overflow will be avoided.
* Whenever user is deleted his privileges will also get deleted.
* Carryout periodic backup of the database and maintain a log.
* Honey Pots Intentionally include some PCs in the network which are vulnerable forhackers. They can be used to catch hackers or fix vulnerability.

**2.13 Maintainability**

(Rating them High, Medium, or Low), or ranking them in order of importance (1, 2, 3, etc Keep a count of the number of lines of code. Though there cannot be a benchmark for the maximum lines of code in a sub routine but higher the lines of code indicates

* Higher is the maintenance.
* Need to split up in to child levels.
* Place every module in Try Catch () and finally () block to prevent disgraceful exit.

Avoid excessive complexity.

Avoid excessive Inheritance.

Variable name should not match the field names.

Reduce complexity of conditional branching.

**2.14 Portability**

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include

1. Percentage of components with host-dependent code
2. Percentage of code that is host dependent
3. Use of a proven portable language
4. Use of a particular compiler or language subset
5. Use of a particular operating system.

Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics.).

|  |  |  |
| --- | --- | --- |
| **ID** | **Characteristic** | **Rank** |
| 1 | Correctness |  |
| 2 | Efficiency |  |
| 3 | Flexibility |  |

**Organizing the Specific Requirements:**

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in section 3. Some of these organizations are described in the following subsections.

**2.15 System Mode**

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency. When organizing by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode*.*

Web Site being developed in HTML5 it is compatible to most of the OS and the Web Browsers.

**2.16 Supporting Information**

The supporting information makes the SRS easier to use. It includes*:*

a) Table of Contents at the front of the document

b) Index

c) Appendices: Definitions of important terminologies are given in the Appendix

The Appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:

a) Sample I/O formats, descriptions of cost analysis studies, results of user surveys;

b) Supporting or background information that can help the readers of the SRS;

c) A description of the problems to be solved by the software;

d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When Appendices are included, the SRS should explicitly state whether or not the Appendices are to be considered part of the requirements.

Tables on the following pages provide alternate ways to structure section 3 on the specific requirements.

**2.17 Document Control**

**Document Storage**

This document was created using standard SRS Template followed by IEEE. The file is stored by the Project Manager, one signed copy is handed over to the authorized representative of the customer and the second copy is kept with the Administrator. It establishes the basis for agreement with the client on what the software product is expected to do, as well as what it is not expected to do.

**Document Owner**

Project Manager is responsible for developing and maintaining this document.

**2.18 Specific Requirements**

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Requirement** | **Remarks** |
|  | 4GB RAM | Without 4 GB performance poor |

**2.19 Performance Requirements**

Presently we are working on three terminals. It is expected that at any point of time three terminals will be in operation simultaneously. The amount of information will be numerical and text oriented and the volume will be limited

**2.20 Software System Attribute**

Analysis: This is an important module and this module should be able to raise an alert by estimating the probability of disease being escalated to epidemic.

Modular approach will be followed.

1. Create Roles.
2. Manage the application.

Graphs and email generation should be asynchronous.

**2.21 Objects**

Following objects are being considered in this application:-

* Authorized users.
* Performance Report.

**2.22 Feature**

The features are listed in the subsequent paragraphs in the form of Stimulus and Response.

**2.23 Stimulus**

* User ID + Password of Data Entry Operators

**2.24 Response**

* Performance Report.
* Login Authentication Alert.
* Registration Authentication Alert.
* Transaction alerts.

**2.25 Functional Hierarchy**

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by common inputs, common outputs, or common internal data access.

But here for this technology we are using, we need uml diagrams.

**2.26 Additional Comments**

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 3.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification. Any additional requirements may be put in a separate section at the end of the SRS.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines below, those sections called “Functional Requirement i” may be described in native language, in pseudo code, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing and outputs.

**CHAPTER-3**

**MODULES**

**3.1 Module-1**

Installation and configuration of cygwin, hive with hadoop and connects to Hive to proceess webservice request.

**3.2 Module-2**

User will interact with the HBase in which service is processed to get the QoS of the service .

**3.3 Module-3**

User observes the QoS management and effeciency of lti-source search process(MSSP) based on MapReduce while processing UDDI.

**3.4 Module-4**

In this module by using the HBase, user can efficiently and reliably retrieve the necessary information about web service selection from the web service repository based on web service selection.

**CHAPTER-4**

**TECHNOLOGY FEATURES**

4.1 Java

**Language:**

Java is an Object Oriented language that enables you to create real world applications. The code reusability feature of Java enables the software developers to upgrade the existing applications without rewriting the entire code of the application. Java can be used to create two types of programs: applications and applets.

An application is a program that runs on your computer, under the operating system of that computer. When used to create application, Java is more or less like one created using C computer language. Rather, it is Java’s ability to create Applet that makes it important.

Java exhibits the following a characteristic because of which java is a suitable language in fulfilling all the requirements of an organization. The characteristics are: Simple, Object-oriented, compiled and interpreted, portable, distributed, Secure Java is called Object Oriented Language .Java is an [open-source](http://en.wikipedia.org/wiki/Open-source_software) [Java EE](http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition)-based [application server](http://en.wikipedia.org/wiki/Application_server). An important distinction for this class of software is that it not only implements a server that runs on Java, but it actually implements the Java EE part of Java. Because it is [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29)-based, the Tomcat application server operates cross-platform: usable on any operating system that supports Java. **4.2 HADOOP:**

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

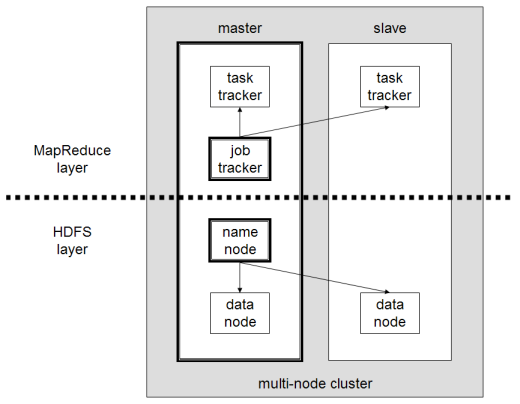
* **Hadoop Common**: The common utilities that support the other Hadoop modules.
* **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

* **Ambari:** A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
* **Avro**: A data serialization system.
* **Cassandra**: A scalable multi-master database with no single points of failure.
* **Chukwa:**A data collection system for managing large distributed systems.
* **HBase:** A scalable, distributed database that supports structured data storage for large tables.
* [**Hive**](http://hive.apache.org/): A data warehouse infrastructure that provides data summarization and ad hoc querying.
* [**Mahout**](http://mahout.apache.org/): A Scalable machine learning and data mining library.
* [**Pig**](http://pig.apache.org/): A high-level data-flow language and execution framework for parallel computation.
* [**Spark**](http://spark.incubator.apache.org/): A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
* [**ZooKeeper**](http://zookeeper.apache.org/): A high-performance coordination service for distributed applications.

**Architecture**:

Hadoop consists of the *Hadoop Common* package, which provides file system and OS level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2) and the [Hadoop Distributed File System](http://en.wikipedia.org/wiki/Hadoop#Hadoop_distributed_file_system) (HDFS). The Hadoop Common package contains the necessary [Java ARchive (JAR)](http://en.wikipedia.org/wiki/JAR_(file_format)) files and scripts needed to start Hadoop. The package also provides source code, documentation and a contribution section that includes projects from the Hadoop Community. For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the network switch) where a worker node is. Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic. HDFS uses this method when replicating data to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable.



A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or *worker node* acts as both a DataNode and TaskTracker, though it is possible to have data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications. Hadoop requires [Java Runtime Environment (JRE)](http://en.wikipedia.org/wiki/JRE) 1.6 or higher. The standard start-up and shutdown scripts require [Secure Shell](http://en.wikipedia.org/wiki/Secure_Shell)(ssh) to be set up between nodes in the cluster.

In a larger cluster, the HDFS is 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, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNode and DataNode architecture of HDFS is replaced by the file-system-specific equivalent.

**4.3 MapReduce**

**MapReduce** is a [programming model](http://en.wikipedia.org/wiki/Programming_model) for processing large data sets with a [parallel](http://en.wikipedia.org/wiki/Parallel_computing), [distributed](http://en.wikipedia.org/wiki/Distributed_computing) algorithm on a [cluster](http://en.wikipedia.org/wiki/Cluster_(computing)).

A MapReduce program is composed of a **Map()** procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a **Reduce()** procedure that 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 by [marshalling](http://en.wikipedia.org/wiki/Marshalling_(computer_science)" \o "Marshalling (computer science)) 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](http://en.wikipedia.org/wiki/Redundancy_(engineering)) and [fault tolerance](http://en.wikipedia.org/wiki/Fault-tolerant_computer_system).

The model is inspired by the [map](http://en.wikipedia.org/wiki/Map_(higher-order_function)) and [reduce](http://en.wikipedia.org/wiki/Fold_(higher-order_function)) functions commonly used in [functional programming](http://en.wikipedia.org/wiki/Functional_programming), although their purpose in the MapReduce framework is not the same as in their original forms. Furthermore, 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 once.

MapReduce [libraries](http://en.wikipedia.org/wiki/Library_(software)) have been written in many programming languages, with different levels of optimization. A popular [open-source](http://en.wikipedia.org/wiki/Open-source_software) implementation is [Apache Hadoop](http://en.wikipedia.org/wiki/Apache_Hadoop). The name MapReduce originally referred to the proprietary [Google](http://en.wikipedia.org/wiki/Google) technology but has since been [genericized](http://en.wikipedia.org/wiki/Generic_trademark).

'MapReduce' is a framework for processing [parallelizable](http://en.wikipedia.org/wiki/Parallel_computing) problems across huge datasets using a large number of computers (nodes), collectively referred to as a [cluster](http://en.wikipedia.org/wiki/Computer_cluster) (if all nodes are on the same local network and use similar hardware) or a [grid](http://en.wikipedia.org/wiki/Grid_Computing) (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Computational processing can occur on data stored either in a [filesystem](http://en.wikipedia.org/wiki/Filesystem" \o "Filesystem) (unstructured) or in a [database](http://en.wikipedia.org/wiki/Database) (structured). MapReduce can take advantage of locality of data, processing data on or near the storage assets to decrease transmission of data.

**"Map" step:** The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level [tree](http://en.wikipedia.org/wiki/Tree_(data_structure)) structure. The worker node processes the smaller problem, and passes the answer back to its master node.

**"Reduce" step:** The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

**4.4 HDFS**

The **Hadoop distributed file system** (**HDFS**) is a distributed, scalable, and portable file-system written in [Java](http://en.wikipedia.org/wiki/Java_(software_platform)) for the Hadoop framework. Each node in a Hadoop instance typically has a single namenode; a cluster of datanodes form the HDFS cluster. The situation is typical because each node does not require a datanode to be present. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses the [TCP/IP](http://en.wikipedia.org/wiki/TCP/IP) layer for communication. Clients use [Remote procedure call](http://en.wikipedia.org/wiki/Remote_procedure_call) (RPC) to communicate between each other.

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. In addition, there are a number of DataNodes, usually one per node in the cluster, which 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. 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 NameNode or the DataNode 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 NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.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. The system is designed in such a way that user data never flows through the NameNode.

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. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

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.

**4.5 HBASE**

Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's [Bigtable: A Distributed Storage System for Structured Data](http://research.google.com/archive/bigtable.html) by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

#### Features :

* Linear and modular scalability.
* Strictly consistent reads and writes.
* Automatic and configurable sharding of tables
* Automatic failover support between RegionServers.
* Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
* Easy to use Java API for client access.
* Block cache and Bloom Filters for real-time queries.
* Query predicate push down via server side Filters.
* Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options.
* Extensible jruby-based (JIRB) shell
* Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX.

**CHAPTER-5**

**DESIGN**

Design Patterns brought a paradigm shift in the way object oriented systems are designed. Instead of relying on the knowledge of problem domain alone, design patterns allow past experience to be utilized while solving new problems. Traditional object oriented design (OOD) approaches such as Booch, OMT, etc. advocated identification and specification of individual objects and classes. Design Patterns on the other hand promote identification and specification of collaborations of objects and classes. However, much of the focus of recent research has been towards identification and cataloging of new design patterns. The effort has been to assimilate knowledge gained from designing systems of the past, in various problem domains. The problem analysis phase has gained little benefit from this paradigm. Most projects still use traditional object oriented analysis (OOA) approaches to identify classes from the problem description. Responsibilities to those classes are assigned based upon the obvious description of entities given in the problem definition.

Pattern Oriented Technique (POT) is a methodology for identifying interactions among classes and mapping them to one or more design patterns. However, this methodology also uses traditional OOA for assigning class responsibilities. As a result, its interaction oriented design phase (driven by design patterns) receives its input in terms of class definitions that might not lead to best possible design.

The missing piece here is the lack of an analysis method that can help in identifying class definitions and the collaborations between them which would be amenable to application of interaction oriented design. There are two key issues here. First is to come up with good class definitions and the second is to identify good class collaborations.

It has been observed in that even arriving at good class definitions from the given problem definition is non-trivial. The key to various successful designs is the presence of abstract classes (such as an event handler) which are not modeled as entities in the physical world and hence do not appear in the problem description. In anticipating change has been proposed as the method for identifying such abstract classes in a problem domain. Another difficult task is related to assignment of responsibilities to entities identified from the problem description. Different responsibility assignments could lead to completely different designs. Current approaches such as Coad and Yourdon, POT etc. follow the simple approach of using entity descriptions in the problem statement to define classes and fix responsibilities. We propose to follow a flexible approach towards assigning responsibilities to classes so that the best responsibility assignment can be chosen.

The second issue is to identify class collaborations. Techniques such as POT analyze interactions among different sets of classes as specified in the problem description. Such interacting classes are then grouped together to identify design patterns that may be applicable. However, as mentioned earlier, only the interactions among obvious classes are determined currently. Other interactions involving abstract classes not present in the problem or interactions that become feasible due to different responsibility assignments are not considered. We present some techniques that enable the designer to capture such interactions as well.

**INTERACTION BASED ANALYSIS AND DESIGN**

**Top-down approach:**

This approach is applicable to situations where the designer knows the solution to the given problem. It is true for problem domains that have well established high-level solutions and different implementations vary in low level details (for e.g. Enterprise Resource Planning (ERP) systems). Her main concern is to realize that solution in a way such that the implemented system has nice properties such as maintainability and reusability etc.

To achieve this goal, the system designer selects appropriate design patterns that form the building blocks of her solution. Having obtained this design template (design type), she maps the classes and objects participating in those patterns to the entities of the problem domain. This mapping implicitly defines the responsibilities of various classes/objects that represent those entities. To help clarify the concept, consider a scenario where an architect is assigned the task of building a flyover. Flyover construction is an established science and the architect knows the solution to the problem. She starts by identifying component patterns such as road strip, support pillars, side railings and so on. Having done that, she maps the participating objects to actual entities in the problem domain. This would involve defining the length and width of the road strip based upon the space constraints specified in the problem. The height and weight of the pillars get decided based upon the load requirements specified. The entry and exit points get decided based upon the geography of the location and so on. This results in a concrete design instance. Some new classes or objects, not existing in the domain model, may also have to be introduced for a successful instantiation of the design template. For instance, the problem domain may not model an abstract entity such as an event handler which may be a participant in some portion of the design template. Such generic classes/objects may be drawn from a common repository of utility classes. Interaction driven analysis phase here is simple since the interactions (in the form of design patterns) are already well established and directly obtained from the knowledge base.

**Bottom -up approach:**

This approach is applicable in scenarios where interactions in the problem domain are not well understood and need to be discovered and explored. This situation is a fundamental problem faced by the designers of object oriented systems. It relates to the fact that objects oriented analysis (OOA) does not help much in creating a solution to the problem at hand. The analysis phase is mainly concerned with enhancing the understanding of the problem domain. This knowledge is then later used by a problem solving approach to come up with a solution possessing good design properties. As a result, at the end of the analysis phase the designer has a set of well defined components that need to be assembled together for realizing a solution. For instance, to build a route finder application the OOA phase helps in modeling the domain objects such as roads, vehicles, cities, addresses etc. but does not actually provide a solution for finding routes between two given addresses. This is similar to having various pieces of a jigsaw puzzle but the puzzle still needs to be solved. The problem in software systems is further complicated by the fact that there is generally no unique solution to a problem. There are always trade-offs at various stages and the resulting designs are a reflection of the choices made at those stages. In the jigsaw puzzle example this is similar to the situation where different sets of the same puzzle are available each differing from another in terms of the design of its component pieces. Some component designs may help in solving the puzzle faster and more efficiently than others.

The bottom-up approach helps in such situations where the entities in the problem domain have been identified by traditional OOA techniques but multiple choices exist in terms of assigning responsibilities to those entities. Unlike top-down approach, the mapping of responsibilities to entities is not dictated by the design solution specified by the designer. Instead, the task of the designer here is to try various responsibility assignments and create an interaction specification involving those objects. The objective of this interaction driven analysis is to obtain an interaction specification that helps in arriving at a solution with best design characteristics possible. Having identified the entities in the domain, the starting point for the designer is to identify various alternatives available for assigning responsibilities to individual objects. Her domain knowledge helps her in this task. Given these alternatives for potential object definitions and standard utility objects (such as schedulers, event handlers etc.), the next step is to find compositions of these building blocks (i.e. interactions of these objects) that provide alternative solutions to the problem. This task is a non-trivial one especially when done manually. There are just too many combinations to be considered, for any human designer to obtain alternative solutions in a reasonable amount of time. We need to apply semi-automated software composition techniques based on some formal specification. Several such approaches have been recently investigated in the context of e-services. These include workflow based approaches and AI Planning based techniques. Other formal techniques for specifying composition include Petri-net based models, automata-based models; temporal logics etc. from verification community and X Query, XML constraint tools based techniques from data management community.

The resulting candidate compositions (i.e. interaction specifications) then need to be compared with existing design patterns either manually or automatically. It is not beyond imagination to visualize that with advancement in automated composition techniques, new design patterns may get identified during this process. For instance, techniques such as Reinforcement Learning have resulted in new novel solutions in various domains such as playing Backgammon. In such a case, the resulting designs may need to be evaluated manually. The best design among the alternatives is then chosen for implementing the system.

**OPEN ISSUES**

**Identifying interactions:**

This is a crucial step in the analysis phase and the success of remaining phases depends on it. The issue here is to identify interactions which are not evident from the problem description but may hold the key to an efficient design solution. The bottom-up approach proposed in this paper takes a step in this direction but a lot more work is needed. The analysis method should be such that it is able to incorporate abstract classes such as event handlers, proxies etc. Moreover, current analysis methods map entities to responsibilities of individual classes in terms of services they provide and methods they invoke on other classes. However, an entity may be realized by a set of classes. For instance, an adapter class hides the interface of an adoptee class and they collectively provide the desired functionality. Similarly, an abstraction and its implementation provide a single functionality through separate classes resulting in increased maintainability. The analysis method needs to be able to determine when is it appropriate to realize an entity responsibility by means of multiple interacting classes.

**Representation of Class Responsibilities:**

Since we need to specify different alternative class responsibilities, as in bottom-up approach, a mechanism is required to document them in a machine interpretable format. Some of these responsibilities would get captured in the form of methods a class exports or methods it invokes on other classes. However, other responsibilities with respect to its interaction with other classes need to be explicitly specified. These may include pre- and post-conditions for different method invocations, other properties such as ’hasSameInterfaceAs <another class>’, ’hidesInterfaceOf <another class>’ etc. Languages such as could be used as it is or extended for this purpose.

**Language for Specifying Design Patterns:**

The approaches for OO Design proposed in this paper favor automatic techniques over manual ones for reasons described earlier. This means that we need a mechanism to be able to express design patterns in a format amenable to be read and interpreted by programs. Some attempts have been made at defining such pattern description languages. One of these or some variation of these could be used to express design patterns in a formal language.

**Comparison of Software Designs:**

Once we have alternative designs available, they need to be compared to arrive at the best one.

Each design may consist of multiple design patterns. The criteria here would not be to simply count the number of design patterns used but to evaluate the interaction between patterns and also between other design elements used. This would involve an understanding of good and bad design interactions and an ability to identify them in a given design. The final challenge would be to do it automatically.

**5.1 Structural Diagram**

**Class Diagram:**

Class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship.



FIGURE 5.1.1 CLASS DIGRAM SHOWING THEIR CORRESPONDING ATTRIBUTES AND OPERATIONS

**5.2 Component Diagram:**

Component diagrams fall under the category of an implementation diagram, a kind of diagram that models the implementation and deployment of the system. A Component Diagram, in particular, is used to describe the dependencies between various software components such as the dependency between executable files and source files. This information is similar to that within make files, which describe source code dependencies and can be used to properly compile an application.

**5.3 Deployment Diagram**

Deployment diagrams are another model in the implementation diagram category. The Deployment diagram models the hardware used in implementing a system and the association between those hardware components. Components can also be shown on a Deployment diagram to show the location of their deployment. Deployment diagrams can also be used early on in the design phase to document the physical architecture of a system.

**5.4 Behavioral Diagrams**

**Use Case Diagram:**

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include:

* Providing a high-level view of what the system does
* Identifying the users ("actors") of the system
* Determining areas needing human-computer interfaces

Use Cases extend beyond pictorial diagrams. In fact, text-based use case descriptions are often used to supplement diagrams, and explore use case functionality in more detail.



**FIGURE 5.4.1 USECASES SHOWING USER AND THEIR FUNCTIONALITY**

**5.5 Sequence Diagram**

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time.



FIGURE 5.5.1 SEQUENCE DIAGRAM SHOWING OBJECTS AND THEIR CORRESPONDING MESSAGES

**5.6 Collaboration Diagram**

Collaboration diagrams model the interactions between objects. This type of diagram is a cross between an object diagram and a sequence diagram. It uses free-form arrangement of objects which makes it easier to see all iterations involving a particular object.



**5.7 State chart Diagram** State diagrams, are used to document the various modes ("State") that a class can go through, and the events that cause a state transition.

**5.8 Activity Diagram**

Activity diagrams are used to document workflows in a system, from the business level down to the operational level. The general purpose of Activity diagrams is to focus on flows driven by internal processing vs. external events.

**CHAPTER-6**

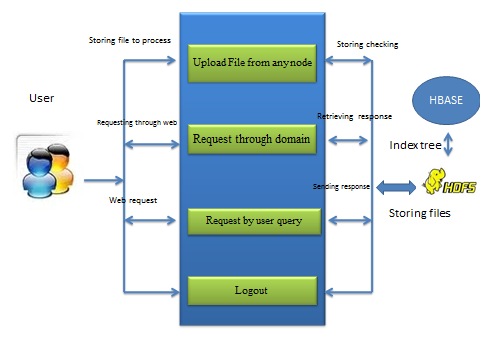
**ARCHITECTURE**

Project architecture represents no of components we are using as part of our project. The architecture describes about the flow of the request processing in the project. Describes about the communication of the components.

Generally in 3-Tier Architecture

* 1.Client(Cygwin)
* 2.MapReduce
* 3.Back end (HDFS)

So it explain like Requests will be taken from the Client and it will be process in mapreduce , the mapreduce will contact the HDFS and process the request.The request will be get back to the map reduce and in turn it will send to the client nothing but Browser.

****

CHAPTER-7

PROGRAMMING CODE

CalculateDistinct.java:

**import** java.io.IOException;

**import** java.util.\*;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.conf.\*;

**import** org.apache.hadoop.io.\*;

**import** org.apache.hadoop.mapred.\*;

**import** org.apache.hadoop.util.\*;

**public** **class** CalculateDistinct {

**public** **static** **class** Map **extends** ~~MapReduceBase~~ **implements** ~~Mapper~~<LongWritable,Text,Text,IntWritable> {

**private** **final** **static** IntWritable *one* = **new** IntWritable(1);

**private** Text word = **new** Text("");

**public** **void** map(LongWritable key, Text value, OutputCollector<Text,IntWritable> output, Reporter reporter) **throws** IOException {

word.set(value.toString());

output.collect(word,*one*);

}

}

**public** **static** **class** Reduce **extends** ~~MapReduceBase~~ **implements** ~~Reducer~~<Text, IntWritable, Text, IntWritable> {

**public** **void** reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) **throws** IOException {

**int** sum = 0;

**while** (values.hasNext()) {

sum += 1;

values.next();

}

**if**(sum==1){

output.collect(key, **new** IntWritable(sum));

}}

}

**public** **static** **void** main(String[] args) **throws** Exception {

~~JobConf~~ conf = **new** JobConf(CalculateDistinct.**class**);

conf.setJobName("Calculate Distinct");

conf.setOutputKeyClass(Text.**class**);

conf.setOutputValueClass(IntWritable.**class**);

conf.setMapperClass(Map.**class**);

conf.setReducerClass(Reduce.**class**);

conf.setInputFormat(~~TextInputFormat~~.**class**);

conf.setOutputFormat(~~TextOutputFormat~~.**class**);

~~FileInputFormat~~.*setInputPaths*(conf, **new** Path("DvIN1"));

FileOutputFormat.*setOutputPath*(conf, **new** Path("Dtss"));

JobClient.*runJob*(conf);

}

}

grep.java:

**import** java.util.Random;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.FileSystem;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapred.\*;

**import** org.apache.hadoop.mapred.lib.\*;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

/\* Extracts matching regexs from input files and counts them. \*/

**public** **class** Grep **extends** Configured **implements** Tool {

**private** Grep() {} // singleton

**public** **int** run(String[] args) **throws** Exception {

System.*out*.println("Grep <inDir> <outDir> <regex> [<group>]");

Path tempDir =

**new** Path("grep-temp-"+

Integer.*toString*(**new** Random().nextInt(Integer.*MAX\_VALUE*)));

~~JobConf~~ grepJob = **new** JobConf(getConf(), Grep.**class**);

**try** {

grepJob.setJobName("grep-search");

~~FileInputFormat~~.*setInputPaths*(grepJob, "DvIN");

grepJob.setMapperClass(RegexMapper.**class**);

grepJob.set("mapred.mapper.regex", "[a-b]+");

**if** (args.length == 4)

grepJob.set("mapred.mapper.regex.group","Dash");

grepJob.setCombinerClass(~~LongSumReducer~~.**class**);

grepJob.setReducerClass(~~LongSumReducer~~.**class**);

FileOutputFormat.*setOutputPath*(grepJob, tempDir);

grepJob.setOutputFormat(~~SequenceFileOutputFormat~~.**class**);

grepJob.setOutputKeyClass(Text.**class**);

grepJob.setOutputValueClass(LongWritable.**class**);

JobClient.*runJob*(grepJob);

~~JobConf~~ sortJob = **new** JobConf(Grep.**class**);

sortJob.setJobName("grep-sort");

~~FileInputFormat~~.*setInputPaths*(sortJob, tempDir);

sortJob.setInputFormat(~~SequenceFileInputFormat~~.**class**);

sortJob.setMapperClass(~~InverseMapper~~.**class**);

sortJob.setNumReduceTasks(1); // write a single file

FileOutputFormat.*setOutputPath*(sortJob, **new** Path("GrepOuts"));

sortJob.setOutputKeyComparatorClass // sort by decreasing freq

(LongWritable.DecreasingComparator.**class**);

JobClient.*runJob*(sortJob);

}

**finally** {

FileSystem.*get*(grepJob).delete(tempDir, **true**);

}

**return** 0;

}

**public** **static** **void** main(String[] args) **throws** Exception {

**int** res = ToolRunner.*run*(**new** Configuration(), **new** Grep(), args);

System.*exit*(res);

}

}

**CHAPTER-8**

**TESTING**

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.

**Defects and failures:**

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirements gaps, e.g., unrecognized requirements that result in errors of omission by the program designer. A common source of requirements gaps is [non-functional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) such as [testability](http://en.wikipedia.org/wiki/Testability), [scalability](http://en.wikipedia.org/wiki/Scalability), [maintainability](http://en.wikipedia.org/wiki/Maintainability), [usability](http://en.wikipedia.org/wiki/Usability), [performance](http://en.wikipedia.org/wiki/Computer_performance), and [security](http://en.wikipedia.org/wiki/Computer_security). Software faults occur through the following process. A programmer makes an [error](http://en.wikipedia.org/wiki/Human_error) (mistake), which results in a [defect](http://en.wikipedia.org/wiki/Fault_%28technology%29) (fault, bug) in the software [source code](http://en.wikipedia.org/wiki/Source_code). If this defect is executed, in certain situations the system will produce wrong results, causing a [failure](http://en.wikipedia.org/wiki/Failure). Not all defects will necessarily result in failures. For example, defects in [dead code](http://en.wikipedia.org/wiki/Dead_code) will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new [hardware](http://en.wikipedia.org/wiki/Hardware) platform, alterations in [source data](http://en.wikipedia.org/wiki/Source_data) or interacting with different software. A single defect may result in a wide range of failure symptoms.

**Compatibility:**

A frequent cause of software failure is [compatibility](http://en.wikipedia.org/wiki/Computer_compatibility) with another application or new [operating system](http://en.wikipedia.org/wiki/Operating_system) (or, increasingly [web browser](http://en.wikipedia.org/wiki/Web_browser) version). In the case of lack of [backward compatibility](http://en.wikipedia.org/wiki/Backward_compatibility) this can occur because the programmers have only considered coding the programs for, or testing the software, on the latest operating system they have access to or else, in isolation (no other conflicting applications running at the same time) or under 'ideal' conditions ('unlimited' memory; 'superfast' processor; latest operating system incorporating all updates, etc). In effect, everything is running "as intended" but only when executing at the same time on the same machine with the particular combination of software and/or hardware. These are some of the hardest failures to predict, detect and test for and many are therefore discovered only after release into the larger world with its largely unknown mix of applications, software and hardware. It is likely that an experienced programmer will have had exposure to these factors through "[co-evolution](http://en.wikipedia.org/wiki/Co-evolution)" with several older systems and be much more aware of potential future compatibility problems and therefore tend to use tried and tested functions or instructions rather than always the latest available which may not be fully compatible with earlier mixtures of software/hardware. This could be considered a prevention oriented strategy that fits well with the latest testing phase suggested by [Dave Gelperin](http://en.wikipedia.org/wiki/Dave_Gelperin) and [William C. Hetzel](http://en.wikipedia.org/wiki/William_C._Hetzel) cited below

**Input combinations and preconditions:**

A problem with software testing is that testing under all combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of [defects](http://en.wikipedia.org/wiki/Software_bug) in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, [non-functional](http://en.wikipedia.org/wiki/Non-functional_requirements) dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*) -- for example, usability, [scalability](http://en.wikipedia.org/wiki/Scalability), [performance](http://en.wikipedia.org/wiki/Computer_performance), [compatibility](http://en.wikipedia.org/wiki/Backward_compatibility), reliability -- can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

**Static vs. dynamic testing:**

There are many approaches to software testing. [Reviews](http://en.wikipedia.org/wiki/Code_review), [walkthroughs](http://en.wikipedia.org/wiki/Software_walkthrough) or [inspections](http://en.wikipedia.org/wiki/Software_inspection) are considered as [static testing](http://en.wikipedia.org/wiki/Static_testing), whereas actually executing programmed code with a given set of [test cases](http://en.wikipedia.org/wiki/Test_case) is referred to as [dynamic testing](http://en.wikipedia.org/wiki/Dynamic_testing). The former can be, and unfortunately in practice often is, omitted, whereas the latter takes place when programs begin to be used for the first time - which is normally considered the beginning of the testing stage. This may actually begin before the program is 100% complete in order to test particular sections of code (modules or discrete functions). For example, [Spreadsheet](http://en.wikipedia.org/wiki/Spreadsheet) programs are, by their very nature, tested to a large extent "[on the fly](http://en.wikipedia.org/wiki/On_the_fly)" during the build process as the result of some calculation or text manipulation is shown interactively immediately after each formula is entered.

**Software verification and validation:**

Software testing is used in association with [verification and validation](http://en.wikipedia.org/wiki/Verification_and_Validation_%28software%29):

[**Verification**](http://en.wikipedia.org/wiki/Verification)**:** Have we built the software right (i.e., does it match the specification?)? It is process based.

[**Validation**](http://en.wikipedia.org/wiki/Validation)**:** Have we built the right software (i.e., is this what the customer wants?)? It is product based.

**The software testing team:**

Software testing can be done by [software testers](http://en.wikipedia.org/wiki/Software_tester). Until the 1950s the term "software tester" was used generally, but later it was also seen as a separate profession. Regarding the periods and the different goals in software testing there have been established different roles: test lead/manager, test designer, tester, test automater/automation developer, and test administrator.

**Software Quality Assurance (SQA):**

Though controversial, software testing may be viewed as an important part of the [software quality assurance](http://en.wikipedia.org/wiki/Software_quality_assurance) (SQA) process. In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in defect rate. What constitutes an acceptable defect rate depends on the nature of the software. An arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than [mission critical](http://en.wikipedia.org/wiki/Mission_critical) software such as that used to control the functions of an airliner. Although there are close links with SQA, testing departments often exist independently, and there may be no SQA function in some companies.

Software Testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, QA is the implementation of policies and procedures intended to prevent defects from occurring in the first

**Testing methods:**

Software testing methods are traditionally divided into [black box testing](http://en.wikipedia.org/wiki/Black_box_testing) and [white box testing](http://en.wikipedia.org/wiki/White_box_testing). These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

**8.1 Black box testing**

[Black box testing](http://en.wikipedia.org/wiki/Black_box_testing) treats the software as a black box without any knowledge of internal implementation. Black box testing methods include [equivalence partitioning](http://en.wikipedia.org/wiki/Equivalence_partitioning), [boundary value analysis](http://en.wikipedia.org/wiki/Boundary_value_analysis), [all-pairs testing](http://en.wikipedia.org/wiki/All-pairs_testing), [fuzz testing](http://en.wikipedia.org/wiki/Fuzz_testing), [model-based testing](http://en.wikipedia.org/wiki/Model-based_testing), [traceability matrix](http://en.wikipedia.org/wiki/Traceability_matrix), [exploratory testing](http://en.wikipedia.org/wiki/Exploratory_testing) and specification-based testing.

**Specification-based testing:**

Specification-based testing aims to test the functionality according to the requirements. Thus, the tester inputs data and only sees the output from the test object. This level of testing usually requires thorough test cases to be provided to the tester who then can simply verify that for a given input, the output value (or behavior), is the same as the expected value specified in the test case. Specification-based testing is necessary but insufficient to guard against certain risks.

Advantages and disadvantages:

The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code MUST have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers don't. BUT, on the other hand, black box testing is like a walk in a dark labyrinth without a flashlight, because the tester doesn't know how the back end was actually constructed. That's why there are situations when 1. A black box tester writes many test cases to check something that can be tested by only one test case and/or 2. Some parts of the back end are not tested at all therefore; black box testing has the advantage of an unaffiliated opinion on the one hand and the disadvantage of blind exploring on the other.

**8.2 White box testing**

[White box testing](http://en.wikipedia.org/wiki/White_box_testing), by contrast to black box testing, is when the tester has access to the internal data structures and algorithms (and the code that implement these)

Types of white box testing

The following types of white box testing exist:

* [API testing](http://en.wikipedia.org/wiki/Api_testing) - Testing of the application using Public and Private APIs.
* [Code coverage](http://en.wikipedia.org/wiki/Code_coverage) - creating tests to satisfy some criteria of code coverage. For example, the test designer can create tests to cause all statements in the program to be executed at least once.
* [Fault injection](http://en.wikipedia.org/wiki/Fault_injection) methods.
* [Mutation testing](http://en.wikipedia.org/wiki/Mutation_testing) methods.
* [Static testing](http://en.wikipedia.org/wiki/Static_testing) - White box testing includes all static testing.

**Code completeness evaluation:**

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important [function points](http://en.wikipedia.org/wiki/Function_points) have been tested.

Two common forms of code coverage are:

* Function coverage, which reports on functions executed and statement coverage, which reports on the number of lines executed to complete the test.

They both return coverage metric, measured as a percentage.

**8.3Grey Box Testing**

In recent years the term grey box testing has come into common usage. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Manipulating input data and formatting output do not qualify as grey-box because the input and output are clearly outside of the black-box we are calling the software under test. This is particularly important when conducting [integration testing](http://en.wikipedia.org/wiki/Integration_testing) between two modules of code written by two different developers, where only the interfaces are exposed for test. Grey box testing may also include [reverse engineering](http://en.wikipedia.org/wiki/Reverse_engineering#Reverse_engineering_of_software) to determine, for instance, boundary values or error messages.

**Acceptance testing:**

Acceptance testing can mean one of two things:

1. A [smoke test](http://en.wikipedia.org/wiki/Smoke_test) is used as an acceptance test prior to introducing a build to the main testing process.
2. Acceptance testing performed by the customer is known as [user acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing#User_acceptance_testing) (UAT).

**8.4Regression Testing**

Regression testing is any type of software testing that seeks to uncover [software regressions](http://en.wikipedia.org/wiki/Software_regression). Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically regressions occur as an [unintended consequence](http://en.wikipedia.org/wiki/Unintended_consequence) of program changes. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged.

**Non Functional Software Testing:**

Special methods exist to test non-functional aspects of software.

* [Performance testing](http://en.wikipedia.org/wiki/Software_performance_testing) checks to see if the software can handle large quantities of data or [users](http://en.wikipedia.org/wiki/Load_testing). This is generally referred to as software [scalability](http://en.wikipedia.org/wiki/Scalability). This activity of Non Functional Software Testing is often times referred to as Load Testing.
* [Usability testing](http://en.wikipedia.org/wiki/Usability_testing) is needed to check if the user interface is easy to use and understand.
* [Security testing](http://en.wikipedia.org/wiki/Security_testing) is essential for software which processes confidential data and to prevent [system intrusion](http://en.wikipedia.org/wiki/Backdoor_%28computing%29) by [hackers](http://en.wikipedia.org/wiki/Hacker_%28computer_security%29).
* [Internationalization and localization](http://en.wikipedia.org/wiki/Internationalization_and_localization) is needed to test these aspects of software, for which a [pseudo localization](http://en.wikipedia.org/wiki/Pseudolocalization) method can be used.

In contrast to functional testing, which establishes the correct operation of the software (correct in that it matches the expected behavior defined in the design requirements), non-functional testing verifies that the software functions properly even when it receives invalid or unexpected inputs. [Software fault injection](http://en.wikipedia.org/wiki/Fault_injection), in the form of [fuzzing](http://en.wikipedia.org/wiki/Fuzz_testing) is an example of non-functional testing. Non-functional testing, especially for software, is designed to establish whether the device under test can tolerate invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-handling routines. Various commercial non-functional testing tools are linked from the [Software fault injection](http://en.wikipedia.org/wiki/Fault_injection) page; there are also numerous open-source and free software tools available that perform non-functional testing.

**Testing process:**

A common practice of software testing is performed by an independent group of testers after the functionality is developed before it is shipped to the customer. This practice often results in the testing phase being used as [project](http://en.wikipedia.org/wiki/Project_management) buffer to compensate for project delays, thereby compromising the time devoted to testing. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

In counterpoint, some emerging software disciplines such as [extreme programming](http://en.wikipedia.org/wiki/Extreme_programming) and the [agile software development](http://en.wikipedia.org/wiki/Agile_software_development) movement, adhere to a "[test-driven software development](http://en.wikipedia.org/wiki/Test-driven_development)" model. In this process [unit tests](http://en.wikipedia.org/wiki/Unit_tests) are written first, by the [software engineers](http://en.wikipedia.org/wiki/Software_engineering) (often with [pair programming](http://en.wikipedia.org/wiki/Pair_programming) in the extreme programming methodology). Of course these tests fail initially; as they are expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

**Testing can be done on the following levels:**

* [Unit testing](http://en.wikipedia.org/wiki/Unit_testing) tests the minimal software component, or module. Each unit (basic component) of the software is tested to verify that the detailed design for the unit has been correctly implemented. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.
* [Integration testing](http://en.wikipedia.org/wiki/Integration_testing) exposes defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.
* [System testing](http://en.wikipedia.org/wiki/System_testing) tests a completely integrated system to verify that it meets its requirements.
* [System integration testing](http://en.wikipedia.org/wiki/System_integration_testing) verifies that a system is integrated to any external or third party systems defined in the system requirements.

Before shipping the final version of software, *alpha* and *beta* testing are often done additionally:

* Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.
* Beta testing comes after alpha testing. Versions of the software, known as [beta versions](http://en.wikipedia.org/wiki/Beta_version), are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or [bugs](http://en.wikipedia.org/wiki/Computer_bug). Sometimes, beta versions are made available to the open public to increase the [feedback](http://en.wikipedia.org/wiki/Feedback#In_organizations) field to a maximal number of future users.

Finally, [acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing) can be conducted by the end-user, customer, or client to validate whether or not to accept the product. Acceptance testing may be performed as part of the hand-off process between any two phases of development.

**Regression testing:**

After modifying software, either for a change in functionality or to fix defects, a [regression test](http://en.wikipedia.org/wiki/Regression_testing) re-runs previously passing tests on the modified software to ensure that the modifications haven't unintentionally caused a [regression](http://en.wikipedia.org/wiki/Software_regression) of previous functionality. Regression testing can be performed at any or all of the above test levels. These regression tests are often [automated](http://en.wikipedia.org/wiki/Test_automation).

More specific forms of regression testing are known as [sanity testing](http://en.wikipedia.org/wiki/Sanity_test), when quickly checking for bizarre behavior, and [smoke testing](http://en.wikipedia.org/wiki/Smoke_testing) when testing for basic functionality.

[Benchmarks](http://en.wikipedia.org/wiki/Benchmark_%28computing%29) may be employed during regression testing to ensure that the performance of the newly modified software will be at least as acceptable as the earlier version or, in the case of code [optimization](http://en.wikipedia.org/wiki/Optimization_%28computer_science%29), that some real improvement has been achieved.

**Testing Tools:**

Program testing and fault detection can be aided significantly by testing tools and [debuggers](http://en.wikipedia.org/wiki/Debugger). Types of testing/debug tools include features such as:

* Program monitors, permitting full or partial monitoring of program code including:
  + [Instruction Set Simulator](http://en.wikipedia.org/wiki/Instruction_Set_Simulator), permitting complete instruction level monitoring and trace facilities
  + [Program animation](http://en.wikipedia.org/wiki/Program_animation), permitting step-by-step execution and conditional [breakpoint](http://en.wikipedia.org/wiki/Breakpoint) at source level or in [machine code](http://en.wikipedia.org/wiki/Machine_code)
  + [code coverage](http://en.wikipedia.org/wiki/Code_coverage) reports
* Formatted [dump](http://en.wikipedia.org/wiki/Dump) or [Symbolic debugging](http://en.wikipedia.org/wiki/Symbolic_debugging), tools allowing inspection of program variables on error or at chosen points.
* [Benchmarks](http://en.wikipedia.org/wiki/Benchmark), allowing run-time performance comparisons to be made.
* [Performance analysis](http://en.wikipedia.org/wiki/Performance_analysis), or profiling tools that can help to highlight [hot spots](http://en.wikipedia.org/wiki/Hot_spot) and resource usage

Some of these features may be incorporated into an [integrated development environment](http://en.wikipedia.org/wiki/Integrated_development_environment) (IDE).

**Measuring software testing:**

Usually, quality is constrained to such topics as [correctness](http://en.wikipedia.org/wiki/Correctness), completeness, [security](http://en.wikipedia.org/wiki/Computer_security_audit),[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] but can also include more technical requirements as described under the [ISO](http://en.wikipedia.org/wiki/International_Organization_for_Standardization) standard [ISO 9126](http://en.wikipedia.org/wiki/ISO_9126), such as capability, [reliability](http://en.wikipedia.org/wiki/Reliability_engineering), [efficiency](http://en.wikipedia.org/wiki/Algorithmic_efficiency), [portability](http://en.wikipedia.org/wiki/Porting), [maintainability](http://en.wikipedia.org/wiki/Maintainability), compatibility, and [usability](http://en.wikipedia.org/wiki/Usability).

There are a number of common software measures, often called "metrics", which are used to measure the state of the software or the adequacy of the testing.

Testing artifacts Software testing process can produce several [artifacts](http://en.wikipedia.org/wiki/Artifact_%28software_development%29).

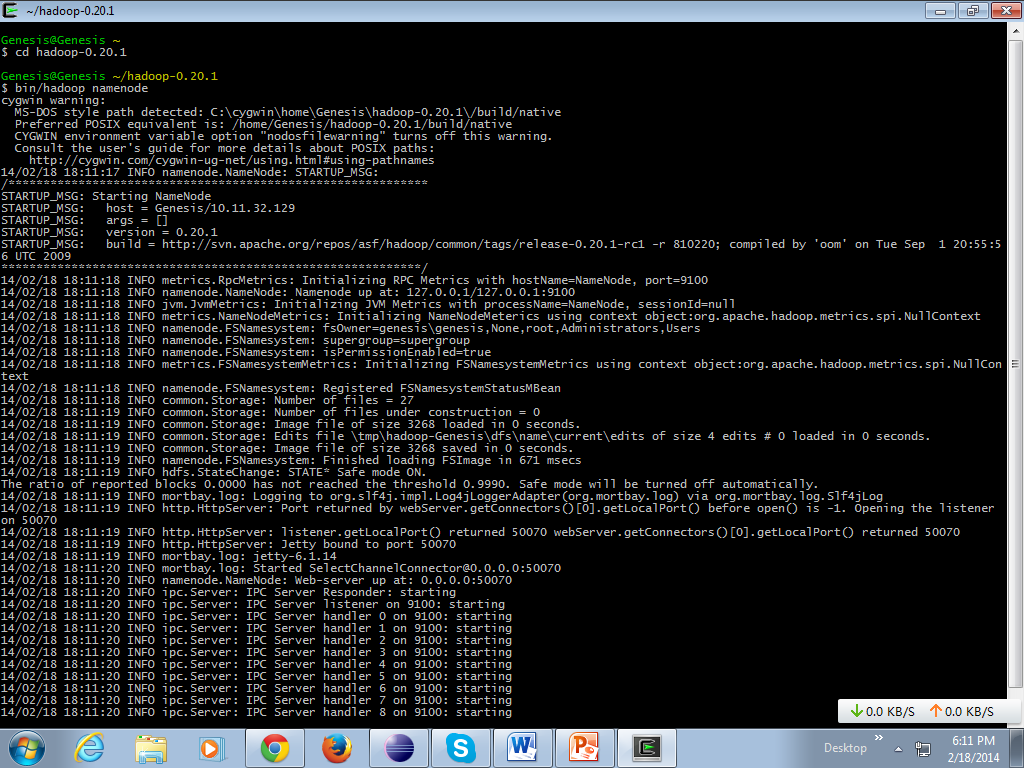
**8.5 TEST CASES:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Input** | **Output** |
| Starting name node | name Node details | Successfully started |
| Starting secondary name node | Secondary name Node details | Successfully started |
| Starting data node | Data node details | Successfully started |
| Starting job tracker node | Job tracker node Details | Successfully started |
| Starting task tracker node | Task tracker node Details | Successfully started |
| Directory Creation | Directory Details | Directory Created in HDFS |
| Uploading File | File Details | Successfully File Uploaded |

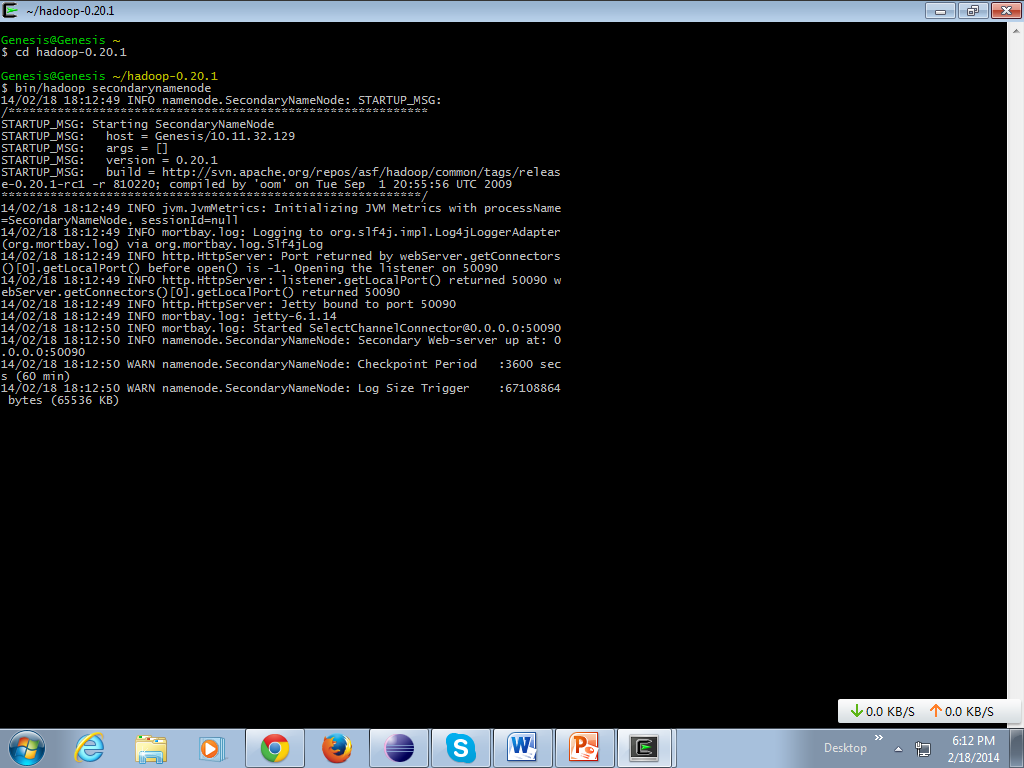
**CHAPTER-9**

**OUTPUT SCREENS**

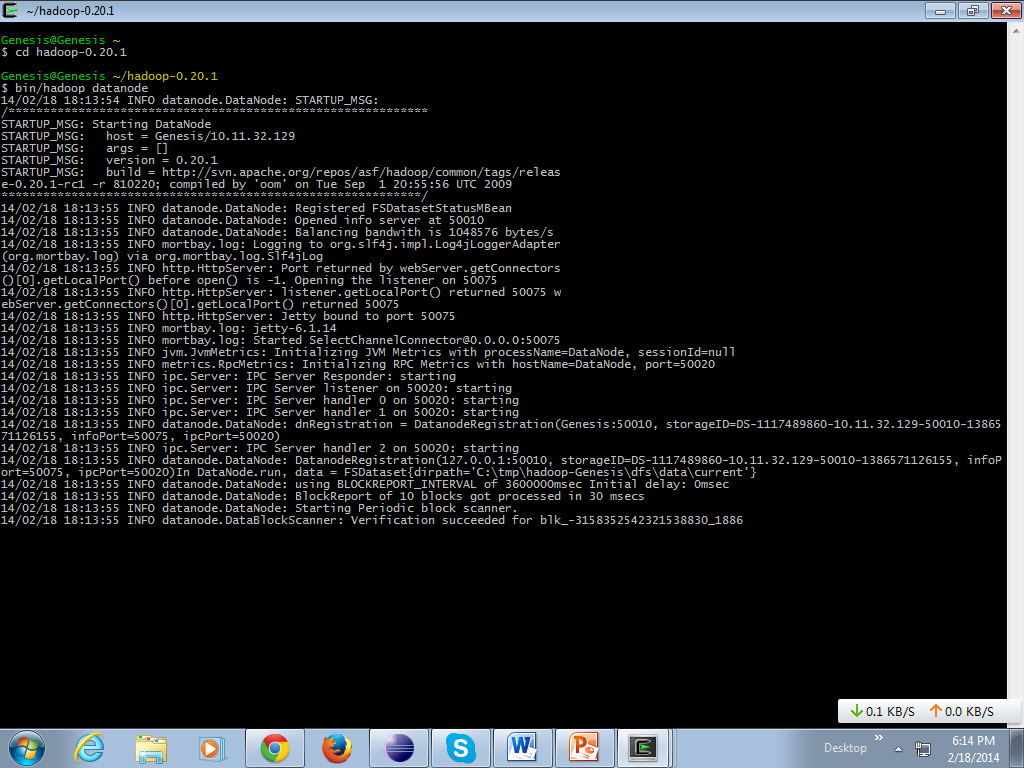
Name Node:



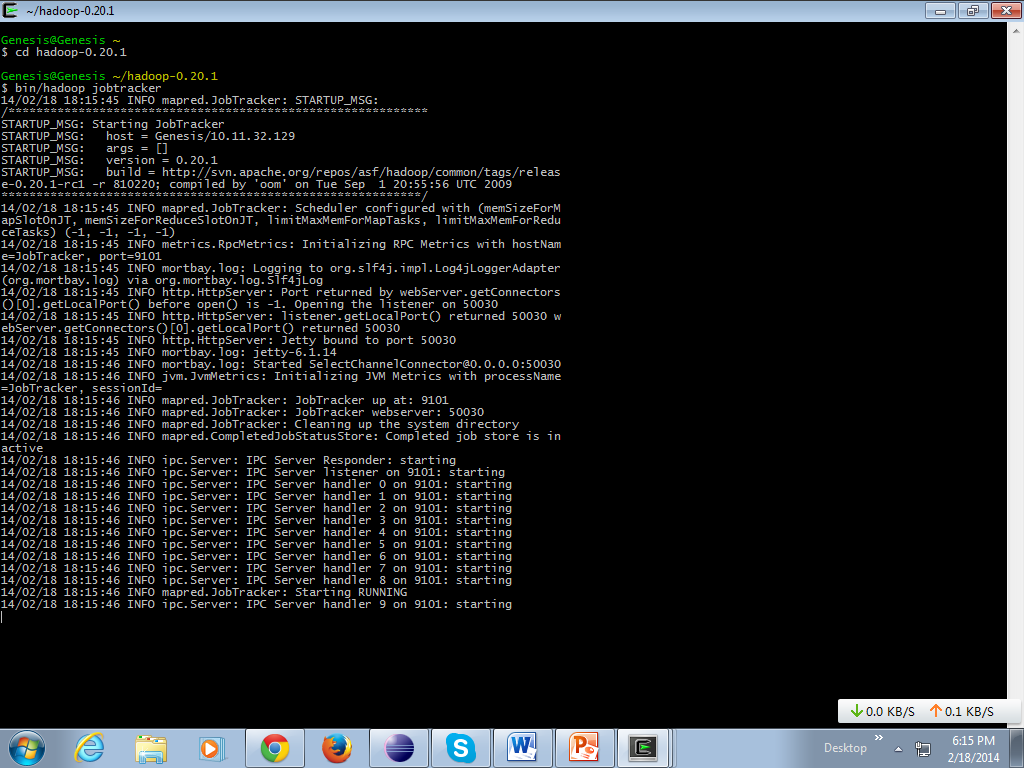
Secondary Name Node **:**



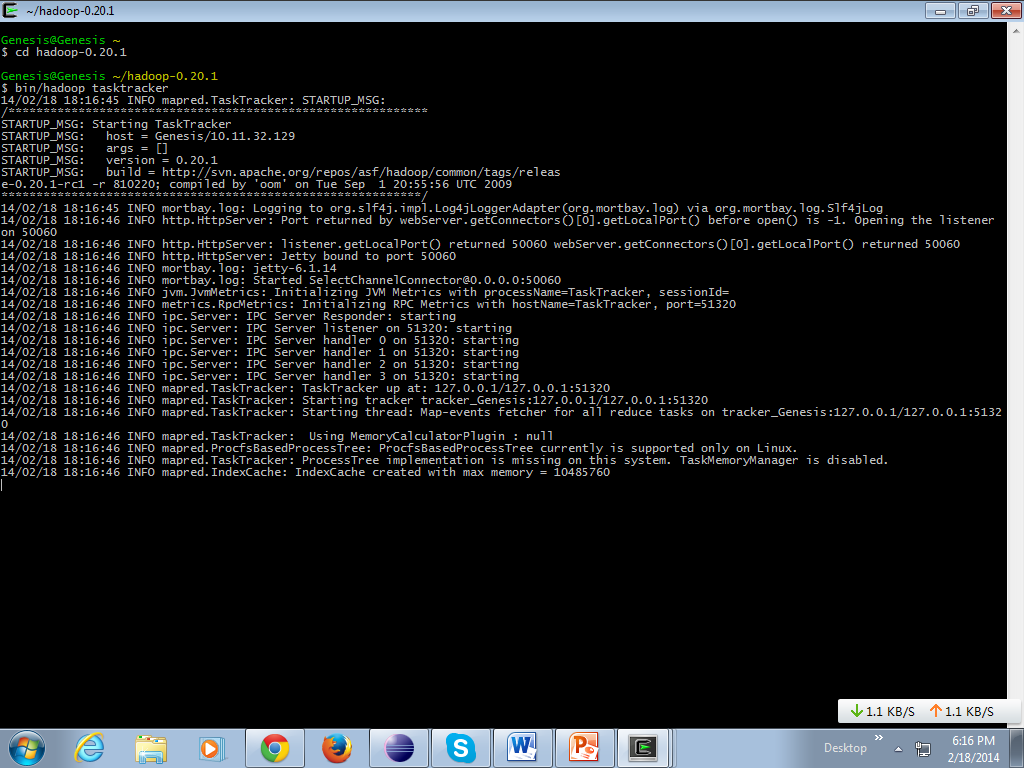
Data Node :



Job Tracker:



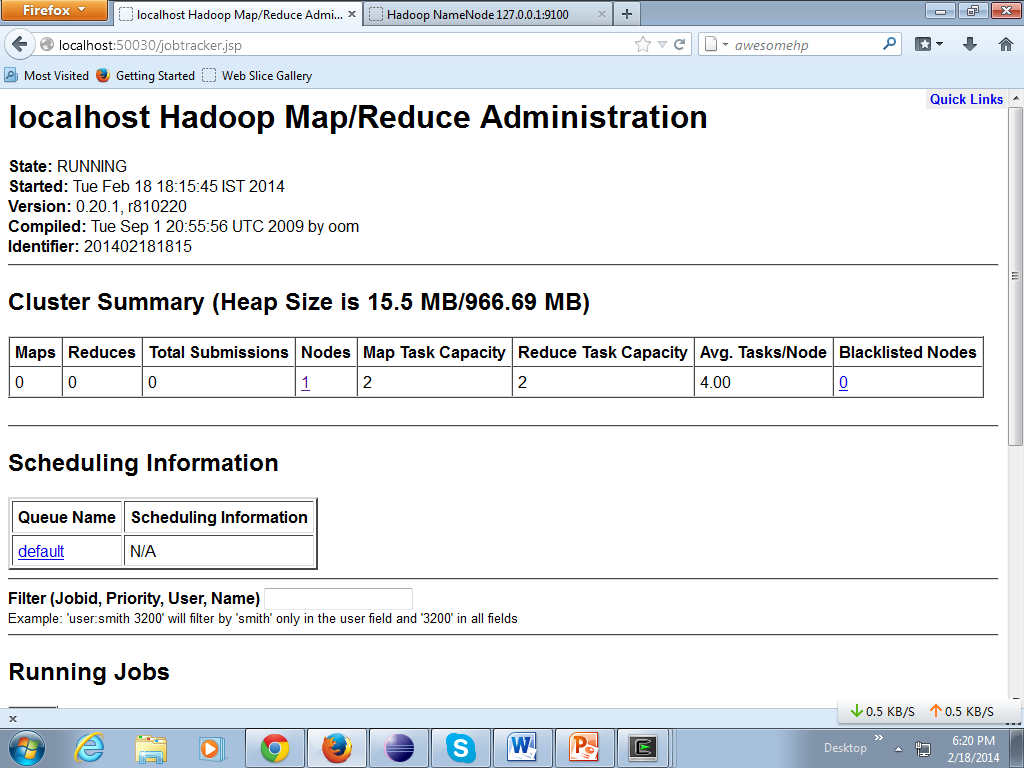
Task Tracker :



Name node :



Job Tracker :







CHAPTER-10

CONCLUSION:

The management efficiency when facing the huge amount of web services, thus, a web service management framework based on Hadoop is proposed. To improve the performance, we introduce Hadoop into this area, where HBase manages the functional and non-functional properties and MapReduce is assign to parallel handle the composition related tasks. The experiments show that the framework has good performance in retrieving process with the help of index mechanism. In web service selection experiments, the performance is far better than that of the serial method as the solution path become complex. However, we prove that this framework is more adaptive for managing large-scale web service data .

CHAPTER-11

FUTURE ENHANCEMENTS

The experiment management system is designed and developed for experimental teaching needs, which can achieve experimental management of computer, network and

information security professional students. The system has the following characters:

**A. Right Management :**

In order to maximize the safety of the protection system, the characters of system are a variety of users, and the user roles of system are divided into three categories: administrators,

teachers, students. The help is from the separation principle of roles and permissions, and flexibly set permissions for different roles. For example, we can arrange a test administrator to be responsible for this experimental courses, then the experimental teachers can only view and count related information of experimental course, other courses do not have permission .

**B. Experimental Control and Report Submittion :**

The instructor can specifY the actionable experimental project, and the system design experimental record, save the 1219 experimental project information that students have taken in pilot project, facilitate faculty management. Students can submit electronic files online lab report. Today, the popularity of digital information storage is easier for the compilations of experimental reports.

**CHAPTER-12**

**BIBLIOGRAPHY**

**List of Reference Documents**

* The Unified Modeling Language Users guide

By Grady Brooch

* Software Engineering, A practitioners approach

By Roger S Pressman

* Software Project Management

By Walker Royce

* Head First Series for Java

**WEB REFERENCES**

* <http://en.wikipedia.org/wiki/HDFS#Hadoop_distributed_file_system>
* <http://hadoop.apache.org/>
* <http://en.wikipedia.org/wiki/Mapreduce>
* <http://en.wikipedia.org/wiki/Main_Page>

**Appendix A**

Definitions of the quality characteristics follow.

|  |  |
| --- | --- |
| **CORRECTNESS** | 1. extent to which program satisfies specifications, fulfills user’s mission objectives |
| **EFFICIENCY** | 1. amount of computing resources and code required to perform function |
| **FLEXIBILITY** | 1. effort needed to modify operational program |
| **INTEGRITY/SECURITY** | 1. factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure |
| **INTEROPERABAILITY** | 1. effort needed to couple one system with another |
| **MAINTAINABILITY** | 1. ease of maintenance of the software itself |
| **PORTABILITY** | 1. ease of porting the software to another host |
| **RELIABILITY** | 1. factors required to establish the required reliability of the system |
| **REUSABILITY** | 1. extent to which it can be reused in another application |
| **TESTABILITY** | 1. effort needed to test to ensure performs as intended |
| **USABILITY** | 1. effort required to learn, operate, prepare input, interpret output |
| **AVAILABILITY** | 1. availability level for the system |