Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master in Computer Science	Spring semester, 2010 Open / Restricted access
Writer: Baodong Jia	…… (Writer's signature)
Faculty supervisor: Professor. PhD. Chunming Rong; PhD Candidate. M.Sc. Tomasz Wiktor Wlodarczyk External supervisor(s):	
Titel of thesis: Data Acquisition in Hadoop System	
Credits (ECTS): 30	
Key words: Hadoop; Data Acquisition; Chukwa; Performance; Real-time data; Historical data.	Pages: …… + enclosure: …… Stavanger, …….. Date/year

Contents

List of Figures

List of Tables

Abstract

Data has become more and more important these years, especially for big companies, and it is of great benefit to dig out useful information inside.

In Oil & Gas industry, there are a lot of data available, both in real-time and historical format. As the amount of data is huge, it is usually infeasible or very time consuming to process the data. Hadoop is introduced to solve this problem.

In order to perform Hadoop jobs, data must exist on the Hadoop filesystem, which brings the problem of data acquisition. In this thesis, two solutions are given out for data acquisition. The performance comparison is introduced afterwards, and solution based on Chukwa is proved to be better than the other solution.

1 Introduction

In this chapter, we will first come to the background of the thesis, and then describe the problem behind. After that, the aim of the thesis will be introduced, and an overview will be given out at the end of this chapter.

1.1 Background

There are a lot of applications generating data everyday. The data has been collected by many companies, and it could be of great benefit to dig out the useful information exists in the data ocean for production.

In Oil & Gas industry, drilling is usually performed by service companies such as Schlumberger, BHI, and Halliburton. Through placing a lot of sensors on drilling bits and platforms, these companies collect drilling data and make it available on their servers. For operators such as Statoil, Shell, and ConocoPhilips, the data provided by service providers is very important. The real time drilling data indicates the drilling status, and operators can get useful information by applying reasoning algorithm on the historical data. But as time goes on, data is accumulated. When the data set reaches a big size like several gigabytes, it is usually infeasible or very time consuming to do reasoning. MapReduce[1] system is introduced in this situation.

MapReduce-based system is used quite a lot in big companies like Google, Yahoo[2], and Facebook. It gathers distributed computational power and is quite efficient for reasoning on large data set. Being a member of MapReduce family, Hadoop[3] is open-source, reliable, and scalable distributed filesystem. As a MapReduce implementation it is used in many critical business areas[4]. After years of improvement, it is quite mature now, and also was adopted as the distributed platform in this master thesis.

In order to provide a predictable and secure solution to drilling and production operations, we initiated a project in UIS(University of Stavanger). There is a

cluster environment on the unix servers of unix lab in UIS, and it consists of 15 unix hosts. The cluster is established based on Hadoop, and it provides replicated storage and distributed computational power for this project. All the drilling data exists on servers of service providers of Statoil. After putting the data to cluster, reasoning algorithms are applied to find out interesting information. Also either suggestions or alarms will be given out according to the requirements of customers.

The main goal of this master thesis is to provide a good solution to feed both real time data and historical data to Hadoop cluster efficiently.

1.2 Problem Description

In the project, reasoning algorithms are applied to drilling data from Statoil to dig out useful information. In order to apply the algorithms, data must exists on the cluster. Algorithms are defined in ontology, and they are parsed by parsing program. Pig script[6] is generated after algorithm parsing. Pig[5] reads the pig script which involves data reading, creates Hadoop jobs, and submits them to cluster. Along with Hadoop, there are two web interfaces for browsing the cluster file system and tracking Hadoop jobs. When Map-Reduce jobs finish, result is stored in the cluster file system.

Data is really important in this project, and only drilling data is used. But in real life, data can be of different varieties, both real time data and historical data, and both in pull scenario and in push scenario.

When processing small number of data file with large size, Hadoop performs efficiently. But when it comes to a big amount of small files, which is the common case in real life, it takes a lot of extra time. As Hadoop does not support appending file content at this moment, data must be accumulated on local disk, and then copied to Hadoop cluster. Hadoop filesystem makes use of the disk of each host, and organizes them in a replicated way. The disk size of each host could be small, and is sometime even not large enough to store data for one job. Even yes, it still

takes a lot of time to copy the big data set to cluster, which could influence cluster efficiency rapidly. In this situation, data acquisition and feeding to cluster become very important to this project.

1.3 Aim of the Thesis

This thesis focuses on data acquisition for Hadoop system, which is a typical MapReduce-based system. Data can be both in real-time and historical, and the way how data servers provide data can be both in pull sense and in push sense.

In this thesis, data acquisition refers to acquiring data from data servers and feeding the acquired data to Hadoop cluster. So it can be divided into two steps: caching data on the local disk of a host, and transferring the data to Hadoop cluster.

The aim of thesis is to provide a good solution for data acquisition with refer to different types of data source in different sense. Two different data sources are used this thesis. One is data from twitter server, and the other is the data provided by services providers of Statoil. Difference solutions will be given out, and efficiency is emphasized as an very important factor.

1.4 Thesis Overview

In this chapter, we have introduced the background of this thesis, the problem addressed for this thesis, and also the aim of the thesis. In chapter 2, related technologies will be given out. Chapter 3 introduces the solution architecture and some discussion of the details. Chapter 4 focuses on the implementation of two possible solutions, and performances analysis is shown in Chapter 5. A conclusion will be given out in Chapter 6.

2 Technologies Behind

In this chapter, we will first give an introduction to DataStrom, and then provide some basic knowledge of Hadoop and Chukwa.

2.1 DataStorm

Cloud-based system has been wildly used to analyze huge amount of data efficiently, and there are many applications built on it. It uses MapReduce paradigm and is easy to scale. However, for applications built on it, designing, configuring, and deploying are still done manually.

DataStrom was proposed in this situation to provide solutions to design, config, and deploy cloud-based systems. DataStrom is an ontology-based framework . Based on Hadoop, it uses Web Ontology Language[7] as its modeling tool for automatic model validation and intelligent data analysis. DataStorm is an adoption and extension of BioSTORM[8], which is an ontology-driven framework for deploying JADE[9] agent system[10] from Stanford University. DataStrom framework consists of four layers, and the architecture of DataStorm is shown in Figure 1. This thesis serves as part of the project we are working on, and DataStrom is the framework supporting this project.

2.2 Hadoop

The Apache Hadoop project develops an open-source software for reliable, scalable, distributed computing[3], and original sponsored by Yahoo. It consists of many subprojects such as Hadoop Common, Chukwa, HBase, HDFS, and so on. Hadoop Common provides common utilities for the other subprojects. Through making use of a number of hosts, Hadoop establishes a super computational distributed system.

Hadoop provides a distributed file system, which stores application data in a replicated way, and also gives user high throughput ability of accessing the data

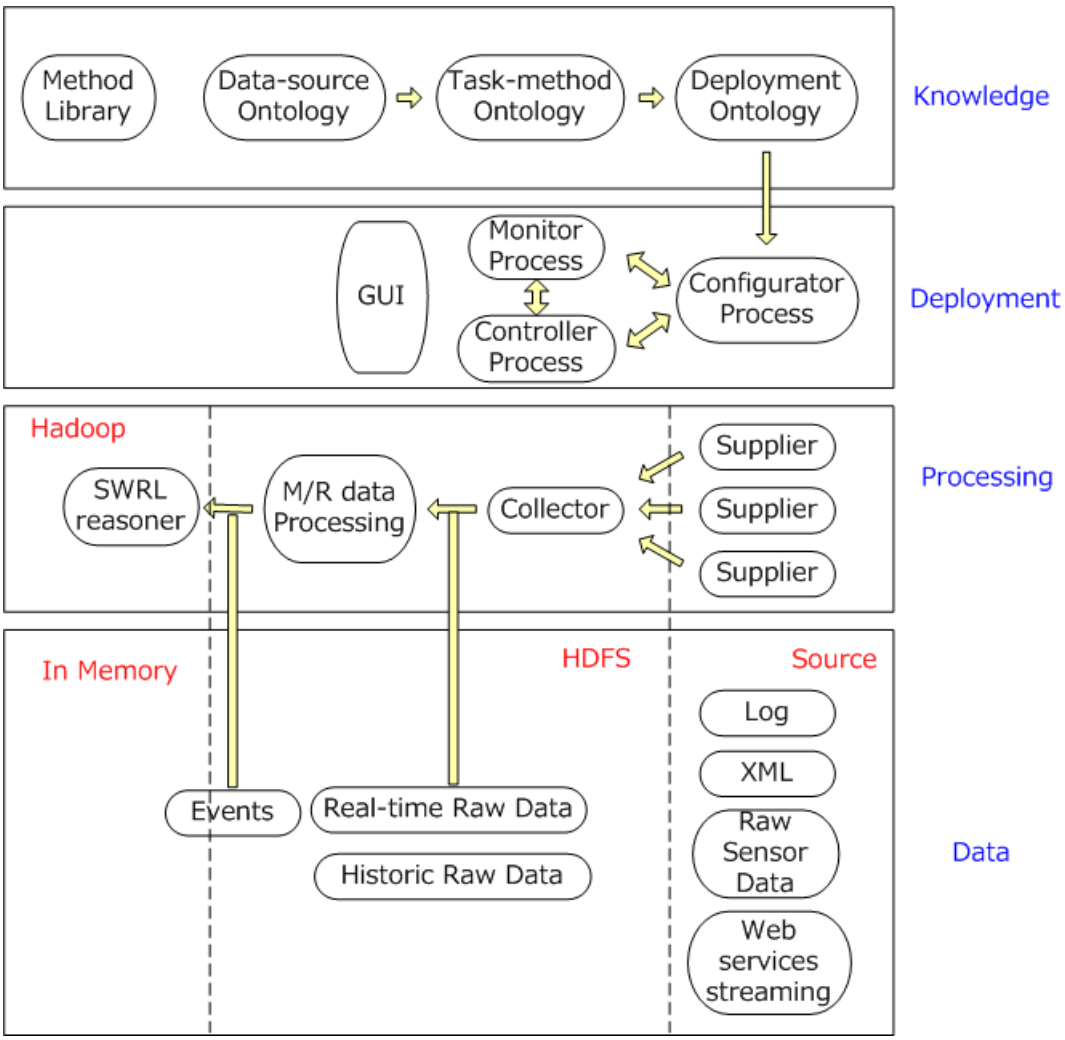

Figure 1: Framework Architecture of DataStrom

on HDFS. As a MapReduce system, it runs jobs very fast by using the aggregated super computational power. There are two web interfaces along with Hadoop. User can browse the HDFS and track jobs through these two interfaces in web browser.

The efficiency of Hadoop depends on the file size, number of files, the number of hosts in the cluster, bandwidth connecting the hosts and so on. Especially, Hadoop is not good at dealing with big amount of files with small file size.

Hadoop is implemented in Java, and it has been used in many big companies for production. In this thesis, it was chosen as the distributed computation platform.

2.3 Chukwa

Chukwa is an open source data collection system designed for monitoring large distributed system[11]. Been built on top of Hadoop, Chukwa inherits the scalability and robustness of it. In order to utilize the collected data in a better way, Chukwa provides a flexible and powerful toolkit to display, monitor, and analyze results.

Chukwa has four main parts: Agents, Collectors, MapReduce jobs, and HICC. The work flow of Chukwa system is shown in Figure 2.

Agents are responsible for collecting data through their adaptors. Adaptors interact with data source, and run inside of agents that are collecting data. As adaptors are quite flexible within agent, it is possible to have several adaptors for an agent to collect data from different source at the same time. Agents run on every host of Hadoop cluster, and data from different hosts may generate different data. Sometimes hosts might share a NFS[12] system, but there is no need to use Chukwa in this situation as all agents will collect the same data.

Collectors gather data from different agents through HTTP, and then write all the data into a Hadoop sequence file called sink file. Sink file consists of records collected by agents. After the sink file reaches a certain size or when a certain time

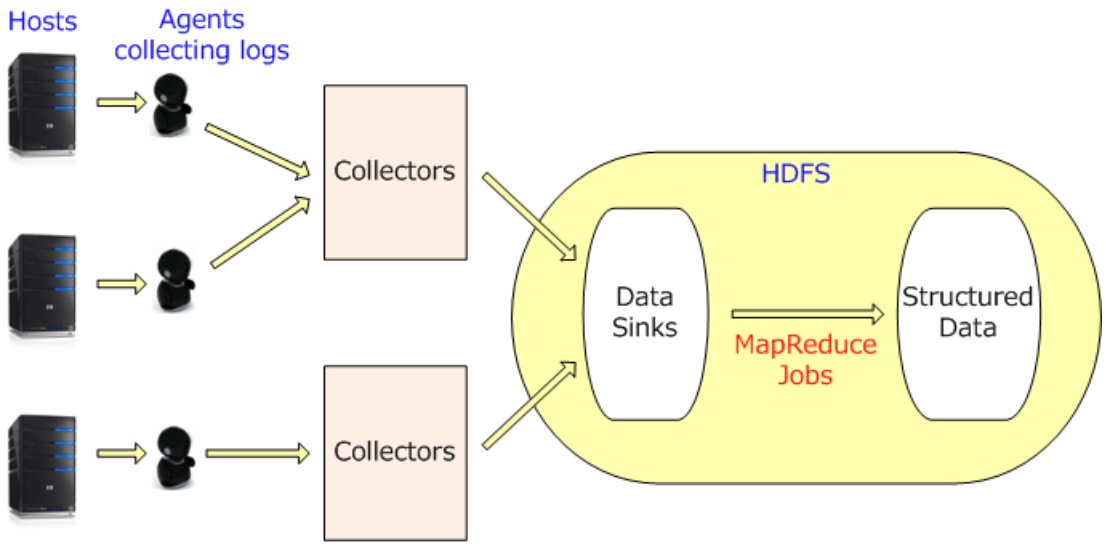

Figure 2: Work flow of Chukwa System

is out, the files will be renamed and made available for MapReduce processing. In this scene, collectors play an important role in reducing the number of HDFS files. Collectors also provide an uniform interface for adaptors, so that users do not need to care about the details of HDFS.

MapReduce job is made up of Archiving jobs and Demux jobs, and it makes the collected data organized and more convenient for analysis. ChukwaRecords are set of key-value pairs, and they are made available for Hadoop jobs to process after Demux jobs parses the collected data.

HICC stands for the Hadoop Infrastructure Care Center, and it is a web interface for displaying data. HICC makes it easier to monitor data.

Chukwa is used in one data acquisition solution in this thesis.

3 Data Acquisition

In this chapter, we will introduce two solutions of data acquisition. First, the traditional way of data acquisition will be given out. After that, a new way based Chukwa will be shown. According to our analysis, Chukwa based solutions proves to be better, and data acquisition from different data sources will be introduced in the end.

3.1 How to Acquire Data for Hadoop?

Hadoop cluster is used for high performance computing. Hadoop runs MapReduce jobs on the cluster, and stores the results on HDFS after the jobs finish. Data is fed to MapReduce jobs, and it must exist on HDFS before MapReduce jobs run.

There are several steps to follow to run a MapReduce job in Hadoop. The first step is to prepare the required data set for the job, and copy it to HDFS. Secondly, submit the job to Hadoop, either by executing a java program invoking Hadoop API, or parsing a pig script by Pig. After the job finishes, the result will be stored in a directory specified by user on HDFS. The last step is to get out the result on HDFS.

As we mentioned before, Hadoop is much more efficient when it comes to small number of files with big file size. But in real life, the most common cases are that data exists in a big number of files with small file size. When Hadoop jobs work on these data files, it takes a lot of extra time. HDFS does not support appending file content at the moment, and the only way is to acquire data from data sources and accumulate the data file to a proper size, and then copy it to HDFS.

The performance of data acquisition in this way is influenced by copying data to HDFS. When data file is relatively small, it takes almost no time to copy. But when there is a large file which is of several gigabytes, the copying time cannot be ignored. Especially in this thesis, where the replicate number of Hadoop is set to two and three, the file size is doubled and three times as much as the original size.

Solution based on Chukwa is proposed to solve this problem.

3.2 Architecture of Data Acquisition Solution using Chukwa

A solution based on Chukwa is proposed to overcome the problem of extra time generated by copying large file to HDFS. The architecture of the solution is shown in Figure 3.

As can be seen from the figure, there are five layers in the system, which are the physical hosts, Hadoop, Chukwa, Chukwa agents, and data from different sources.

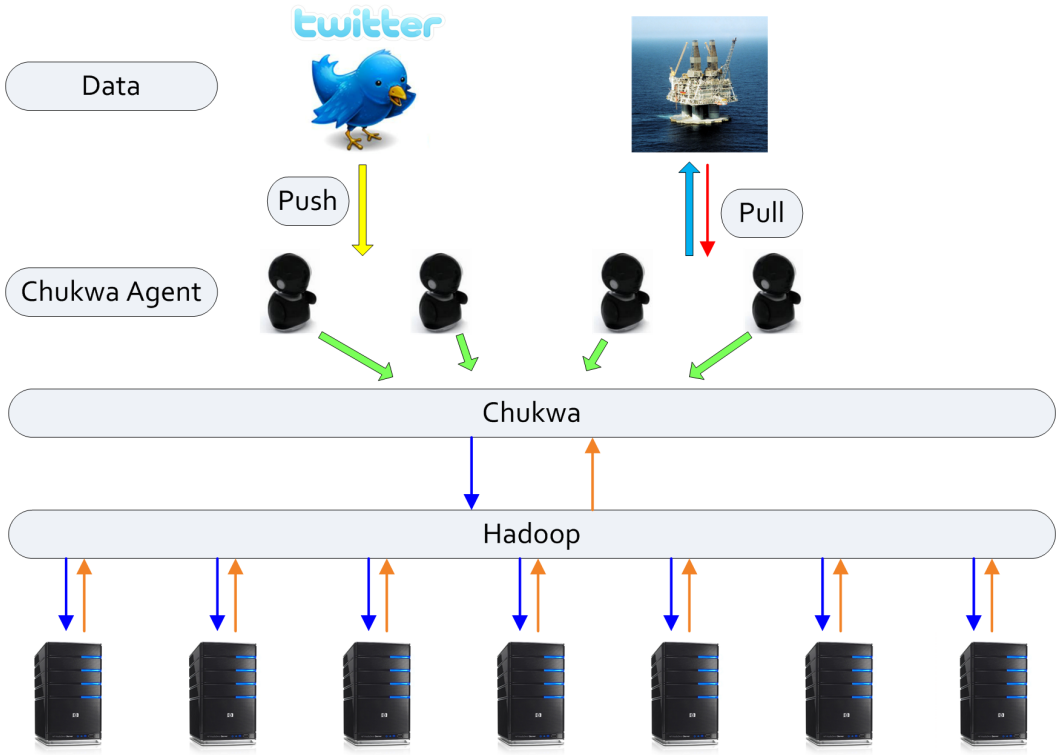

Figure 3: Architecture of Data Acquisition Solution

In this thesis, there are two different data sources. One is the real-time data from Twitter server, and the other is the drilling data from services provider of Statoil. Data is acquired by different java applications, and stored locally on the host of Hadoop.

Hadoop makes use of a handful hosts, and composes a reliable and scalable super computational cluster environment. Chukwa exists on top of Hadoop, and is responsible for feeding organized data into the cluster.

Chukwa agents play an role of transferring data that exists on each host to Chukwa through its adaptors. When the whole system runs, an adaptor of *File-Tailing* is added to Chukwa agent, which makes agent check if a file has been appended every two seconds. If yes, Chukwa agent will write the new added records to a temporary file on Chukwa. When the temporary file reaches a certain size, or a preset time is out, the file will be moved into the next step in Chukwa processing.

When data is transferred from Chukwa to HDFS of Hadoop, Hadoop jobs are ready to be executed.

3.3 Why Chukwa?

There are two different solutions for data acquisition. One is introduced in the beginning of this chapter, and the other one is based on Chukwa. For the first one, copying data from local host to Hadoop cluster takes a lot of extra time, which is not necessary, and sometimes even decreases the efficiency of Hadoop cluster rapidly. That is the main reason that Chukwa based solution is proposed.

3.3.1 Solution Without Chukwa

In order to put data into cluster, the data must be acquired first. After data acquisition programs get the data, they store it locally on the host. As HDFS does not support appending file content, the only possible way is to copy the acquired data into cluster filesystem. Figure 4 illustrates this very well.

Hadoop is quite efficient with processing small amount of big size files. But when there are a large number of small size files, it takes too much extra time. Example from an article[13] showing that it takes only 4 seconds to process a log

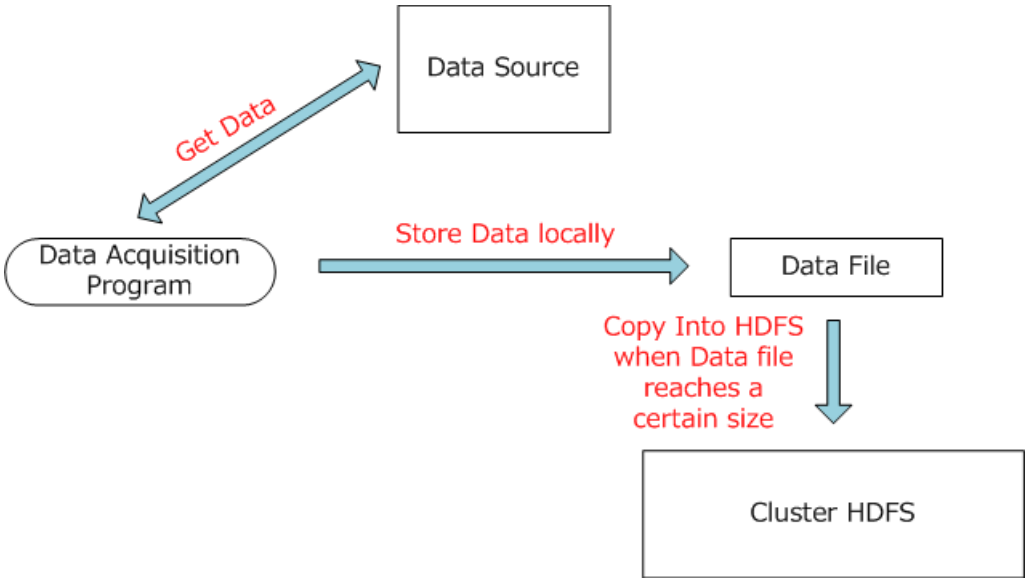

Figure 4: Data Acquisition without Chukwa

file of 100M by a simple java program, but 14 seconds when running it in Hadoop local mode, 30 seconds when running it in one host Hadoop MapReduce mode, and even slower when running it in two hosts Hadoop MapReduce mode connected by 10M network interface card.

In order to make Hadoop work efficiently, the file acquired from data sources need to be accumulated to a certain size on the host before copying. But we will meet two more problems in this situation. When the data file is small, it takes almost no time to copy data to cluster. As HDFS stores data in a replicate way, data file is actually copied several times, and it consumes a lot of extra time when it comes to a data file with big file size. Another problem is that Hadoop jobs can not process the data in a real-time way or within a time that user could endure. The reason is that it takes time for the data file to reach a certain size, and it might also postpone jobs because of copying data.

3.3.2 Solution With Chukwa

A better solution based on Chukwa is proposed, and it improves the performance of data acquisition through avoiding data copying.

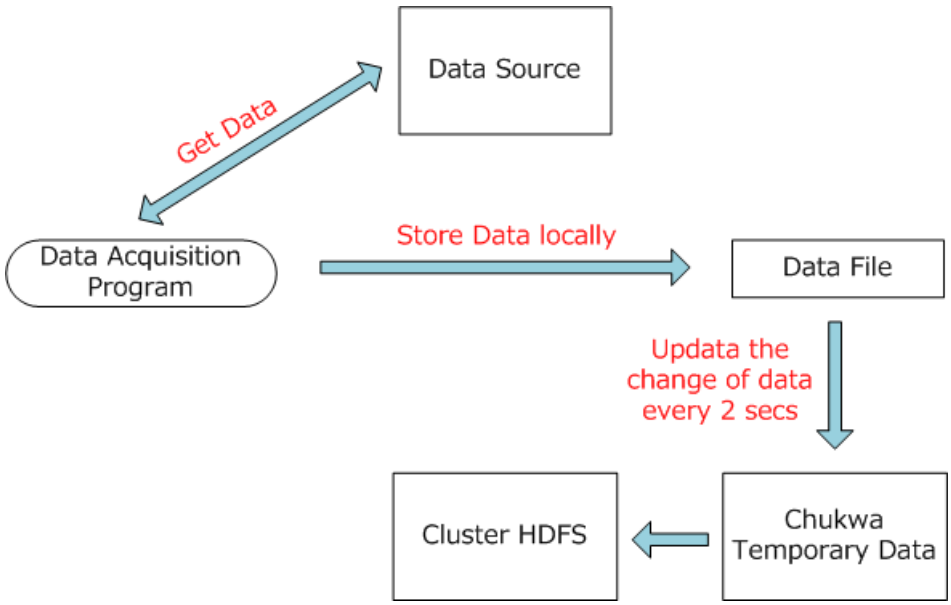

Figure 5: Data Acquisition with Chukwa

After data acquisition program gets the data from data source, it store the data locally to a data file on the host. Chukwa agent runs on the host, and an adaptor called *FileTailing* is added to it. The adaptor will check if the data file has been appended every 2 seconds, and update the change of data to a Chukwa temporary file. The processes are shown in Figure 5. After Chukwa organizes and processes the data, it forms the final data for Hadoop jobs.

Chukwa makes data available on the Hadoop cluster immediately after the data acquisition program gets data, and it saves time to copy data to HDFS. When the storage in a host is not enough for a large file, it becomes a problem for the first solution of data acquisition, as data file need to be accumulated to a certain size. As the data is already transferred to Chukwa system, the former data records can be deleted in the host, which solves the problem mentioned above.

As an open source data collection system, Chukwa also provides functions of displaying, monitoring, and analyzing results, which is convenient for users to make better use of the data collected.

3.4 Acquire Data From Different Data Sources

Data can be divided into different types according to different criteria. For example, data has two types which are historical data and real time data according to time. Data provider supplies their developers with different data accessing APIs, thus data can be divided into data fetched in pull scene and data fetched in push scene.

We have two data sources available in this thesis. One is the data exists on Twitter server, and it is in real-time format. The other one is the drilling data from service provider of Statoil in a test server. For security reasons, data fetched from this data source is only a historical data set, but real-time drilling data will be provided later.

3.4.1 Streaming Data from Twitter

Twitter provides social networking and microblogging services for its users, and is quite a popular platform now. There are around 50 million new records generated every day, and it still keeps growing up.

Twitter supplies a data stream for its developers. A data stream is a real-time, continuous, ordered sequence of items[14]. Through the data stream, Twitter gives user opportunities to do real-time search of twitter messages, and also do some reasoning for marketing.

Twitter4J[15] is a library for Twitter API based on Java. With the Twitter4j streaming API, developers will get the twitter data stream easily.

3.4.2 Historical Data from Service Provider of Statoil

Drilling is usually performed by the services companies such as Schlumberger, BHI and Halliburton. These companies collect data from drilling equipments and platforms, and make the data available on their servers through web interfaces or accessing API.

Operators such as Statoil, Shell, and ConocoPhilips, pay services companies, and get the drilling data from their servers.

In cooperation with Statoil, we get permission of accessing the drilling data from its service provider. For security reasons, only a test server is available, and data exists on this server is in historical format. Statoil provided a set of APIs for accessing the data servers, and different data can be fetched. In order to focus the reasoning part in the project, only time based drilling data is used in this thesis.

4 Implementation

In this chapter, the implementation of data acquisition will be introduced. We implement data acquisition in two different ways, both using Chukwa and stand-alone application without Chukwa involved. But first of all, the environment of implementation will be given out.

4.1 Implementation Environment

All implementations and experiments are based on Hadoop cluster. The Hadoop cluster consists of 15 unix hosts that existed at the unix lab of UIS. One of hosts is tagged as namenode, and the others are used datanodes. Namenode plays a role of center point in cluster, and all the information of storing data is saved on it. There is only one namenode in a Hadoop cluster, which could be the bottleneck of a cluster system. Datanodes provide the physical space for storing data. As HDFS stores data in a replicate way, same data may exists on different datanodes.

The operating system of the hosts is linux with centOS distribution. On top of the operating system is Hadoop with version 0.20.2. Chukwa is used on top of Hadoop for data acquisition, and the acquisition programs for both Chukwa based implementation and stand-alone application are written in Java in eclipse. Python script is used for generating test data set, and linux shell script is used for testing the programs.

4.1.1 Hadoop Setup

There are a lot of linux hosts in the unix lab of UIS, and we plan to use 15 of them for Hadoop cluster. Nodes in Hadoop have two types, which is namenode and datanode. Figure6 shows the structure of Hadoop nodes. As can be seen from the figure, *badne3* is used as namenode. *Badne4*, *badne5*, *badne6*, *badne7*, *badne8*, *pitter1*, *pitter2*, *pitter3*, *pitter4*, *pitter5*, *pitter6*, *pitter8*, *pitter9*, *pitter10* are used as datanode.

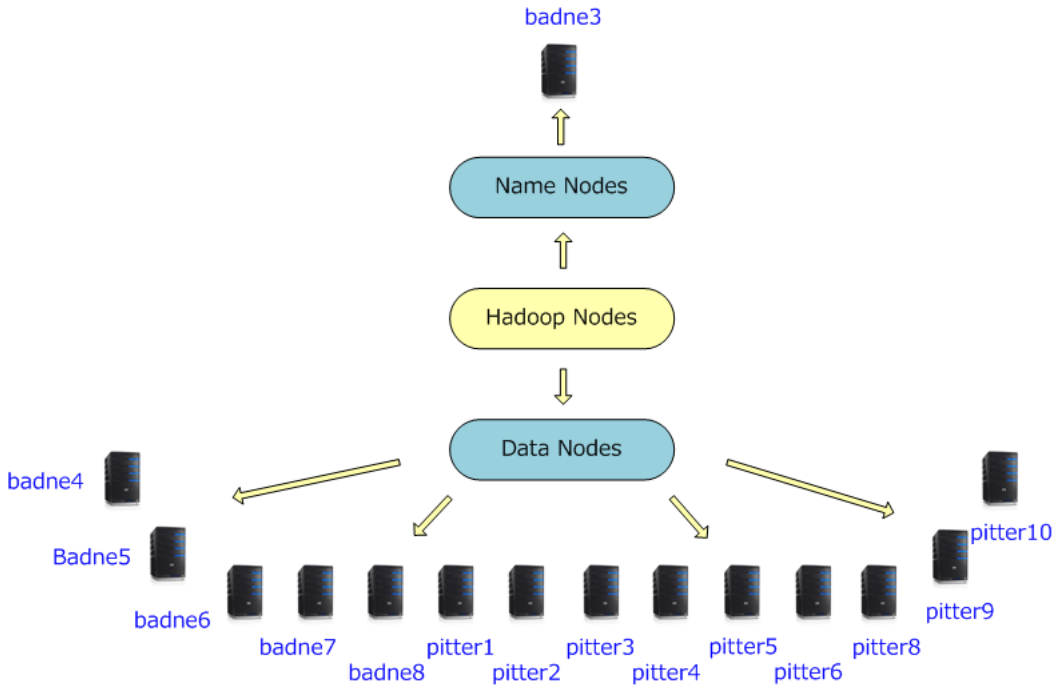

Figure 6: Node Structure of Hadoop

In order to install Hadoop, Java need to be installed on each host. Shell scripts are used for starting related Hadoop daemons remotely. SSH is used here, and it also need to be installed on each host.

There are three modes supported by Hadoop cluster. They are *local* mode, *pseudo-distributed* mode, and *fully-distributed* mode. The third one is used as the cluster mode in this thesis.

After downloading the Hadoop setup files, and before running the scripts, some configuration needs to be done. The three main configuration files are *conf/core-site.xml*, *conf/hdfs-site.xml*, *conf/mapred-site.xml*. *core-site.xml* is used to specify the address of namenode. The corresponding content for our cluster is showed as following.

```
<configuration>
  <property>
    <name>fs.default.name</name>
```

```
    <value>hdfs://badne3:9000</value>
  </property>
</configuration>
```

conf/hdfs-site.xml is used to config HDFS. The replica number, directory of HDFS, and also some other parameters are specified here. The content of our cluster is showed as following.

```
<configuration>
  <property>
    <name>dfs.data.dir</name>
    <value>/local/Hadoop/HadoopSpace/HadoopDataDir</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
</configuration>
```

As we can see, the replica number is 2 for our cluster, but it is changed later to 3 for performance comparison. The content of *conf/mapred-site.xml* is showed blow, and it indicates the addresses of job tracker.

```
<property>
    <name>mapred.job.tracker</name>
    <value>badne3:9001</value>
  </property>
</configuration>
```

After the configuration, we are ready to start the cluster. Two commands are mainly used to start Hadoop. The following one is used to format a new filesystem.

$$bin/Hadoopnamenode - format$$

The other one used is

$$bin/start - all.sh$$

Which starts all the nodes and job trackers of Hadoop. Now, Hadoop is ready to use.

4.1.2 Chukwa Setup

In order to install Chukwa, Java and Hadoop need to be installed on each host.

13 hosts are used in Chukwa, and they consists of agents and collects. Figure 7 shows the structure of Chukwa. As can be seen, *pitter1*, *pitter2*, *pitter3*, *pitter4*, *pitter5*, *pitter6*, *pitter8*, *pitter9*, *pitter10* are used as agent, and *badne5*, *badne6*, *badne7*, *badne8* are used as collector.

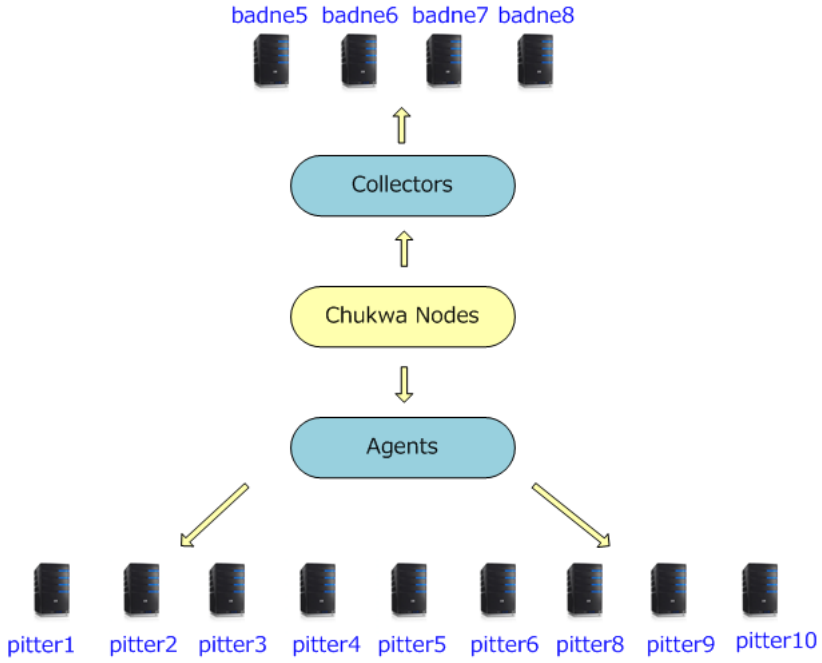

Figure 7: Node Structure of Chukwa

After downloading the Chukwa setup file, some configuration needs to be done.

The first task is to modify *conf/chukwa-env.sh*. Variable *JAVA_HOME* needs to be set correctly, and also *CHUKWA_LOG* and *CHUKWA_DIR* can be set to

specify the corresponding directories.

In *conf* directory, agents and collectors can be set. The file *agents* is used to list the address of all agents. When running Chukwa scripts, it will be used to start all agents. The content of *agents* of our Chukwa system is showed as following.

```
pitter1.ux.uis.no
pitter2.ux.uis.no
pitter3.ux.uis.no
pitter4.ux.uis.no
pitter5.ux.uis.no
pitter6.ux.uis.no
pitter8.ux.uis.no
pitter9.ux.uis.no
pitter10.ux.uis.no
```

The file *collectors* is used to list the address of all collectors, and its content is showed as following.

```
badne5.ux.uis.no
badne6.ux.uis.no
badne7.ux.uis.no
badne8.ux.uis.no
```

conf/chukwa-agent-conf.xml is used for agents settings. Some useful ones, such as the agent control port and Chukwa agent tag are shown below.

```
<property>
    <name>chukwaAgent.control.port</name>
    <value>9093</value>
    <description>The socket port number the agent's
    control interface can be contacted at.</description>
</property>
```

```
<property>
    <name>chukwaAgent.tags</name>
    <value>cluster="uisChukwa"</value>
    <description>The cluster's name for this agent</description>
</property>
```

conf/chukwa-collector-conf.xml is used for collectors settings. Two of the most important ones in this file is shown as following.

```
<property>
    <name>writer.hdfs.filesystem</name>
    <value>hdfs://badne3.ux.uis.no:9000/</value>
    <description>HDFS to dump to</description>
</property>
<property>
    <name>chukwaCollector.http.port</name>
    <value>8080</value>
    <description>The HTTP port num
    ber the collector will listen on</description>
</property>
```

The first one gives out the address of namenode of Hadoop, and the second one shows the accessing port of collectors.

After finishing all the settings, Chukwa agents and collectors are ready to start up. Command

$$bin/start - agents.sh$$

is used to start all the agents. Command

$$bin/start - collectors.sh$$

is used to start all the collectors.

When data collecting is finished, agents and collectors can be stopped using the following commands.

$$bin/stop - agents.sh$$

$$bin/stop - collectors.sh$$

One important thing that needs to be paid attention is the sequence. When starting Chukwa, collectors must be started first, and then agents can be started. The reason is that if the agents are started first, they actually collect data already. But as the collectors are not up and running yet, the data agents collected is missing. Same principle applies to stop Chukwa. Agents must be stopped first, and then collectors can be stopped.

4.2 Data Acquisition Using Chukwa

4.2.1 log4j

The Apache Logging Services Project provides an open-source software related to the logging of application behavior and free for people to use[17]. Log4j is the logging services for Java, and it provides a better way of debugging and analyzing code.

Log4j has three important components which are Logger, Appender, and Layout. There is only one root Logger, and it always exists. Appender is used to specify the location where all logs will be put. The available appenders are console, files, GUI components, and so on. One Logger can have more than one appenders. Layout is assigned to appenders, and it gives out the format of printing logs.

Log4j defines 5 levels for log, which are DEBUG, INFO, WARN, ERROR, and FATAL. Log can only be printed out when its level is high than the level defined in configuration file. When users want to print out a different level of information or print out nothing, the only thing they need to do is to modify the configuration file, but the program itself.

Log4j is used in the implementation of data acquisition programs in this thesis, both for saving records and debugging.

4.2.2 Acquire Data from Twitter

Twitter provides a data stream for its developers, and the data stream can be used through Twitter APIs. In order to make it easier for user to use the APIs, twitter4J supplies a set of APIs based on java for accessing twitter data. There is an example[16] available for streaming twitter data using twitter4J, and it is modified to save the data into a file locally on a host. The corresponding code is shown as following.

```
public void streamingData() throws TwitterException {
        StatusListener listener = new StatusListener() {
            public void onStatus(Status status) {
                String record = status.getUser().getName() + " : "
                        + status.getText();
                logger.info(record);
            }
            public void onDeletionNotice(
                    StatusDeletionNotice statusDeletionNotice) {
            }
            public void onTrackLimitationNotice(
                    int numberOfLimitedStatuses) {
            }
            public void onException(Exception ex) {
                ex.printStackTrace();
            }
        };
        TwitterStream twitterStream = new TwitterStreamFactory(listener)
```

```
                .getInstance(twitterID, twitterPassword);
        twitterStream.sample();
        try {
            Thread.sleep(1000000000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

In the code shown above, the streaming data is output using *logger.info* function. The configuration of logger is defined in the test program, as shown in the blow.

```
        Logger logger = Logger.getLogger(twChukwaApp.class);
        logger.setAdditivity(false);
        PatternLayout p1 = new PatternLayout("%m%n");
        RollingFileAppender a1 = null;
        try {
            a1 = new RollingFileAppender(p1, "./twChukwa.log");
        } catch (IOException e) {
            e.printStackTrace();
        }
        a1.setMaxBackupIndex(0);
        a1.setMaxFileSize("1GB");
        logger.addAppender(a1);
        logger.setLevel(Level.INFO);
```

A *RollingFileAppender* is created, and twitter streaming data is stored in *twChukwa.log*. *%m%n* means only the message is output, while the other information such as date and time is omitted. The maximum size of data file is set to 1GB, and there is no backup for this file. When data size reaches 1GB, the file will be emptied, and new data continues fill the file from the beginning. The level logging is set to *INFO*,

which means all messages except *DEBUG* messages will be output to the data file.

As the data file will be emptied after data file reaches the maximum file size, and Chukwa agent check the data file every 2 seconds. Some data that is generated between a Chukwa update and another update that happens after the data file is emptied might be ignored. Thus the quality of data acquired might be brought down.

4.2.3 Acquire Data from Service Providers of Statoil

Statoil provides the API for accessing the drilling data from its service providers. Data consists of real-time and historical drilling and monitoring data from drilling bits and platforms. Through using the API, we are able to fetch the data from the server they provided. As for secure purposes, the server is just a test server, and data existed there is historical data, but real-time one. But it is still enough for experiment for now.

In this thesis, we feel more interested in drilling data indicating depth of bits according to time stamps. and the corresponding code for fetching the data is as following.

```
String uidWell = "W-68953";
String uidWellbore = "B-69018";
String uidLog = "L-69019-Time";
String[] mne = { "TIME", "DBTM" };
try {
    log = connector.getLog(provider, uidWell, uidWellbore, uidLog,
            startTime, endTime, mne);
    if (log != null) {
        logdata = log.getLogData();
        List<String> logData = logdata.getData();
        String record = "";
```

```
            for (Iterator it = logData.iterator(); it.hasNext();) {
                String tmp = (String) it.next();
                String[] tmpArray = tmp.split(",");
                record = tmpArray[0] + ", " + tmpArray[1];
                logger.info(record);
            }
        } else
            System.out.println("No logData found!!");
    } catch (WITSMLConnectorException exp) {
        exp.printStackTrace();
    }
```

During my work in the *Computer Science project* course, a Java based GUI application was developed for fetching different data from the test server, and it provides a more flexible way of data fetching. In the code shown above, only time based depth data is fetched according to the need for now, and it is controlled by setting the value of variable *mne*. Well, wellbore, and log are also set to fixed value in order to implement the solution in an easy way for debugging. The implementation can be made more portable according to the need of project.

Data fetched from server is stored using *logger*, and the definition is the same as the one used for twitter.

4.2.4 Chukwa Adaptor Used

Data acquired has been stored in file *twChukwa.log*, and Chukwa agent plays an important role to put the acquired data into cluster. Chukwa agent works together with adaptors. One agent can have many adaptors, and the most common used ones are provided by Chukwa already. In this thesis, an adaptor called *FileTailing* is used. It detects if the content of the target file has been changed every 2 seconds. If yes, the new added content will be added in to Chukwa collector, and

then processed further more in Chukwa system.

The following command is used to add an *FileTailing* adaptor to Chukwa agent to collect data.

```
add filetailer.FileTailingAdaptor twData
/home/prosjekt/Hadoop/baodong/testProgram/twChukwa.log 0
```

The first filed of the command is an *add* adaptor keyword. Field 2 gives out the class of adaptor, and field 3 is the target file to monitor. Field 4 specifies the initial offset, which is usually set to 0.

4.3 Data Acquisition by Stand-alone Application

In the Stand-alone application, the way of acquiring data stays the same. The only difference is that data copying to HDFS is involved as HDFS does not support file content appending.

The corresponding code for copying acquired data to cluster is shown as following.

```
public void copyDataFile() {
        try {
            fsURI = new URI("hdfs://badne3:9000/");
            fs = FileSystem.get(fsURI, conf);
            ogFilePathSrc = new Path(filePath);
            ogFilePathDes = new Path(ogDataStreamFilename);
            if (fs.exists(ogFilePathDes)) {
                fs.delete(ogFilePathDes);
            }
            fs.copyFromLocalFile(ogFilePathSrc, ogFilePathDes);
        } catch (Exception e) {
            e.printStackTrace();
```

```
            System.exit(1);
        }
    }
```

Before copying, it will first check if the file is already on HDFS. If yes, it will delete the old one, and then copy the new one to HDFS using file copy API provided by Hadoop.

When it comes to big data file, data copying may consume a lot of time. As HDFS provides a replicate way of storing data, the actually time used may exceed the expectation of users.

4.4 Testing

The testing is based on data acquisition of Twitter real-time data stream.

After compiling the related code, a runable jar file is created, and it can be executed by java. In order to calculated the exact total time used by the test program, shell script is used as shown in Figure 8, which is named *twNormal.sh*.

```
#!/bin/sh
echo -e 'Start time of twNormalApp:\t' `date`
java -jar twNormalApp.jar 4800000
echo -e 'End time of twNormalApp.jar:\t' `date`
```

Figure 8: Shell Script of Executing a Test Program

Date command is used before and after the execution of test program to make time stamp. The actually run time can be calculated by these two values. As the minimum unit used for recording is second, precision of total time used is second, which is enough for performance analysis shown in the next chapter.

For *twChukwa.sh*, the content is almost the same as twNormal.sh, where the name of the jar file is changed.

In order to start the two test programs at the same time, another shell script called *runTest.sh* is made as shown in Figure 9.

```
#!/bin/sh
echo "Starting test..."
./twChukwa.sh > cLog &
./twNormal.sh > nLog &
echo "Test Started..."
```

Figure 9: Shell Script of Starting Test Programs at the Same Time

As can be seen from the figure, two test programs can not be started at exactly the same time. But through using the shell script, the time difference is minimized.

```
#!/usr/local/bin/python
i=1
f=file('testRecord20M','w')
while(i<1900000):
        print i
        result="hello world"
        f.write(result)
        i=i+1
f.close()
```

Figure 10: Python code for Generating test data

When testing the performance of copying data to cluster, Hadoop command is used, which is shown as following.

```
Hadoop fs -copyFromLocal testRecord1g baodong/compData
```

As we need different size of data files and it is very slow to accumulate data from twitter server, data generated by python is used for testing purpose.

Code shown in Figure 10 is used to generate a test data file of 20M. The number in *while* can be changed in order to generate different size of files.

5 Performance Analysis

In this chapter, we will give out the performance analysis of data acquisition implemented both using Chukwa, and stand-alone application. First, we will compare the quality of the data acquired, and then introduce the time differences with refer to small data size. As when data size is big, it plays an important role to copy data to HDFS, and we will focus on this in the last part of this chapter.

5.1 Quality of the Data Acquired in Different Ways

The quality of the data acquired is regards as one important factor for data acquisition. We did a lot of test with the quality of the data acquired in different ways. Figure 11 shows the size of the data acquired in a certain time. As can be seen from the figure, data acquisition in both ways got the almost the same data with no differences.

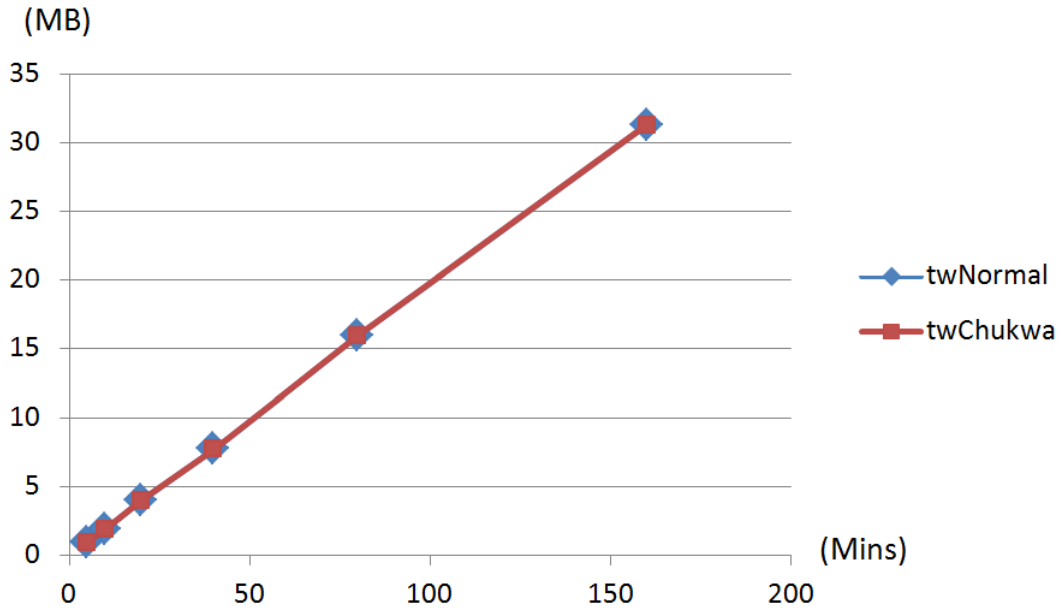

Figure 11: The Size of Data acquired by Time

In order to know whether they acquired exactly the same data, we look into

the data files generated using both ways, and the result is that they have different content at the beginning and the end of the files. For the content in between, they match perfectly. The main reason is that the two test programs can not be started at exactly the same time.

We start the two test programs one after another using linux shell script, which leads that the two programs can not be started at exactly the same time. The program that was started early always got more data than the other one, but only a few records more because the other one is started immediately after the first one. That is why the content of two data files differes at the beginning of the files. As both programs had the same time of data acquisition, but started at different point-in-time, they ended in at different point-in-time, which made the content of two data files also differes at the end of the files.

The maximum size of data file is set to 1GB when implementing data acquisition using Chukwa. When the data file exceeds 1GB, the whole file will be erased, and data accumulates from zero again. As Chukwa agent check the file content every 2 seconds, data generated between a Chukwa update and another update that happens after the data file is emptied might be ignored. This may reduces the quality of the data acquired, but it depends on the maximum size of data file and the type of application used. When the maximum size is very large, the data missed is tiny comparing to the maximum size. For some applications like some twitter data based applications, the small amount of missing data does not cause any problem. But for some others that require all the data existed on the server, the Chukwa based data acquisition solution still need to be improved.

5.2 Time Used for Data Acquisition With Refer to Small Data Size

In addition to the quality of data acquired, time used for data acquisition is another important factor. In this section, time used for data acquisition consists of time

used to acquire data from servers and put acquired data into HDFS.

The time differences for both test programs are shown in Figure 12. This test is based on data sets with small size.

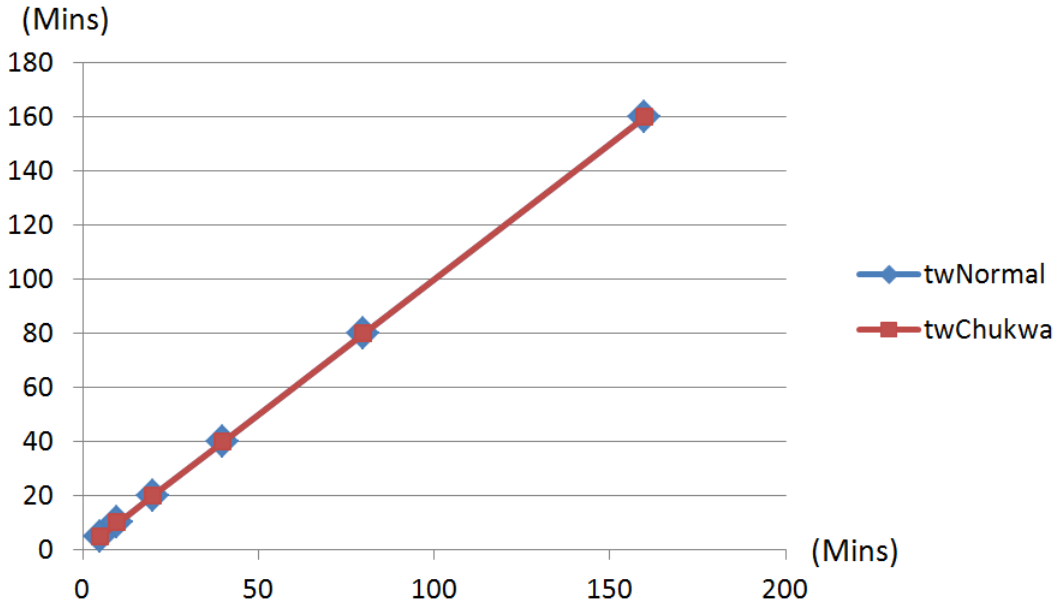

Figure 12: Actual Time Used for Acquisition in a Certain Time

The two test programs are implemented in two different ways. For program *twNormal*, it accumulates the acquired data, and copy the data to HDFS when data size reaches a certain amount or data acquisition finishes. For program *twChukwa*, data file is monitored by Chukwa agents, and acquired data is updated every 2 seconds, which means the data acquired will be put to HDFS every 2 seconds.

As *twChukwa* does not have data copying involved, it saves a lot of time when it comes to big data set. But for small data set, it takes almost no time to copy the data file to HDFS, which makes no time differences for both test programs as shown in Figure 12.

5.3 Performance of Data Copying to HDFS

When the data file is small, there is almost no time differences of data acquisition implemented in the two different ways mentioned in early section, as it takes almost no time to copy the data files to HDFS. But when it comes to big data sets, the copying time can not be ignored. On one hand, it takes time to copy large file to HDFS. On another hand, The replicas mechanism of HDFS makes the actual data file several times larger than the original data, which causes the copying process more time consuming.

When data set is big, the time difference is mainly caused by copying data. The performance of data copying to HDFS is shown in Figure 13.

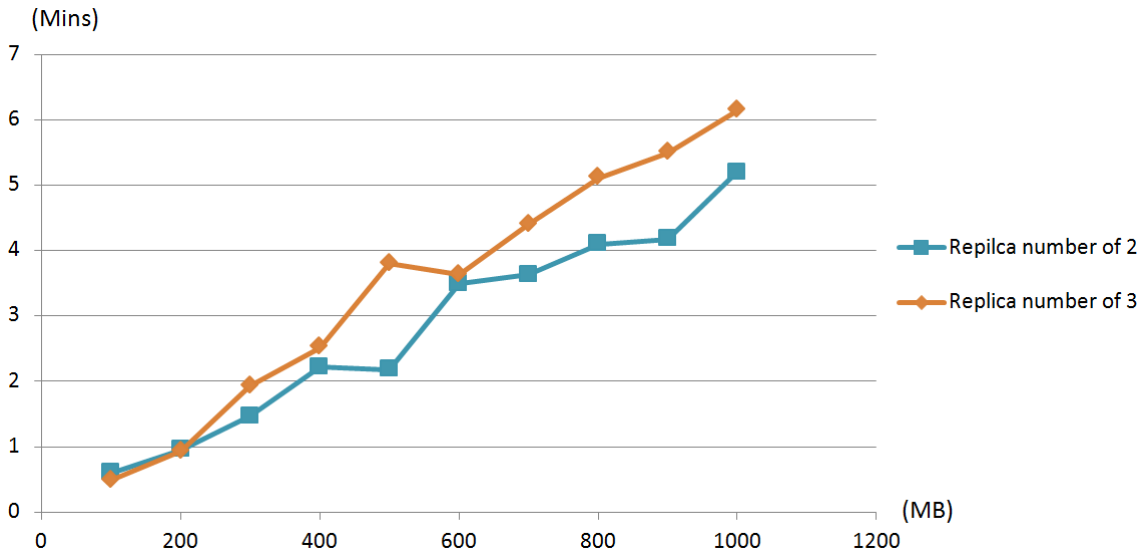

Figure 13: Time Used to Copy Data Set to HDFS With Different Replica Number

As can been seen from this figure, the time of copying increases almost linearly with the size of data. Hadoop is running on 15 hosts of the unix lab at UIS, and the hosts are shared by a lot of users running different jobs. As the computational resources might be occupied by the other users during testing, some points deviate from the line as shown in the figure.

The slope of the line is determined by replica number set in Hadoop configuration

file. Replica number of 2 and 3 is used in the test. As can be seen, the slope of line is bigger when replica number is bigger, and vice versa. The reason for this is that when replica number is bigger than 1, the actual data size several times is bigger than the original data size, which consumes more time to copy.

5.4 Critical Value of Generating Time Differences

There is no time differences of data acquisition when data set is small. As it takes time to copy large file to HDFS, and also dues to the replicas mechanism of Hadoop cluster, there is a big time difference when data set is large. But what is the critical value of data size?

5.4.1 How to Find Critical Value

In order to find critical value, test data is used as it is very time consuming to generate a data file with certain size for test purposes. The only reason why there are time differences for test program is data copying, and we measure the performance of data copying here to find the critical value.

Critical value is a value of data file size. It is the corresponding size of data file for generating time difference of data acquisition. When the time difference is larger than zero, we say there is a time difference, and thus the size of data file that starts to bring time differences is the value we are looking for. But as the hosts that Hadoop is running on might be occupied by jobs running by some other users, we may get different time to copy the same data file twice. So the time differences corresponding to critical value is defined as a time range. We consider the time range to be (0, 10) with unit of second when finding the critical value, and the data file size of starting to generate time difference larger than 10 is the cirtical value we are looking for.

In order to find the critical value, we started with data size of 100M. If the time difference is between 0 and 10, we increase the data file to 200M. If the time

Size of Data set	Time used
20M	2s
30M	3s
40M	3s
50M	8s

Table 1: Time used for copying according to the size of data set with replica number of 2

difference is still in the range (0, 10), we double the data size to 400M. The same principle applies when time difference is larger than 10, and we decrease file to 50M and 25M respectively. If the time difference is larger than 10 when the size is 200M, we will go down to 150M, and if it is smaller than 10 when the size is 50M, we will go up to 75M.

The test in this chapter to find critical value is based on Hadoop cluster with replica number of 2 and 3, and size of data file stored on cluster is doubled or as much as three times as it exists on local host. For different replica number, the actual size of data file is different as there are different number of copies of the same data set on Hadoop cluster, thus the critical value is different.

5.4.2 Critical Value

After did some experiment with copying as shown in Table 1, we found out that time differences appeared obviously when data set reached a size of 50M. When data size is less than 50M and bigger than 40M, it only takes several 3 seconds at most to copy the data. As the computational resources may not be occupied by the testing program only, value in Table 1 is bigger than the actual value. So we came out the conclusion that the critical value for data size of generating time differences is between *40M* and *50M*.

So, for data set smaller than *40M*, there is no time difference between the two solutions proposed in this thesis, and when data set is larger than *50M*, there is

Size of Data set	Time used
10M	2s
15M	2s
20M	8s
30M	10s
40M	21s

Table 2: Time used for copying according to the size of data set with replica number of 3

time difference.

The critical value found here is in the cluster environment where replica number is set to 2, and different replica number for cluster will result in different critical value. Since the actual data size increase with the increasing of replica number, critical value will become larger when replica number is only 1, and much smaller when replica number is bigger than 2. In order to prove this, we also found out the critical value when replica number is 3. Table 2 shows the related data. Through applying the same principle, we found out the the critical value is between *15M* and *20M*.

5.5 Chukwa Based Solution is Faster

Through the performance comparison, we find out that when the data set is small, it cost almost the same time for both Chukwa based solution and the normal solution. When the data set is bigger than the critical value, the stand-alone application takes more time.

To explain this in more detail, both of the solutions spend the same time for data acquisition when data set size is smaller than the critical value. If the size of data set is in the scope of critical value, the two solutions have around 2-10s time differences. When the size of data set is larger than the critical value, it takes

more time for the normal solution to acquire data.

As the size of data set is usually large data set in real life production, we can say that Chukwa based solution is faster.

6 Conclusion

In this thesis, we focused on solutions for data acquisition with a good performance.

There were two different data sources used in this thesis. Data from services provider is in historical format with pulling acquisition API, while data from Twitter server is real-time streaming data.

Two different solutions for data acquisition were introduced and implemented. One acquires data from data sources, and then copies the data file to HDFS. The other one is based on Chukwa. It makes agent check data file every 2 seconds, and saves time that is used for copying in the first one, as well as providing some other benefits such as displaying, monitoring, and analyzing results.

Performances comparison of the two solutions was given out, both on the quality of the data acquired by these two solutions, and the time used for data acquisition. The size of data file which brings time differences was found out at the end of the thesis. Solution based on Chukwa was proved to be better than the other one for data acquisition in Hadoop system.

7 Acknowledgements

Special thanks are given to the following people.

Professor. PhD. Chunming Rong. Chunming Rong is the tutor of mine, and also the person who brought this interesting master topic.

PhD Candidate. M.Sc. Tomasz Wiktor Wlodarczyk. Tomasz Wiktor Wlodarczyk gave me many advices when doing this master thesis, and also helped me with the improvement of this thesis.

References

[1] Jeffrey Dean, Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, Communications of the ACM, Volume 51, Issue 1, pp. 107-113, 2008.

[2] Baeza-Yates, R. and Ramakrishnan, R. *Data Challenges at Yahoo!*. Proceedings of EDBT, March 2008.

[3] *Hadoop homepage*, http://hadoop.apache.org/

[4] Jerome Boulon, Andy Konwinski, Runping Qi *Chukwa: A large-scale monitoring system*, Cloud Computing and its Applications, pp. 1-5, Chicago, IL, 2008.

[5] *Pig homepage*, http://hadoop.apache.org/pig/

[6] *Pig Latin Reference Manual*, http://hadoop.apache.org/pig/docs/r0.5.0/piglatin_reference.html

[7] *OWL homepage*, http://www.w3.org/2004/OWL/

[8] *BioSTORM homepage*, http://biostorm.stanford.edu/doku.php

[9] *Jade homepage*, http://jade.tilab.com/

[10] C.I. Nyulas, M.J. O'Connor, S.W. Tu, D.L. Buckeridge, A. Okhmatovskaia, and M.A. Musen *An Ontology-Driven Framework for Deploying JADE Agent Systems*, Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on, Los Alamitos, CA, USA: IEEE Computer Society, pp. 573-577, 2008.

[11] *Chukwa homepage*, http://hadoop.apache.org/chukwa/

[12] *Network File System*, http://en.wikipedia.org/wiki/Network_File_System_(protocol)

[13] *An example that shows the efficiency of Hadoop*, http://developer.51cto.com/art/201006/203554.htm

[14] Lukasz Golab, M. Tamer Ozsu. *Issues in Data Stream Management*, ACM SIGMOD Record, Volume 32, Issue 2, pp. 5-14, 2003.

[15] *Twitter4j homepage*, http://twitter4j.org

[16] *Example of using twitter4j to get stream data*, http://twitter4j.org/en/code-examples.html#streaming

[17] *log4j homepage*, http://logging.apache.org/index.html