

Bachelor's thesis



**Czech
Technical
University
in Prague**

F3

Faculty of Electrical Engineering
Department of Measurement

Big Data and Its Analysis

Oleg Ostashchuk

Open Informatics - Computer Systems

May 2014

Supervisor: Miloš Kozák



CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Measurement

Technická 2, 166 27 Prague 6, Czech Republic

Academic year **2013/2014**

BACHELOR PROJECT ASSIGNMENT

Student: **Oleg Ostashchuk**

Study programme: **Open Informatics**
Specialisation: **Computer Systems**

Title of Bachelor Project: **Big Data and Its Analysis**

Guidelines:


Study area of Big data analysis, elaborate on the associated problems and suggest their solutions using freely available tools, such as Hadoop, etc. Using these tools, prepare the environment for Big data analysis. Try to analyze given large amounts of data and describe the approach you choose.

Bibliography/Sources:

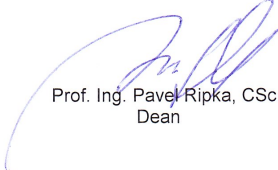
- [1] WHITE, Tom: Hadoop: the definitive guide. 3rd ed. Sebastopol: O'Reilly, 2012, xxiii, 657 s. ISBN 978-1-449-31152-0.
- [2] LAM, Chuck: Hadoop in action. Greenwich: Manning Publications, 2011, xxi, 312 S. ISBN 978-1-935182-19-1.
- [3] JURNEY, Russell: Agile Data Science: Building Data Analytics Applications with Hadoop. O'Reilly Media, 2003. ISBN 978-1-4493-2626-5.
- [4] LIU, Henry H.: Hadoop essentials: a quantitative approach. Charleston, S.C.: PerfMath. ISBN 14-802-1637-2.

Bachelor Project Supervisor: **Ing. Miloš Kozák (K 13132)**

Valid until: **the end of the summer semester of academic year 2014/2015**


Prof. Ing. Vladimír Haasz, CSc.
Head of Department




Prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 27, 2014

Acknowledgement / Declaration

I would like to thank the supervisor of my thesis, Ing. Miloš Kozák for his help, useful tips and for friendly access throughout period of writing my thesis.

I declare, that I have completed this thesis independently and that I have used only the sources listed in the bibliography.

In Prague 23. 05. 2014

.....

Abstrakt / Abstract

Cílem této práce je nastudovat problematiku velkých dat a připravit distribuované prostředí pro jejich zpracování pomocí Hadoop frameworku. Také připravit aplikace pro analýzu dat na bázi MapReduce algoritmu. Všechny úkoly prezentované v moji práci se zastávají dvou hlavních principů, využívat jenom volně dostupné nástroje a hledat takové řešení problému, aby mohlo být použito v reálné aplikaci z praxe.

The main aims of this thesis was to study the area of Big data, to prepare a distributed data processing environment using Hadoop framework, to develop MapReduce algorithms to exploit prepared environment. All work presented in my thesis was based on two main principles, to use just freely available tools and to demonstrate solutions ready to be used in real life applications.

Contents /

1 Introduction to Big Data and Hadoop	1
1.1 Big Data	1
1.2 Principles of data processing	2
1.3 Hadoop framework.....	3
1.4 HDFS.....	5
1.5 MapReduce.....	7
1.6 YARN	9
1.7 Alternative distributions.....	10
1.7.1 Cloudera Distribution for Hadoop	10
1.7.2 Hortonworks Data Platform.....	10
1.7.3 MapR.....	11
2 Installing Hadoop in pseudo distributed mode	12
2.1 Choice of Operating system ...	12
2.2 CentOS	12
2.3 Installation and configuration of CentOS	13
2.3.1 Network configuration ...	13
2.3.2 Disable SELinux and Firewall.....	15
2.3.3 Java installation	15
2.3.4 Establish SSH connection	15
2.4 Choice of Hadoop distribution	16
2.5 Installation and configuration of Apache Hadoop	16
2.5.1 Installing Apache Hadoop	16
2.5.2 Modifying .bashrc.....	16
2.5.3 Configuring Apache Hadoop	17
3 Installing CDH in fully-distributed mode	21
3.1 Hardware Specification	21
3.2 Installing CDH.....	22
4 Data Analysis	27
4.1 Data description	27
4.2 Tasks description	28
4.3 Tasks realization	28
4.3.1 Tasks definition	28
4.3.2 Downloading data	28
4.3.3 Preparing data	28
4.3.4 Loading data to HDFS ..	29
4.3.5 Performing MapReduce job	29
4.3.6 Observing results	30
4.3.7 Pearson's chi-squared test of calculated results .	30
4.3.8 Minimal and maximal daily temperatures.....	32
5 Conclusion	33
References	35
A Source code: MapReduce java program	37
A.1 TempMapper class	37
A.2 TempReducer class	39
A.3 TempJob class	40
B Source code: PrepareFiles java program	41
B.1 PrepareFiles class	41
B.2 FilesMerger class.....	43
C CD contents	45

Chapter 1

Introduction to Big Data and Hadoop

Humans are creating records for many years. Naturally, important of them they have been trying to store, the reasons why were they doing it are different, certainly just for case that the records will be needed sometimes later. People have been creating archives, where all these records were stored, we can imagine big rooms with high shelves filled by many folders and records, sometimes not just the rooms but even whole buildings are used for archives. Searching information in that kind of archives can be not so easy and sometimes requires more time to be spent on it. Invention of computer in 20th century significantly influenced the data storing. It enabled to store records in absolutely different way, in digital form. Huge volumes of space previously needed are not required now. Since the computers started to be used for storing data, it enabled to to search needed records significantly faster than before, and to decrease amount of resources needed for storing and seeking information.

Many organizations and institutions started to use computers. Certainly their work has been producing a lot of records. For many years computers with ease managed with all input information to be processed and stored. The problems appeared over the past few years, when we were able to observe rapid progression of data, produced by people in digital form. Processing these huge data sets, seemed to be not an easy task, and requires enough resources.

1.1 Big Data

In last years organizations are generating huge amounts of information about their customers. Usually to be successful in their branch of activity, there is often a necessity of storing, as much useful information as is possible, for further analysis.

- Financial institutions store much more information about their clients, than usually people expect. They observe records about various clients' accounts, such as income, payments, periodically balance state and many other. All these records are processed to create a vision of client's expected behavior. It seemed to be very useful information for financial institutions like banks and insurance companies. After they are able to obtain client's expected behavior, they can decrease risk in making decisions, about providing or not providing different kind of services (such as loans) to the client.
- Search engines is good example, where enormous amount of data is need to be processed. At first search engines need to store a lot of information about

web pages, its content, keywords, whether the web page is popular or not. Then users, they involve another huge amount of information to be recorded. Example, often entered words to be searched, then web pages, in which user was interested and visited them. These records are being stored to ensure that people could easily search the web page with the content they really need.

- Institutions in sphere of telecommunication, such as mobile operators, store almost any information about client's actions. All records about calls, their duration, the numbers that are called, amount of traffic used for internet services, and many other services, it is all recorded and used for later analysis. What allows to optimize services, so they maximally satisfied the client's demand.
- Social networks are nowadays getting more and more popular. People upload their photos and videos, update their statuses, message their friends. It is an example of case where the problem is not related to the processing of huge data sets of log files, but to the storage resources.

Naturally the general aim, why companies in all these sectors, are storing and analyzing data is to make more profit. Many of these greatest organizations, which obtain a lot of records about people and their activities sometimes even cooperate with security agencies, which collect and analyze data about people for safety reasons.

And now the term, Big Data, it represents all those previously mentioned large data sets, which seems become a key basis of competition, underpinning new waves of productivity growth and innovation.

1.2 Principles of data processing

Processing of large data sets has always been limited to the processing power that can be built into a single computer. The rapid progression of generated data to be stored and analyzed, observed during last few years, caused that existing tools for it, were becoming inadequate to process such large data sets. Companies like Google, Yahoo and Amazon were first which were able to feel insufficiency of existing data storage systems. Even fast and expensive computers with high performance were unable to manage with a given problems. The problem is easy to explain. However the storage capacities of hard drives have increased massively over the years, access speeds to the medium wasn't improved so much. So there was a necessity of system with a new approach for storing data. Today there are two approaches to scaling a system, generally referred to as scale-up and scale-out. [1]

Classic data processing systems have typically been working on large computers with usually large price tags. As the size of the data grows, and the data processing reaches computer's performance limits, the solution is in moving to

a bigger and more powerful computers. This approach of scaling a system is known as scale-up. Today given approach is still very useful and widely used for applications such as commercial database engines, where the software handles the complexities of utilizing the currently available hardware. The problem appears in practical limits on just how big a single computer can be, so at some point, scale-up cannot be extended any further.

Instead of growing a system onto larger and larger hardware, the scale-out approach spreads the processing onto more and more machines. An alternative solution is based on idea of distributed data storage. If the data set doubles, simply use two servers instead of a single double-sized one. If it doubles again, then just move to four hosts. The obvious benefit of this approach is that purchase costs remain much lower than for scale-up. Server hardware costs tend to increase sharply when you are looking for larger machines, for example ten times increased processing power may cost a hundred times as much, but buying ten slower machines will affect the price just lineary. The downside of the scale-out approach, as the scale-out systems get larger, the difficulties caused by the complexity of the parallelism in the systems have become significant. One of the problem is related to count of used machines, as we start to use multiple computers, the chance of failure one of them is increased, what makes higher risk of data loss. This can be solved by data replication, stored data are replicated on different hosts, what excludes probability of data loss. Data processing sometimes requires the data stored on different hosts to be processed together at once, what demands often complex processes like shifting and combining the data partitions. In general, effectively utilizing multiple computers seemed to be a difficult task, and implementing the necessary mechaisms for it can entail enormous effort.

1.3 Hadoop framework

The Apache Hadoop software library is free open source 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. Mostly, Hadoop term is known for programing model MapReduce and for its distributed filesystem – HDFS, but in fact the name is used for several Apache projects related to distributed computing and data processing.

Previously was mentioned, that distributed data processing leads to several complexities. Hadoop manages with all those mentioned problems of distributed data storage systems. That's why it becomes to be popular and recently often used. For people who never heard about Hadoop, it can be described with three main properties which makes it so useful.

- **Concurrency** — everything works in parallel. Data are splitted into partitions and stored in distributed file system. Later computing processes over the data are being executed simultaneously on different machines.
- **Reliability** — Hadoop is a reliable system, that takes care of data replication (with a suitable replication count) and in case of some failure it ensures that data are not lost.
- **Scalability** — whenever there is a need to handle larger data, Hadoop scales linearly, just by adding another nodes to the cluster.

In general Hadoop is the thing, that allows users to use complex distributed system simply, as locally working system. Software developers don't have to care about whole complexity of distributed system and can easily focus on the aims of their work.

As the distributed computing is getting popular, there are appearing different projects, not necessarily under Apache, providing complementary services to Hadoop. Currently Hadoop base consists of four sub-projects, providing the base functionality of the system.

- **Hadoop Common** — A middleware, set of the common utilities and libraries that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS)** – Distributed file system, designed for storing very large data sets by scaling out across a cluster of hosts. Files are splitted into so called block of given size. It achieves high availability through replication.
- **MapReduce** — Programing framework used for performing distributed computing processes. Data are processed in two main steps mapping and reducing.
- **YARN** — A new MapReduce runtime designed for better job scheduling and cluster resource management.

Except for four main Hadoop components, enabling base functionality of distributed system, there are plenty of other complementary projects, supplementing additional facilities.

Hive — The Apache Hive data warehouse software facilitates querying and managing large datasets stored in distributed file systems. Hive provides a mechanisms to query the data using a SQL-like language called HiveQL. HiveQL queries are built to work in MapReduce model, at the same time language also allows traditional map/reduce programmers to add their custom map and reduce

functions when it is inconvenient or inefficient to express this logic in HiveQL.

Pig — Apache Pig runs on Hadoop and makes use of MapReduce and the Hadoop Distributed File System (HDFS). The language for the platform is called Pig Latin, which abstracts from the Java MapReduce idiom into a form similar to SQL. Pig Latin is a data flow language that describes how data will be transformed. Similarly as Hive, Pig also allows users to extend Pig Latin with own user defined functions, which can be written in Java or Python.

HBase — Hbase is a column-oriented distributed database system, built on top of HDFS. HBase component is being used when a real-time read/write access to large datasets is required.

Sqoop — A tool for transferring data between relational databases and Hadoop.

To get something running on Hadoop, it obviously need to be configured. One useful property of Hadoop is, that it lets the user decide in which mode should Hadoop run.

- **Local standalone mode** — This is the default mode if anything else is not configured. In this mode, all the components of Hadoop, such as NameNode, DataNode, JobTracker, and TaskTracker, run in a single Java process.
- **Pseudo-distributed mode** — In this mode, a separate JVM is spawned for each of the Hadoop components and they communicate across network sockets, effectively giving a fully functioning minicluster on a single host. This is very useful mode for developing applications, as the everything works like on the real cluster, just with one node. It will be not possible to test efficiency of distributed system, but testing application on smaller amounts of data, saves the time.
- **Fully distributed mode** — In this mode, Hadoop is spread across multiple machines where each node has its assigned function.

Fully distributed mode is obviously the only one that can scale Hadoop across a cluster of machines, but it requires more configuration work. While developing the software, generally the pseudo-distributed mode is preferred, even when running applications on a single host in the pseudo-distributed mode, everything is almost identical to how it works on a real cluster.

1.4 HDFS

Hadoop Distributed File System, or just HDFS – it is main storage system, used by Hadoop components. It was designed for storing huge amounts of information

(terabyte even petabyte), simultaneously providing high rate of access speed. Files stored in HDFS are splitted onto blocks, replicated and distributed over the nodes of cluster, this provides high rate of reliability and computing speed. The system is designed in the way that, files in HDFS are written only once, and editing of already written files is not possible. As the HDFS is block-structured file system, files during their loading have to be physically splitted onto the blocks of fixed size. Size of the blocks is set in cluster's configuration file (default size is 64 MB). Each file block is then replicated with the preset replication quotient (default value is 3). Replicated blocks are stored on physically separated machines, there are used suitable mechanisms to spread copied blocks over cluster nodes, what ensures, that in case of some failure, needed data are not lost and could be read from another node. As the HDFS was designed for storing big files, it is normal that files stored in HDFS could easily exceed storage capacity of single machine's hard-drive, nothing guarantees that blocks of one file will be stored on the same machine, blocks are spread over the cluster automatically, using the suitable hash function.

HDFS provides all mentioned mechanisms by running set of daemons. These daemons could be understood as an individual programs, each performing its specific role. Some daemons are running on one server, and another are running across multiple machines. Currently HDFS includes these daemons:

- **NameNode** — Hadoop implements a master/slave architecture for both, distributed storage and distributed computation. It performs master role of HDFS. Namenode manages general information about structure of the file system, like file system tree, metadata for all files and directories in the tree. Metadata of file doesn't require much of storage, it contains only information about file name, permissions and where file blocks are physically stored, in addition, to ensure that structure of metadata wasn't modified by multiple simultaneously connected clients, all metadata are stored in the main memory of single machine running the namenode daemon. The general functionality of namenode can be described as managing of memory and assigning the simple I/O tasks to slave nodes. To decrease the workload of machine, server hosting the namenode typically doesn't store any user data or perform computing operations.
- **DataNode** — Datanodes daemons are usually running on the slave machines of the cluster, performing simple read and write tasks of HDFS files' blocks to the concrete place of memory on the local file system. Namenode holds an information where each file block is stored, and when read or write task is performed, namenode provides this information to the client, which further communicates directly with the datanode. Datanodes are constantly reporting to the namenode, each datanode informs namenode about file blocks it is currently storing.

- **Secondary NameNode** — Each cluster has many datanodes and just one namenode, therefore namenode is a single point of failure. Any hardware or software failure of the machine, running the namenode daemon, can cause whole storage system crash. For this cases was invented secondary namenode, it's an assistant daemon form monitoring the state of cluster. During functionality of cluster, all metadata about files and file system tree are managed by namenode, secondary name node just periodically takes snapshots of this information (intervals can be modified). In case of namenode's failure, secondary name node doesn't substitute the namnode's function, it just helps to minimize the data loss and time of recovery process.

HDFS file blocks are not part of the local file system, usually all nodes of the cluster are locally running on linux OS, but any operations like copying, moving of the HDFS file blocks within local operating system are not possible. Instead of it, HDFS contain its own utility that supports file managing commands, syntax is used similar to linux OS's commands. [1–3]

To open the file, client firstly communicates with namenode, it receipts list of file's blocks, from which file consists and where are they stored. Further the client communicates directly with the datanode, and namenode doesn't take part in any information exchanging.

1.5 MapReduce

MapReduce term refers to two distinct things: the programming model and the specific implementation of the framework in Hadoop. Programing model is designed to simplify the development of large-scale, distributed, fault-tolerant data processing applications. MapReduce is primarily a way of writing applications. In MapReduce, developers write jobs that usually consist of two steps, a map function and a reduce function, and the framework handles all complexities of parallelizing the work, scheduling tasks of the job on slave machines, recovering from failures, and others. Developers don't have to implement complex and repetitious code and instead, can easily focus on algorithms and logic of their work.

Hadoop MapReduce is a specific implementation of the MapReduce programming model, and the computation component of the Apache Hadoop project. The combination of HDFS and MapReduce is what makes Hadoop so powerful.

In mapreduce programming model, users are responsible to write a client application that submits one or more mapreduce jobs that consists of user-defined map and reduce functions and a job configuration. The framework handles breaking the job into tasks, scheduling tasks to run on machines, monitoring each task's health, and performing any necessary retries of failed tasks. Commonly, the

input and output data sets are one or more files stored on a distributed filesystem.

A mapreduce job is made up of four distinct stages, executed in order: client job, map task execution, shuffle and sort, and reduce task execution. Client applications can really be any type of application the developer desires, from commandline tools to services. The job itself is made up of code written by a developer using the MapReduce APIs and the configuration which specifies things such as the input and output data sets.

Similarly as the HDFS, Hadoop's MapReduce component is working in master/slave architecture. Whole logic is implemented in running one master daemon that is responsible for resource managing and task scheduling, and running many slave daemons on working machines, where each tasks are running.

In Hadoop's MapReduce component, there are two job related daemons Jobtracker (master service) and Tasktracker (worker service):

- **Jobtracker** — Similarly as the HDFS's daemon – namenode, the jobtracker is the master process, responsible for accepting job submissions from clients, scheduling tasks to run them on working machines, and providing administrative functions such as working node's health and task progress. Similarly as the namenode, there is only one jobtracker per MapReduce cluster, what makes it a single point of failure, it usually runs on reliable hardware since a failure of the master will result in the failure of all running jobs.

Just like the relationship between datanodes and the namenode in HDFS, the similar communication is being performed between MapReduce's daemons, tasktrackers inform the jobtracker about their current health and status by way of regular heartbeats. Each heartbeat contains the total number of map and reduce task slots available, and detailed information about any currently executing tasks. After a configurable period of no report from tasktracker side, a tasktracker is assumed dead.

When a job is submitted, information about each running task that makes up the job is stored in jobtracker's node memory. This task information updates with each tasktracker heartbeat while the tasks are running, providing a near real-time view of task progress and health. After the job completes, this information is retained for a configurable time period or until a specified number of jobs is executed. On an active cluster where many jobs, each with many tasks, are running, this information can consume a considerable amount of RAM memory.

The process of deciding which tasks of a job should be executed on which worker nodes is referred to as task scheduling. Similarly like CPU is being shared for computing on one machine, tasks in a MapReduce cluster share

working nodes, the only difference, that there is no switching between tasks running — when a task executes, it executes completely.

- **Tasktracker** — The second daemon, obviously similar to HDFS’s datanode, is a daemon with slave role of architecture. Tasktracker receives task assignments from the jobtracker, executes given tasks, locally over the data portion stored exactly on the same machine, and periodically reports the progress back to jobtracker. There is always running only one tasktracker on each working node. As it could be predicted from the architecture principles, both tasktrackers and datanodes daemons are running on the same working node’s machines, what makes each node both, a compute node and a storage node.

Each tasktracker has a configurable number of map and reduce task slots that indicate how many tasks of each type it is capable of executing in parallel. A task slot is exactly what it sounds like, it is an allocation of available resources on a worker node to which a task may be assigned. A tasktracker executes some number of map and reduce tasks in parallel. Map and reduce slots are configured separately, as they consume resources differently. For efficient work of the program, it is very important to pick up the correct number of map and reduce task slots, to reach the full use of the working node’s machine.

[4, 2–3]

1.6 YARN

Architecture of distributed data processing described before, seemed to work very effectively until the some limitations were reached. With the rising number of nodes in Hadoop cluster, was seen that resource requirements on a single running jobtracker were just too high. In new releases of Hadoop, traditional MapReduce component has been modified. New component is named YARN (“Yet Another Resource Negotiator “), sometimes also referred as MapReduce 2. The main differences are related to principles of jobtracker work. In new releases, rather than have a single daemon, that assigns resources such as CPU, memory and manages MapReduce jobs, these functions were separated into two daemons.

The resource management aspect of the jobtracker is now being run as a new daemon called the resource manager. A separate daemon responsible for creating and allocating resources to multiple Map-Reduce jobs. Each Map-Reduce job is now assigned as an individual application, and in difference to have a single jobtracker per whole cluster, each job now has its own jobtracker equivalent called an application master that runs on one of the working nodes of the cluster.

This is significant difference, in comparing to the previously centralized jobtracker. In new concept application master of one job is now completely isolated from any other. This means that if some unexpected failure will occur within the jobtracker, other jobs are not affected. Further, because the jobtracker is now

dedicated to a specific job, multiple jobtrackers can be running on the cluster at once. When an application completes, its application master, such as the jobtracker, and other resources are returned back to the cluster. As a result, there's no central jobtracker daemon in YARN.

Worker nodes in YARN have also undergone some changes. Now the tasktracker runs as a new daemon called the node manager. While the tasktracker expressly handled MapReduce specific functionality such as launching and managing tasks, the node manager is more generic. Node manager launches any type of process, dictated by the application, in an application container. [5]

1.7 Alternative distributions

Apache Hadoop is an open source project, what allows other enterprises to use and extend the code base with additional features. Beside the new components that are being developed by the companies, configuring of already known open source Apache components, sometimes requires much effort and wide knowledge, to ensure that whole complex system will be safe and all components will cooperate properly. These are general branches of services provided by companies in the mentioned sphere.

1.7.1 Cloudera Distribution for Hadoop

Cloudera Distribution for Hadoop is probably the most known widely used Hadoop distribution, that is often referred as CDH. Cloudera is the company where Doug Cutting, the creator of Hadoop, is currently working. Useful Hadoop component, Sqoop was also created by Cloudera and contributed back to the open source version. The Cloudera distribution is available at its official webpage, containing a large number of Apache products pre-configured to work properly together, including Hadoop itself, then previously mentioned components as Hive, Pig, Hbase and tools such as Sqoop and other lesser-known products. Cloudera Distribution is available in several package formats, some of which are available to be used for free, and presents the software in a ready-to-go form. The base Hadoop product, for example, is separated into different packages for the components such as NameNode, TaskTracker, and so on, and for each, there is integration with the standard Linux service infrastructure. Beside the base Hadoop, CDH was the first widely available alternative distribution, and its availability for free cost has made it as very popular choice. Cloudera also provides additional commercial-only products, such as different management tools, in addition to training, support, and consultancy services. [6]

1.7.2 Hortonworks Data Platform

Hortonworks is the Yahoo derived company, created as a division responsible for the development of Hadoop. In difference to Cloudera, Hortonworks presents all its modifications 100% open source including Apache Hadoop, without any non-open components. Hortonworks offers a very good, easy-to-use sandbox for getting

started. Hortonworks developed and committed enhancements into the core of Hadoop that make possible to run it natively on the Microsoft Windows platforms including Windows Server and Windows Azure. They have also produced own pre-integrated Hadoop distribution, called the Hortonworks Data Platform (HDP) and available to be downloaded from the official webpage. Hortonworks Data Platform is in fact very similar to its Cloudera's opponent, and the general differences can be watched in their focus. Hortonworks follows the fact that HDP is fully open source, including the management tool, what is different for Cloudera Distribution, which provides management tool as part of commercial release. Hortonworks does not offer any commercial software, its business model focuses instead on offering professional services and support for the platform.

■ 1.7.3 MapR

MapR is a little bit different type of distribution offered by MapR Technologies, the company and its distributions are usually referred simply as MapR. Their distribution, as usually, is based on Hadoop but it has added a number of changes and enhancements. MapR successfully focused it's activity on performance and availability, it was the first distribution to offer a high-availability solution for the Hadoop NameNode and JobTracker, what is a significant weakness of Hadoop core, known as single points of failure. MapR replaced HDFS with a full POSIX-compliant filesystem that can easily be mounted remotely. MapR provides both a community and enterprise edition of its distribution, not all the extensions are available in the free product. MapR similarly as the previous two companies also provides support services as part of the enterprise product subscription, in addition to training and consultancy.

Chapter 2

Installing Hadoop in pseudo distributed mode

In this chapter I'm going to describe installation and configuration of Apache Hadoop framework on one computer, in pseudo-distributed mode. This is usually the first and most typical procedure for persons that are introducing with hadoop. Pseudo-distributed mode doesn't reflect any advantage of the distributed data processing but it allows to feel importance of any configurations required to be done on the real cluster. It is a good option for software developing and testing it on small amounts of data.

Following sections of this chapter describe steps, that are going to be realized on one computer. In the case of running cluster in fully-distributed mode, each of its host machines requires the same configurations like it is needed to be done on one computer running pseudo-distributed mode of hadoop. Therefore following chapter describes base configurations that will be later used when creating real fully-distributed cluster.

2.1 Choice of Operating system

Decision about choosing the operating system of host machines is the first step, that has to be done. Beside some presented ideas and solutions of Hortonworks company about using Microsoft Windows platform, I better preferred Linux distribution, as it is generally recommended by Apache Hadoop. My decision about concrete Linux distribution was mainly influenced by many recommendations at the internet, and by general fact, that all last stable distributions of Cloudera and Hortonworks companies were presented on host machines, running on CentOS. Therefore for my further steps I have chosen using of CentOS.

2.2 CentOS

Community Enterprise Operating System, or mostly known just for CentOS term, is an enterprise Linux distribution. It was developed by the CentOS Project community using the source code of the commercial Linux distribution, the Redhat Enterprise Linux (RHEL). CentOS was created to be a freeware alternative to RHEL and to have a Linux distribution that is as stable as its commercial counterpart and can keep up with the requirements of the enterprise. Besides the popularity and widely useability of commercial RHEL, its freeware alternative doesn't fall behind in its popularity. As the CentOS was built using

the source code of RHEL, it is binary compatible with the RHEL.

Another significant advantage of CentOS operating system is related to its developers' principles, that CentOS will follow the redistribution rules of RHEL making it truly free alternative to the original one. CentOS is continuously being developed by its core developers and community, providing all security and software updates, what ensures the stability of the distribution.

In addition, Red Hat in year 2012 has announced that it intends to maintain the life cycle of its Red Hat Enterprise Linux distribution of versions 5 and 6 for ten years, instead of the previously planned seven. Actually, the customers with the additional three year Extended Lifecycle Support will also receive additional support beyond the standard ten year period, meaning that RHEL users can receive up to 13 years' support. Following this extension, RHEL 5 should come to the end of its life cycle in March 2017 and RHEL 6 in November 2020, with Extended Lifecycle Support coming to end three years later. Because of the effort of CentOS's developers community and Red Hat Enterprise Linux extended Life Cycle, CentOS is able to have a constant release upgrades scheduled to new releases of RHEL expectedly for the same 10 year's period. Thanks to roots of RHEL, CentOS generally seems to be a good choice to be used in many projects, regarding to its compatibility, quality, and support properties.

2.3 Installation and configuration of CentOS

Generally it is recommended to use the same operating system for all nodes of the cluster, in my case, my cluster consists just from one computer running on CentOS 6.4 version with minimal installation. Installation of operating system itself is an easy process and doesn't require much effort, in case of CentOS, it takes approximately 10 minutes. Image disks with the source files of operating system can be downloaded from any of mirrors sites on the official website www.centos.org. Similarly any other information and more detailed installation's guide can be obtained on the same website. After the installation is completed, there have to be done several configurations, therefore for further steps we need to be logged as root user. Minimal version installation contains minimum of preset settings or preinstalled programs, what ensures that operating system will contain just the things we need, and nothing else will consume computer's resources.

2.3.1 Network configuration

Network configuration is first significant task to be done. I consider that all computers of the cluster are already physically connected to one network. Even if the cluster consists just from the one computer, it still requires network to be configured. It is important that all computers of the cluster have assigned static IP address, that all nodes were able to communicate with required destination node anytime. Other required settings mostly depend on the concrete network configurations, to which computers are connected and all useful parameters can

be obtained from network's administrator.

In CentOS operating system most of required settings are configured by interfaces' configuration files in the following directory.

```
/etc/sysconfig/network-scripts/
```

In my case, computer to the network is connected by the ethernet cable, therefore I need to edit ethernet interface configuration file:

```
/etc/sysconfig/network-scripts/ifcfg-eth0
```

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=192.168.227.101
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
DNS1=8.8.8.8
DNS2=8.8.4.4
```

First line identifies the concrete used interface. If there are more ethernet interfaces on the used computer, than it is important choose correct one. Bootproto parameter corresponds to way how the whole configuration will run and how the IP address will be assigned to the computer. There are two options, the first one, let the dhcp server assign IP address and network configuration, then bootproto parameter requires to be set as `dhcp`. The second option is to assign static IP address by yourself, this option corresponds to `static` bootproto parameter and requires that all further network configurations to be done by the user. About the onboot parameter there is no need for long description, it just enables to start interface after operating system boot. Then following lines have to be set in the case that bootproto option is `static`, they include parameters like IP address, subnet mask, default gateway, domain name server address, all this information may be obtained from network administrator. In dependency on operating system version it is possible that given parameters are needed to be set also in other configuration files, but in case of CentOS 6.4 all information of interfaces' configuration files is extracted automatically after network service restart command.

Then another network configuration file has to be edited:

```
/etc/sysconfig/network
```

```
NETWORKING=yes
HOSTNAME=node1.hadoop
```

Content of the given file just enables network and declares the hostname of current computer. In relation to this, hostnames of all computers used in the cluster have to be declared in hosts file on each computer. It is important thing, as the future steps of installation will require that it was possible to access all computers of the cluster with the hostnames they have assigned.

```
/etc/hosts

127.0.0.1      localhost
192.168.227.101  node1
...
```

At this moment that's all we need to configure about the network, and to let operating system use configured settings there is a need to restart network service, it can be done by running following command:

```
# service network restart
```

■ 2.3.2 Disable SELinux and Firewall

According to the Cloudera's installation guide it is recommended to maximally isolate cluster's computers from the outside world, and to disable SELinux and firewall on each computer. This can be done by editing following configuration file

```
/etc/selinux/config

...
SELINUX=disabled
...
```

and running following commands:

```
# system-config-firewall-tui
# reboot
```

■ 2.3.3 Java installation

Hadoop is written in Java, therefore it needs Java to be installed on each host of the cluster. According to the official documentation it is possible to use 6th or 7th Java versions. In my case I'm going use OpenJDK 7 version. In CentOS it can be installed by running following commands:

```
# yum install java-1.7.0-openjdk
```

■ 2.3.4 Establish SSH connection

Before the Hadoop installation can be run, all computers of the cluster have to be reachable through SSH connection, therefore all computers need to have the same root's password or added public ssh keys as authorized keys. Even if cluster consists just from one node, each Hadoop's daemons like namenode, datanode, jobtracker, tasktracker, all communicate through the ssh port. Following commands will ensure SSH connection in CentOS.

This command installs openssh server and client.

```
# yum -y install openssh-server openssh-clients
```

These commands generates new ssh key and copy public key to authorized keys.

```
# ssh-keygen -t rsa -P ""
# cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

At this point that's all about operating system's configuration.

2.4 Choice of Hadoop distribution

Another point for decisions is a choice of Hadoop distribution. As I have previously mentioned, today effectively working Hadoop cluster usually represents a complex system, built of a high number of different components. Installation and configuration of that kind system is not an easy task, therefore some companies provides commercial either freeware distributions of Hadoop including installation, consultation and maintenance services. Usually in practice it is much more common that distributions from Cloudera or Hortonworks are used on real Hadoop clusters. But on the other hand, to demonstrate clear work of Hadoop framework without any built-in components it is good to install freeware Apache Hadoop distribution, therefore in this chapter I'm going to describe its installation and configuration to work in pseudo-distributed mode.

At the moment of writing my thesis there was already published Apache Hadoop 2.2.0 as stable release. This version already contain significant improvements in HDFS and MapReduce, over previous stable releases (hadoop-1.x). Improvements are briefly described in previous chapter, but more detailed information and list of all improvements can be found in official documentation. New Hadoop releases of version 2 are backward compatible, therefore all programs working with hadoop-1.x can be easily transferred to work on cluster running Hadoop version 2.

2.5 Installation and configuration of Apache Hadoop

2.5.1 Installing Apache Hadoop

At first we need to download required framework files from one of the official mirror sites. It is also considered that all following commands are run by root user.

```
# wget mirror.hosting90.cz/apache/hadoop/common/hadoop-2.2.0/hadoop-2.2.0.tar.gz
# mv /downloads/hadoop-2.2.0.tar.gz /usr/local/
# cd /usr/local/
# tar xzf hadoop-2.2.0.tar.gz
# mv hadoop-2.2.0 hadoop
```

After download is completed, archive files are extracted to directory:

```
/usr/local/
```

2.5.2 Modifying .bashrc

For more comfortable further work it is good add new permanent variables in .bashrc file.

```
$HOME/.bashrc
#Hadoop variables
```



```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.45.x86_64
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
```

By fulfilling this step we are done with all prerequisite actions. Now it is time to configure Hadoop framework with settings that we need. All further actions will be realized in directory:

```
/usr/local/hadoop
```

■ 2.5.3 Configuring Apache Hadoop

Firstly we need to open and edit `hadoop-env.sh` file, by entering `JAVA_HOME` variable value.

```
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
...
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.45.x86_64
...
```

In configuration file `etc/hadoop/slaves` it is required to enter hostnames of nodes that are going to be used in cluster. In the case of pseudo-distributed mode there is entered just hostname of one node. If some other nodes are planned to be added to the cluster, all their hostnames have to be entered to this file.

Main system settings are set in `core-site.xml`

```
/usr/local/hadoop/etc/hadoop/core-site.xml
...
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://node1:9000</value>
  </property>
</configuration>
```

All HDFS setting have to be configured in `hdfs-site.xml`

```
/usr/local/hadoop/etc/hadoop/hdfs-site.xml
...
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
```

```

    </property>
    <property>
      <name>dfs.namenode.name.dir</name>
      <value>
        file:/usr/local/hadoop/tmp/hdfs/namenode
      </value>
    </property>
    <property>
      <name>dfs.datanode.data.dir</name>
      <value>
        file:/usr/local/hadoop/tmp/hdfs/datanode
      </value>
    </property>
  </configuration>
  ...

```

For example here in this file, value of parameter `dfs.replication` means number of replicas of each block in HDFS. By default this values equals 3, but if we want to run Hadoop in pseudo-distributed mode, we will have just one node in the cluster, this value can not be greater than number of nodes in cluster, therefore this value is set to 1.

Parameters `dfs.namenode.name.dir` and `dfs.datanode.data.dir` tells route to the directories where the data and information of HDFS will be physically located. If the given folders don't exist, they have to be created manually, before using them.

We are going to use YARN, following file has to be edited.

```

/usr/local/hadoop/etc/hadoop/mapred-site.xml
...
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
...

```

All further setting related to YARN have to be configured in `yarn-site.xml`

```

/usr/local/hadoop/etc/hadoop/yarn-site.xml
...
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce
      .shuffle.class</name>

```

```

        <value>org.apache.hadoop.mapred
.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler
        .address</name>
        <value>node1:8030</value>
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>node1:8032</value>
    </property>
    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>node1:8088</value>
    </property>
    <property>
        <name>yarn.resourcemanager.resource-tracker
        .address</name>
        <value>node1:8031</value>
    </property>
    <property>
        <name>yarn.resourcemanager.admin.address</name>
        <value>node1:8033</value>
    </property>
</configuration>
...

```

Settings of resourcemanager are required to enable, that all nodes of cluster were able to be seen by resource manager panel. Command for formatting HDFS has to be run:

```
# bin/hdfs namenode \{format
```

And now finally Hadoop services can be run:

```
# sbin/start-dfs.sh
# sbin/start-yarn.sh
```

In previous versions of Hadoop, there was used script `sbin/start-all.sh`, for starting all Hadoop services, but in new version of Hadoop since 2.*.* this script was declared as deprecated.

At the end we can check which java processes are running and we should obtain output like this:

```
# jps
4868 SecondaryNameNode
5243 NodeManager
5035 ResourceManager
4409 NameNode
4622 DataNode
5517 Jps
```


Chapter 3

Installing CDH in fully-distributed mode

In previous chapter was described configuration of operating system for single computer cluster. In fact, that is a base configuration for each node of the cluster. After providing given settings on each host machine, the system together is ready for Hadoop framework to be launched on it.

Running Hadoop on a single node in pseudo-distributed mode is good for learning new system and software developing reasons, but the main aim of this part of my thesis, was to keep moving forward in usability point of view. Therefore in the given chapter I'm going to describe efforts required to be made, when creating real fully-distributed cluster for data processing. Hadoop was designed to run and coordinate work on big clusters consisting of hundreds nodes. During executing my academical project I wasn't able to use so big amount of computers, but in fact it doesn't change the situation very much, as the work of cluster preparation is in principles the same.

3.1 Hardware Specification

It is hard to predict hardware requirements for Hadoop cluster, as the situation rapidly changes with time. It mainly depends on what will be the main aim of work of the given cluster. Hadoop is designed to run on so called **cheap** hardware. It doesn't mean that it should use cheap components. In general cheap components with higher failure probability don't seem to be an economical choice, when creating cluster of hundreds machines. So called **cheap** means that hardware components used in Hadoop's cluster don't have to be the newest and the most powerful, presented on the markets. High efficiency of the cluster is reached by alternative ways. This was already described in chapter 1.

For my academical purposes was absolutely enough to design cluster of 5 nodes. All 5 nodes are independently running in one common virtualized environment, connected by a virtual switcher enabling 1Gbps speed for ethernet technology. Cluster contain one master node running main master's daemons: NameNode and JobTracker. Master node also runs few additional services, mainly Cloudera's monitoring related services. Then there are four slave nodes, used for data computation. They all are running just DataNode and TaskTracker daemons. All CPU, RAM and storage capacity properties used for each node are written in the table below.

node	CPU cores	RAM	Storage cap.
Master node	4	4 GB	20 GB
Slave nodes	1	2 GB	30 GB

Table 3.1. Hardware parameters

3.2 Installing CDH

For today Cloudera's distribution for Hadoop seems to be the most perspective choice for running multi-node cluster, providing reliable base Hadoop system with additional components, including administration and monitoring services. Therefore for my further steps, I have chosen Cloudera's freeware distribution. Decision of used distribution is related to the aim of my project, to represent real usable system that correspond to distributed data processing systems used in practice. At the moment of installation, CDH 4 was declared as stable version and recommended by official website.

After the all required operating system's settings from previous chapter are successfully done on each node, cluster is ready for CDH installation. There are several ways how CDH can be installed. Official installation guide recommends to use Cloudera Manager (CM) for CDH installation. It facilitates CDH installation and enables to perform whole installation via web browser. CM manager has to be installed just on one node, user don't have to download and configure required CDH files on each node independently, this all is done by CM itself.

Cloudera Manager installation can be launched by entering following commands:

```
# wget archive.cloudera.com/cm4/installer/latest/cloudera-manager-
installer.bin
# chmod +x cloudera-manager-installer.bin
# ./cloudera-manager-installer.bin
```

Installation of CM runs in blue console window. CM additionally installs Java, if the user hasn't already done it before. In my case, I have performed CM installation on master node and all further installation step were performed from this node.

After installation of CM is complete, there appears a window with the message, that informs how to proceed with the CDH installation. It tells, that now it is possible to make connection to node's IP address via 7180 port. Simply in other words, now it is possible to open a website in web-browser to continue CDH installation. In my case address with given port looked 10.11.121.65:7180 . Further installation and configuration steps will be performed by using graphical user interface in web-browser.

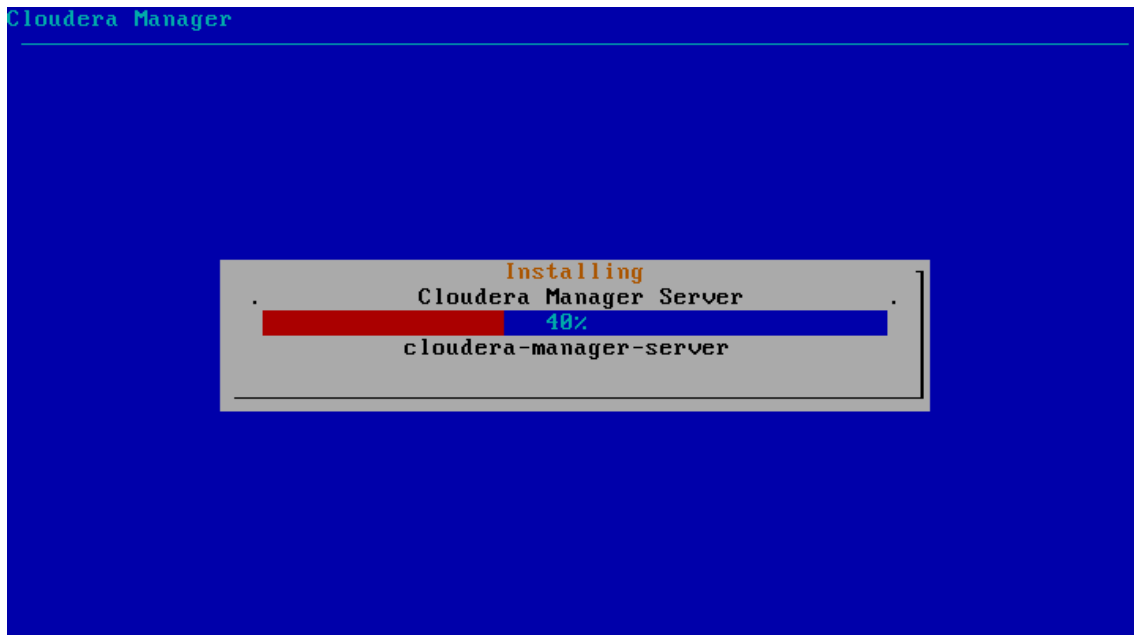


Figure 3.1. Installation of Cloudera Manager

Welcome to Cloudera. Which edition do you want to deploy?

Upgrading to **Cloudera Enterprise** provides important features that help you manage and monitor your Hadoop clusters in mission-critical environments.

	Cloudera Standard ✓	Cloudera Enterprise Trial	Cloudera Enterprise
License	Free	60 Days Post trial period, the product will continue to function as Cloudera Standard . Your cluster and your data will remain unaffected .	Annual Subscription(s) Upload License
Node Limit	Unlimited	Unlimited	Unlimited
CDH	✓	✓	✓
Core Cloudera Manager Features	✓	✓	✓
Advanced Cloudera Manager Features (click link below for details)		✓	✓
Backup & Disaster Recovery †		✓	✓
Cloudera Navigator †		✓	✓
Cloudera Support			✓

† Purchased as separate products.

For full list of features available in **Cloudera Standard** and **Cloudera Enterprise**, [click here](#).[©]

Figure 3.2. CDH edition options

First step, CDH edition has to be chosen. For freeware version, Cloudera Standard option has to be chosen (figure 2).

In the next step, nodes that will be used, have to be specified. Nodes' hostnames or IP addresses can be entered to perform search. In my case, I have entered hostnames of 5 nodes (figure 3).

In further steps CM will ask to provide password or SSH private key that is accepted by all nodes. The reason, CM needs to download and configure CDH files on each node. If we decided to install Hadoop by not using CM, all this configurations have to be performed independently on each node. If cluster

Specify hosts for your CDH cluster installation.

Cloudera recommends including Cloudera Manager server's host because it is often used for the Cloudera Management Service, and because this will enable health monitoring for that host.

Hint: Search for hostnames and/or IP addresses using [patterns](#).

5 hosts scanned, 5 running SSH. [New Search](#)

<input checked="" type="checkbox"/> Expanded Query	Hostname (FQDN)	IP Address	Currently Managed	Result
<input checked="" type="checkbox"/> m4	m4	10.11.121.64	No	✓ Host ready: 3 ms response time.
<input checked="" type="checkbox"/> m5	m5	10.11.121.65	No	✓ Host ready: 0 ms response time.
<input checked="" type="checkbox"/> s2	s2	10.11.121.62	No	✓ Host ready: 2 ms response time.
<input checked="" type="checkbox"/> s3	s3	10.11.121.63	No	✓ Host ready: 0 ms response time.
<input checked="" type="checkbox"/> s6	s6	10.11.121.66	No	✓ Host ready: 2 ms response time.

Figure 3.3. Specifying nodes

consists of hundreds nodes, it will require much more effort, than using CM.

After providing required SSH login credentials, CM begins to downloading and installing phase, this is usually the longest part of installation (figure 4). CM will also create few databases, that will be later used by HDFS for storing metadata about files' blocks, CM monitoring web services also requires databases. Usually open-source RDBMS like MySQL or PostgreSQL are used.

Cluster Installation

Installation completed successfully.

5 of 5 host(s) completed successfully.

Hostname	IP Address	Progress	Status
m4	10.11.121.64	<div style="width: 100%; height: 10px; background-color: green;"></div>	✓ Installation completed successfully. Details
m5	10.11.121.65	<div style="width: 100%; height: 10px; background-color: green;"></div>	✓ Installation completed successfully. Details
s2	10.11.121.62	<div style="width: 100%; height: 10px; background-color: green;"></div>	✓ Installation completed successfully. Details
s3	10.11.121.63	<div style="width: 100%; height: 10px; background-color: green;"></div>	✓ Installation completed successfully. Details
s6	10.11.121.66	<div style="width: 100%; height: 10px; background-color: green;"></div>	✓ Installation completed successfully. Details

Figure 3.4. Installation completed successfully on each node

Installation completed successfully on each node (figure 4), now it is time to perform configurations. CM will inspect the cluster's nodes and provide its own scheduled design of nodes' roles, of course roles scheduling can be changed manually. Then CM provide user to set configuration values in web-browser window, but in fact it is just simple editing of configuration xml files that were described in chapter 2 during installing Hadoop in pseudo-distributed mode. Additionally CM provides a brief description of each configured property, what is sometimes very helpful (figure 5).

Review configuration changes

Set the following configuration values for your new role(s). Required values are marked with *.

Group	Parameter	Recommended Value	Description
Service hbase1			
Service-Wide	HDFS Root Directory* hbase.rootdir	/hbase default value	The HDFS directory shared by HBase RegionServers.
Service hdfs1			
DataNode (Default) Show Members	DataNode Data Directory* dfs.datanode.data.dir	<input type="text" value="/dfs/dn"/> Reset to empty default value	Comma-delimited list of directories on the local file system where the DataNode stores HDFS block data. Typical values are /data/N/dfs/dn for N = 1, 2, 3... These directories should be mounted using the noatime option and the disks should be configured using JBOD. RAID is not recommended.
DataNode (Default) Show Members	DataNode Failed Volumes Tolerated dfs.datanode.failed.volumes.tolerated	0 default value	The number of volumes that are allowed to fail before a DataNode stops offering service. By default, any volume failure will cause a DataNode to shutdown.
DataNode (1) Show Members	DataNode Data Directory* dfs.datanode.data.dir	<input type="text" value="/dfs/dn"/> Reset to empty default value	Comma-delimited list of directories on the local file system where the DataNode stores HDFS block data. Typical values are /data/N/dfs/dn for N = 1, 2, 3... These directories should be mounted using the noatime option and the disks should be configured using JBOD. RAID is not recommended.
DataNode (1) Show Members	DataNode Failed Volumes Tolerated dfs.datanode.failed.volumes.tolerated	0 default value	The number of volumes that are allowed to fail before a DataNode stops offering service. By default, any volume failure will cause a DataNode to shutdown.
NameNode (Default) Show Members	NameNode Data Directories* dfs.namenode.name.dir	<input type="text" value="/dfs/nn"/> Reset to empty default value	Determines where on the local file system the NameNode should store the name table (fsimage). For redundancy, enter a comma-delimited list of directories to replicate the name table.

[Back](#) [Continue](#)

Figure 3.5. Setting of configuration values

By confirming all entered configurations, installation process is completed. Cluster is ready to be used for solving contemporary problems of Big Data. Now all effort can be focused on developing algorithms of distributed data processing. CM can further be used for monitoring and administration services (figure 6).

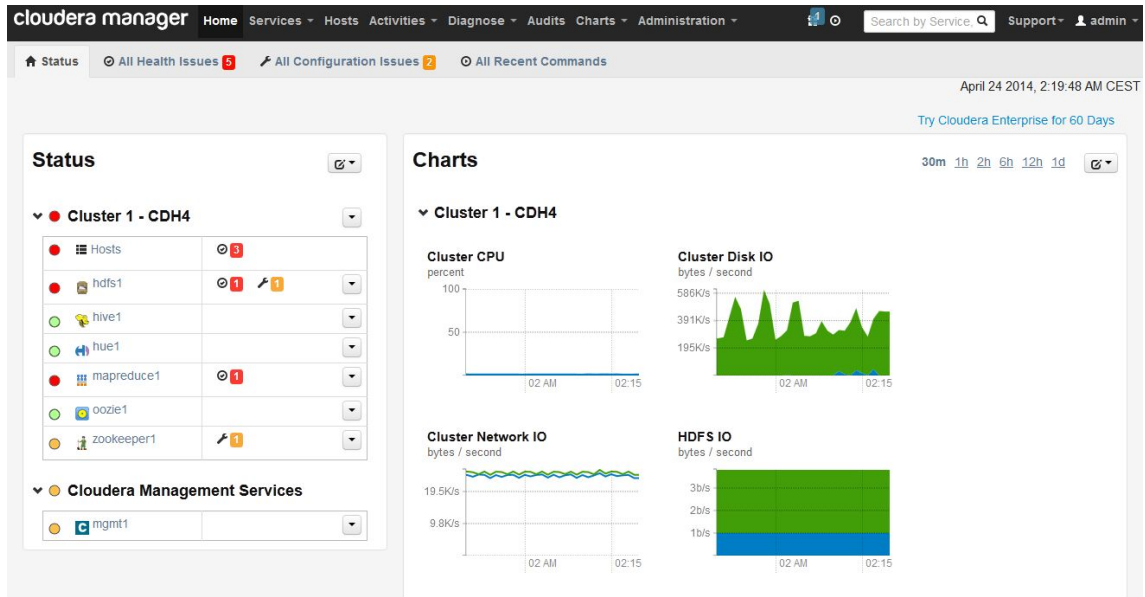


Figure 3.6. CM monitoring website

Chapter 4

Data Analysis

4.1 Data description

In previous chapters I have described what are Big data and how to prepare distributed data processing system for them. Now I can proceed to the final aim of my bachelor thesis, usage of Hadoop. In this chapter I am going to demonstrate usage, of previously prepared Hadoop framework by analyzing freely available weather data-sets.

Data-sets used in my project consists of daily measurements provided from 5162 European weather stations in period approximately since the end of 18th century till today. Naturally different stations provided own data in different periods. Used data-sets were obtained from National Oceanic and Atmospheric Administration (NOAA). “NOAA’s National Climatic Data Center (NCDC) is responsible for preserving, monitoring, assessing, and providing public access to the Nation’s treasure of climate and historical weather data and information. “
[<http://ncdc.noaa.gov/>]

Total amount of unarchived data was 18.8 GB. Data records contained information mainly about temperatures, rainfalls, wind speed and direction, humidity and few others less interesting records. In my project I employed just temperature records. Generally stations are provide three daily temperature values, maximal temperature, minimal temperature and mean daily temperature.

As I mentioned before, data-sets contains a lot of additional information, that I will not employ in my project, therefore first step required to be done is to filter them and obtain just useable information. For better organizations data-sets of NCDC are splitted on many small files divided by stations and category of information they contain. This form data storage isn’t suitable for Hadoop’s HDFS. HDFS is designed to store less number of very large files, not for storing many small files. This is caused by fact that files in HDFS are splitted on blocks of strictly give size, mainly 64MB. This causes that even small files of size about 1-2 MB, each will consume full 64 MB data block. HDFS and this technique was deeply described in previous chapters. Due to mentioned discrepancy of weather data-sets, each file firstly need to be merged into one big input file before loading to HDFS.

4.2 Tasks description

1. Define main aim to be reached by data analysis.
2. Download weather data from NCDC servers.
3. Filter and merge usable (temperature) records into one file.
4. Load data to HDFS.
5. Run MapReduce analysis job.
6. Study and display obtained results.

4.3 Tasks realization

4.3.1 Tasks definition

Main aim: Analyze daily temperature data sets for 20th century. Calculate mean yearly temperature over all European weather stations that have been providing data records constantly during period 1900 – 2000. Design results in graph and apply Pearson's chi-squared test over obtained results.

4.3.2 Downloading data

Data were obtained from NCDC's ftp server.

```
ftp://ftp.ncdc.noaa.gov/pub/data/
```

4.3.3 Preparing data

First step – parsing of file with the list of all weather stations and information about them. File contained 12 fields. For aims of my project I am interested just in 4 of them:

STAID - Station identifier

ELEI - Element identifier, represents category of provided records (temperature, rainfalls, etc)

START - Begin date of measurements YYYYMMDD

STOP - End date of measurements YYYYMMDD

From this file were extracted station's IDs of weather stations that have been providing temperature measurements in period from 1900 to 2000.

Second step – merging of all temperature file of stations with ID's obtained from the first step. As result, one big file with all temperature records is obtained.

All file parsing and merging actions were made by applications written in Java. Source Java code is provided in attachment at the end of document.

■ 4.3.4 Loading data to HDFS

Firstly there is created a directory in HDFS for project's files. Then input files are copied to HDFS. All actions are made by running Hadoop shell commands from command line.

```
# hadoop fs -mkdir /user/oleg/BP/pr1
# hadoop fs -mkdir /user/oleg/BP/pr1/input
#
# hadoop fs -copyFromLocal /root/mergedFile /user/oleg/BP/pr1/input
```

Now file is stored in distributed file system.

■ 4.3.5 Performing MapReduce job

MapReduce job obviously consists of two phases, Map and Reduce.

Map task – performs parsing of date, temperature, checks quality tag and if the parsed record (line of read input file) satisfy given criteria, results are paired into key – value pairs that are sent to the reduce task. Key – value pair is represented by year as key, and temperature as value.

When all map tasks are finished, Hadoop shuffles all map tasks' outputs and generates input for reduce task. Input for reduce task is similar to the output of map tasks. Again it is a key – value pairs. Keys represent years, but value is in form of iterable list of temperatures corresponding to the each year.

Reduce task – performs calculation of mean temperature for each year over all stations. Output of reduce tasks is also output of whole MapReduce job. Output has a form of text file with two fields per line. First field is a year in range 1900 – 2000, second field corresponding mean temperature.

Whole MapReduce job was written in Java, and consists of three Java classes: TemperatureMapper class, TemperatureReducer class, TemperatureJob class. All source files and corresponding documentation are provided in attachment at the end of document.

4.3.6 Observing results

Result – output file contains 101 records of year in range 1900-2000 and corresponding calculated mean temperature. Values of output file have been sketched to the graph. Additionally in the graph can be observed linear regression a with a slowly rising tendency.

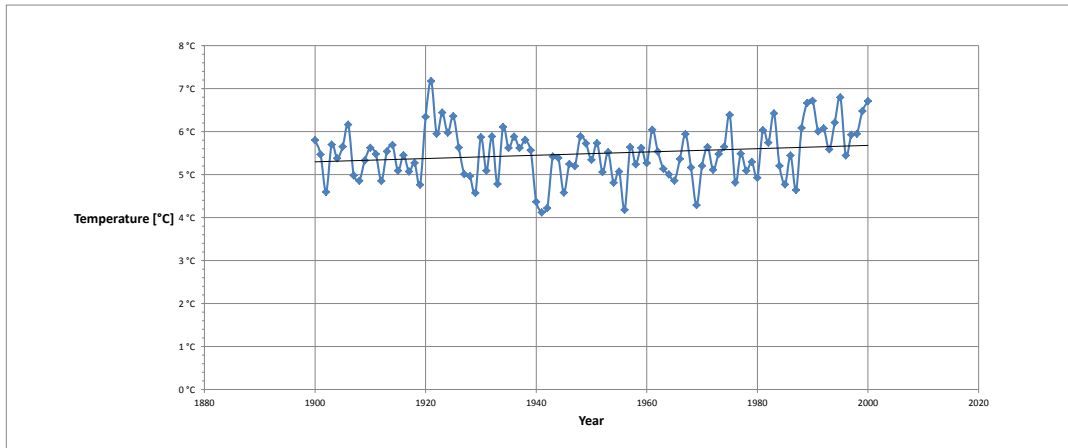


Figure 4.1. Mean temperatures by year in range 1900 - 2000

To provide more deeply analysis of weather data I have applied a statistical test over the calculated results. I have used Pearson's chi-squared test, that can be applied to test null hypothesis stating that the frequency of certain events observed in a sample is consistent with a particular theoretical distribution.

To provide more deeply analysis of weather data I have applied a statistical test over the calculated results. I have used Pearson's chi-squared test, that can be applied to test null hypothesis stating that the frequency of certain events observed in a sample is consistent with a particular theoretical distribution. [7]

4.3.7 Pearson's chi-squared test of calculated results

General formula of Pearson's chi-squared test:

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

H0 – null hypothesis:

Hypothesis says, that set of yearly mean temperatures in 20th century, represents a uniform distribution. In other words, slow rising tendency seen from the graph doesn't mean that there is a Global warming.

H1 – alternative hypothesis:

Alternative hypothesis says the negation of H0, thus temperature values don't

Year	1900	1901	1902	...	1998	1999
Observed Temp.	5.8032393	5.468648	4.595253	...	5.947022	6.476354
Probability	0.01	0.01	0.01	...	0.01	0.01
Sum of Temp.	547.66899					
Expected Temp.	5.4766899	5.47669	5.47669	...	5.47669	5.47669
$(O - E_i)^2$	0.1066345	6.47E-05	0.776932	...	0.221213	0.999328
$\frac{(O - E_i)^2}{E_i}$	0.0194706	1.18E-05	0.141862	...	0.040392	0.182469
$\chi^2_{0,95} (99) = 124.34 > 6.531$						

Table 4.1. Pearson's chi-squared test of mean temperatures

represent a uniform distribution and Global warming is a probable consequence. Desired significance level is 0.05 (or equivalently, 5%).

Conclusion:

According to the calculated results of Pearson's chi-squared test, hypothesis H0 can not be disproven, thus H1 can not be proven, within the desired significance level. It means, that yearly mean temperatures in years 1900 - 1999 could have a uniform probability distribution. Slowly rising tendency that is observed in the graph, can be explained as a fact, that temperature values are periodically rising and falling with time. Otherwise, if the temperature values really have a rising tendency, it requires longer period to be examined to prove that its values don't represent uniform probability distribution, therefore fact of Global warming wasn't proved.

4.3.8 Minimal and maximal daily temperatures

Similar calculations using Hadoop have been performed for minimal and maximal daily temperature records. Results of MapReduce job were not tested by Pearson's chi-squared test, as its result would be obviously the same as in the case of mean temperatures. Values are just sketched to the graph, including linear regression line. Tendencies look interesting. While minimal daily temperatures represent similar as mean temperatures, slowly rising tendency, daily maximal temperatures have an opposite character, slowly decreasing tendency. Graphic of daily maximal temperatures designed by red curve in the graph doubts statements about Global warming.

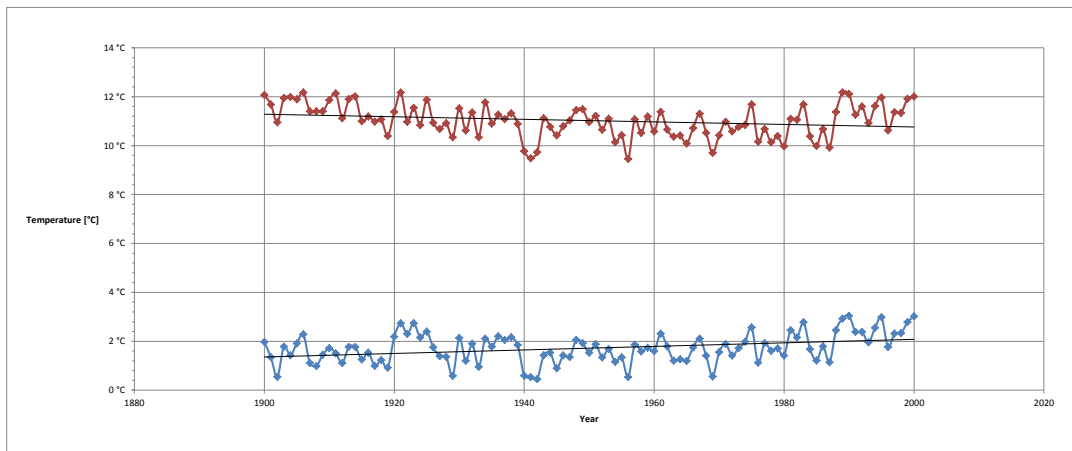


Figure 4.2. Minimal and maximal daily temperatures tendencies in years 1900 - 2000

Chapter 5

Conclusion

The main aim of this project was to prepare a distributed data processing environment for Big data. After this was fulfilled, the rest of assignment is represented for developing an algorithms to exploit prepared environment. All work presented in my project was based on two main principles, to use just freely available tools and to demonstrate solutions ready to be used in real life applications.

Created data processing environment is based on freely available Apache Hadoop framework. Given framework represents one of the most perspective technology for processing very large amounts of information. Environment preparation was performed in two steps. Firstly framework was installed to be used simply on one server in pseudo-distributed mode. This enabled me to deeply introduce with the given system, to study its whole complexity and to prepare solutions how to proceed with given tasks. Second step was an installation of framework in real distributed mode.

Hadoop cluster prepared in my project contains five computers. It is not much, it represents just very small cluster, if we compare it with the Hadoop clusters used in real life applications, but on the other hand it is absolutely enough for my academical purposes. Hadoop is designed to work with MapReduce algorithm model. Programs implementing MapReduce algorithm are designed to work independently on the number of nodes in the cluster. That's why five nodes' cluster absolutely satisfied requirements of usability.

After system was installed and configured, it was ready to be exploited. To make experiments I required some sample data. There was more options for data analysis, but data-sets obtained from National Climatic Data Center seemed to be the most interesting and seemed to have the most suitable structure for demonstrating MapReduce algorithm in work. Obtained data-sets contained daily records from thousands European weather stations that have provided their measurements in period of last 3 centuries. Total size of raw data was 18.8 GB. Data were loaded to distributed storage system and successfully processed by a MapReduce application. Results of the application were further tested by Pearson's chi-squared test. This enabled to make statements about Global warming.

Hadoop framework and distributed data processing model itself seems to be very perspective. During performing this project I have gained a lots of experiences related to Big data sphere and I hope, I will be able to employ them in my future life.



References

- [1] WHITE, Tom. Hadoop: the definitive guide. 3rd ed. Sebastopol: O'Reilly, 2012, xxiii, 657 s. ISBN 978-1-449-31152-0.
<http://hadoopbook.com/>.
- [2] LAM, Chuck. Hadoop in action. Greenwich: Manning Publications, 2011, xxi, 312 S. ISBN 978-1-935182-19-1.
- [3] SAMMER, Eric. Hadoop operations. 1st ed. Sebastopol, CA: O'Reilly, 2012, xii, 282 p. ISBN 14-493-2705-2.
- [4] OWENS, Jonathan R, Jon LENTZ a Brian FEMIANO. Hadoop real-world solutions cookbook: Realistic, simple code examples to solve problems at scale with Hadoop and related technologies. Birmingham [England]: Packt Pub., 2013, iv, 298 p. ISBN 978-1-84951-912-0.
- [5] Apache Hadoop 2.2.0 Documentation [online]. 2014 [cit. 2014-03-24].
<http://hadoop.apache.org/docs/stable/>.
- [6] CDH4.6.0 Documentation [online]. [cit. 2014-05-23].
<http://www.cloudera.com/content/support/en/documentation/cdh4-documentation/cdh4-documentation-v4-latest.html>.
- [7] NAVARA, Mirko. Pravděpodobnost a matematická statistika. Vyd. 1. Praha: Nakladatelství ČVUT, 2007, 240 s. ISBN 978-80-01-03795-9.
<http://cmp.felk.cvut.cz/~navara/psi/>.

Appendix A

Source code: MapReduce java program

A.1 TempMapper class

```
package BP.map_red_job;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TempMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    public static final int START_YEAR = 1960;
    public static final int STOP_YEAR = 2010;

    public static final int PRAGUE_STATION_ID = 27;

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        // value = one line of input file
        String line = value.toString();

        // Year
        String year = line.substring(14, 18);
        int y = Integer.parseInt(year);

        if ( y >= START_YEAR && y <= STOP_YEAR) {

            // Q_TG - quality tag [ 0 -> OK ; 9 -> wrong ]
            int qTag = Integer.parseInt(line.substring(33, 34)
                \\.replaceAll("\\s+", ""));

            // check [if quality tag = 0]
            if (qTag == 0) {

                // Temperature value
```

```
        int airTemperature = Integer.parseInt
        \\(line.substring(23, 28).replaceAll("\\s+",""));

        // write [key, value] to result
        context.write(new Text(year),
            new IntWritable(airTemperature));
    }
}

public static boolean checkID(int chkID, String line) {
    int ID = Integer.parseInt(line.substring(0, 6)
        \\replaceAll("\\s+",""));
    return (ID == chkID);
}
}
```

A.2 TempReducer class

```
package BP.map_red_job;

import java.io.IOException;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class TempReducer
    extends Reducer<Text, IntWritable, Text, FloatWritable> {

    @Override
    public void reduce(Text key,
        Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

        float result, sum = 0;
        int numOfValues = 0;

        for (IntWritable value : values) {
            sum += value.get();
            numOfValues++;
        }

        result = sum/numOfValues;
        result = result/10;

        context.write(key, new FloatWritable(result));
    }
}
```

A.3 TempJob class

```
package BP.map_red_job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class TempJob {

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

        String inputPath = "hdfs:///user/oleg/BP/pr1/inputTX";
        String outputPath = "hdfs:///user/oleg/BP/pr1/outputTX";

        Configuration conf = new Configuration();

        Job job = new Job(conf);

        job.setJarByClass(TempJob.class);
        job.setJobName("Max temperature");

        FileInputFormat.addInputPath(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setMapperClass(TempMapper.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        job.setReducerClass(TempReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```


Appendix B

Source code: PrepareFiles java program

B.1 PrepareFiles class

```
package BP.prep_files;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

/**
 *
 * @author Oleg Ostashchuk
 */
public class PrepareFiles {

    public static final int START_YEAR = 1960;
    public static final int STOP_YEAR = 2010;

    public static final String MEAN_TEMPERATURE = "TG";
    public static final String MAX_TEMPERATURE = "TX";
    public static final String MIN_TEMPERATURE = "TN";

    public static final String CATEGORY = MAX_TEMPERATURE;

    public static final String LIST_FILE = "...";
    public static final String INPUT_DIRECTORY = "...";
    public static final String OUTPUT_FILE = "..." + CATEGORY;

    public static boolean[] list = new boolean[10000];

    static void readListFile(File file) {

        BufferedReader reader = null;
        String line;

        try {
            reader = new BufferedReader(new FileReader(file));
            int lineCount = 0;
            while ((line = reader.readLine()) != null) {
                if(lineCount >= 25){
                    if (getStartYear(line)<=START_YEAR
```

```
        && getStopYear(line)>=STOP_YEAR) {
            if (getCategory(line).contains(CATEGORY)) {
                list[getStatiobID(line)] = true;
            }
        }
        lineCount++;
    }

} catch (FileNotFoundException ex) {
    ex.printStackTrace();
} catch (IOException ex) {
    ex.printStackTrace();
} finally {
    try {
        if (reader != null) {
            reader.close();
        }
    } catch (IOException ex) {
        System.out.println(ex);
    }
}

public static String getCategory(String line) {
    return line.substring(84, 86);
}

public static int getStatiobID(String line) {
    return Integer.parseInt(line.substring(0, 5)
        \\\.replaceAll("\\s+", ""));
}

public static int getStartYear(String line) {
    return Integer.parseInt(line.substring(88, 92));
}

public static int getStopYear(String line) {
    return Integer.parseInt(line.substring(97, 101));
}

public static void main(String[] args) {

    readListFile(new File(LIST_FILE));

    FilesMerger.GetFiles(list, INPUT_DIRECTORY);

}

}
```

B.2 FilesMerger class

```
package BP.prep_files;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 *
 * @author Oleg Ostashchuk
 */
public class FilesMerger {

    public static String prefix = PrepareFiles.CATEGORY;
    public static String mergedFilePath = PrepareFiles.OUTPUT_FILE;

    public static void getFiles(boolean[] list, String path) {

        File[] allFiles = new File(path).listFiles();
        File[] filteredFiles = new File[500];

        int count = 0;

        for (File file : allFiles) {
            if (!file.isDirectory()) {
                if ( isTempFile(file) && isInList(file, list) ) {
                    filteredFiles[count] = new File(path + "/"
                        + file.getName());
                    count++;
                }
            }
        }
        mergeFiles(filteredFiles, count);
    }

    public static boolean isInList(File file, boolean[] list) {
        return list[Integer.parseInt(file.getName().substring(8, 14))];
    }

    public static boolean isTempFile(File file) {
        return file.getName().contains(prefix);
    }

    public static void mergeFiles(File[] files, int count) {

        File mergedFile = new File(mergedFilePath);
```

```
    FileWriter fstream = null;
    BufferedWriter out = null;

    try {
        fstream = new FileWriter(mergedFile, true);
        out = new BufferedWriter(fstream);
    } catch (IOException ex) {
        ex.printStackTrace();
    }

    int lineNum;

    for (int i = 0; i < count; i++) {

        lineNum = 0;
        System.out.println("merging: " + files[i].getName());
        FileInputStream fis;
        try {
            fis = new FileInputStream(files[i]);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(fis));

            String aLine;
            while ((aLine = in.readLine()) != null) {
                if (lineNum>20) {
                    out.write(aLine);
                    out.newLine();
                }
                lineNum++;
            }
            in.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    try {
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

Appendix C

CD contents

Path	Description
bp.pdf	Bachelor thesis in pdf format
source_code	directory with two NetBeans projects
calculations.xlsx	MS Excel spreadsheet with graphs and calculations