Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

Final Master Thesis

Data Locality in Hadoop

Justyna Kałużka

supervised by

Dr. Oscar Romero Moral
Petar Jovanovic

July 2016

Abstract

Current market tendencies show the need of storing and processing rapidly growing amounts of data. Therefore, it implies the demand for distributed storage and data processing systems. The Apache Hadoop is an open-source framework for managing such computing clusters in an effective, fault-tolerant way.

Dealing with large volumes of data, Hadoop, and its storage system HDFS (Hadoop Distributed File System), face challenges to keep the high efficiency with computing in a reasonable time. The typical Hadoop implementation transfers computation to the data, rather than shipping data across the cluster. Otherwise, moving the big quantities of data through the network could significantly delay data processing tasks. However, while a task is already running, Hadoop favours local data access and chooses blocks from the nearest nodes. Next, the necessary blocks are moved just when they are needed in the given ask.

For supporting the Hadoop's data locality preferences, in this thesis, we propose adding an innovative functionality to its distributed file system (HDFS), that enables moving data blocks on request. In-advance shipping of data makes it possible to forcedly redistribute data between nodes in order to easily adapt it to the given processing tasks. New functionality enables the instructed movement of data blocks within the cluster. Data can be shifted either by user running the proper HDFS shell command or programmatically by other module like an appropriate scheduler.

In order to develop such functionality, the detailed analysis of Apache Hadoop source code and its components (specifically HDFS) was conducted. Research resulted in a deep understanding of internal architecture, what made it possible to compare the possible approaches to achieve the desired solution, and develop the chosen one.

Acknowledgements

Foremost, I would like to express my gratitude to my thesis supervisors Dr. Oscar Romero and Petar Jovanovic from Department of Service and Information System Engineering at Universitat Politècnica de Catalunya. I thank them for giving me opportunity to work with them on this innovative project, and then for their help, patience and advices.

I am also very thankful to my coordinator at Łódź University of Technology Dr. Małgorzata Napieralska and director of Department of Microelectronics and Computer Science Prof. Andrzej Napieralski for their help and encouragement. Without their involvement my exchange stay at UPC would not be possible.

My thanks also go to all my friends for cheering me up after long days in front of computer.

Finally, I would like to thank my parents for their support through almost twenty years of my education, for unconditional believing in me and encouraging to reach higher, and my boyfriend Rui for his understanding, being always for me during the tough moments, and not letting me to give up on the way to achieve my goals.

Contents

Chapter 1

Introduction

According to the definition[1] *data* is the general term specifying facts and information which can be transmitted or processed in order to be used for calculations or analysis. Also, data can refer to both useful and irrelevant, as well as the redundant information. One can say that overall data transferred through the media of communication is the way people exchange facts, memories, and knowledge.

The amount of data generated by people but especially computers is rapidly growing last years because of the development of modern technologies and therefore the digitalization of everyday life. In addition, professional sectors like health care, business and social sciences require storing and analysing loads of data. Nowadays one can talk about the quantity which was impossible to store so also proceed even few years ago, mainly because of the hardware limitations. Moreover, the current trends do not seem to stop the next years. This is why there is a need for creating and continuous development of technologies which enable managing and analyzing these large data quantities. Therefore, also the right terms for naming the whole phenomenon appeared and nowadays, we are talking about so called *Big Data*.

1.1 Big Data

Term Big Data intuitively refers to the large datasets of size exceeding the storage ability of the typical relational databases. However, it is not possible to define the threshold limit of the data size beyond which one can talk about "big" data. Definition varies amongst the sectors but particularly shifts along with appearance of new technologies and as a result - storage capabilities [13].

The popular idea to describe the whole phenomenon is the concept of "Vs", which was the first time introduced in 2011 by Doug Laney. This American researcher and analyst initially proposed the *volume*, *velocity*, and *variety* as features describing the exploding data management challenge [12].

[1] Source: Merriam-Webster `http://www.merriam-webster.com/`

- **Volume** as the very first aspect refers to the obvious issue of the data quantity and its massive growth last years. Originally, Laney was referring mainly to the e-commerce getting popular that time 15 years ago. Thence he underlined the impact of the online trading in the increasing amount of data generated by machines, not only humans.
- **Velocity** refers to the speed at which data is generated. In order to make these bigger amounts of data more efficiently used, the increased velocity of data transmission was needed. Caching provides the instant access and live analysis of streaming data, whereas point-to-point data routing enables the massive and continuous flow.
- **Variety** describes one of the main problems of Big Data, i.e., common lack of structure. Many companies store data without preliminary knowing its purpose, different formats and sources complicate later data mining and analysing.

Original concept of 3 Vs has been extended during years by two more terms:

- **Veracity** refers to the poor quality of stored data due to its inconsistency and incompleteness. Referred as the big challenge for data scientists, lack of accuracy and noise raise the need of cleaning data before any further analysis.
- **Value** is specified as the most important of the Vs features [14]. Marr underlines the usefulness of collected information and ability to turn it into desired value. Value is sometimes also called Validity paying even more attention to its relativeness [17].

There appears other terms (authors seems to diligently follow the rule of V's) like **Variability** which is often mistaken with Variety, but specifies not formats of collected data but its attributes which are especially relevant in data analysis. Even **Visualization** is sometimes specified as a Big Data feature. Being extremely crucial to make huge amounts of data easily comprehensible, it can also involve User Experience Design [18].

1.1.1 Big Data Applications

The term Big Data is sometimes criticized as too general - as "all talk, no action" together with other "trendy" terms as Data Science, Cloud Computing or Internet of Things, which appear nowadays in many aspects of modern world [2].

The best example are **health care** centers. Integrating data from medical records enables hospitals and other institutions to store patients' health history. Moreover the medical research centers are continuously working on new treatment methods and vaccines. Results of these experiments are stored very often without preliminary defined purpose and used in later tests.

The obvious way how all internet users are generating zetabytes of data are **social media** and content shared online by everyday users. Such data is of special interest for **business and social sciences**. Companies use new technologies

as the main promotion channel and analyze all stored information in order to target their customers.

Both public and private sectors, as well as the general industry, are benefiting from storing and processing data. Current trends show that this tendency will be growing during the next years.

1.1.2 Big Data Technologies

When it comes to the big volumes of data, the next question is how to handle it in an efficient, and at the same time cost-effective, manner. Therefore, increasing need of storage and processing implies the development of new technologies. The reasonable consequence was that the top companies in the technological world got involved in creating necessary solutions. This is how the project now known as the Apache Hadoop and licensed as the open-source software was created.

This framework enables the distribution of storage and data processing across the computers cluster - set of connected machines, called also nodes. Files are divided into smaller parts (chunks, blocks) and can be replicated on more than one machine for providing more reliable and fault-tolerant data processing. One of the Hadoop's principles states "Moving Computation is Cheaper than Moving Data". It states that it is more efficient to execute tasks close to the node with needed data rather than moving the data itself, especially in case of huge blocks [4].

Chapter 2 elaborates further the architecture of framework focusing on its components as well.

1.2 Contribution to open-source project

What is worth to mention - Hadoop is a free and fully open-source framework under license of Apache Software Foundation, i.e., available to be modified and enhanced by anyone. The Apache projects are identified as developed in collaborative process by volunteers (often referred as the open-source community) and with open and pragmatic software license.

Open source community is very varied but always helpful and open to new contributions. Informations about Hadoop and its components are always free and available online - not only documentation but also numerous tutorials, scientific papers and other studies. New functionalities developed with this thesis will be also shared with the community in order to make it available for other contributors.

1.3 Motivation

Although Hadoop is an effective way to store and process huge amounts of data in a reasonable time, it obviously has some bottlenecks. One of them refers to so called Data Locality. After a new task from user is recognized by the system,

it is processed and system recognizes which of data blocks are needed for that task. By default the local data access is favored over this from the remote nodes.

Although one of the Hadoop's basic concepts is to move computation to data, framework handles data shipping with its built-in scheduler. Blocks of data are transferred just before they are needed. An alternative way, which would significantly increase the system performance, is to transfer data in-advance.

Moving data blocks between nodes would enable studies on more advanced scheduling algorithms. So called in-advance data shipping could allow user to manually relocate data, rather than rely on default arrangement, and take advantage later from its locality. Currently, Hadoop automatically organizes blocks across cluster without user interference. Therefore designing and developing functionality, so called forced or instructed replication, is very challenging and innovative [10].

This thesis contains the analysis of possible approaches which could make it possible to develop the desired functionality. Later the proposed solution is elaborated in more detail and with the description of how extra replication functionality was developed.

1.4 Outline

This thesis is the extract and description of research about the Apache Hadoop and is aimed to explain the studies on Data Locality as the framework's bottleneck. The following chapter introduces Hadoop and its components together with the brief history of development focusing especially on HDFS as it is later put to the further analysis.

Chapter 4 focuses on code analysis and possible approaches to the desired solution. There is also the detailed description of compiling the binary source code of Hadoop, as well as setting single- and multi-node clusters. In the end, the chosen approach is elaborated with view of code which is going to be altered.

Chapter 5 contains class diagrams of enhanced code and explains the modifications (which are enclosed in full version in Appendices). In the end there is chapter 6 with summary and possible path of future development, and all Appendices.

Chapter 2

Background - Hadoop framework

This chapter briefly outlines the development history and the general architecture and then elaborates the individual components, in view of their role in the whole framework, paying special attention to Hadoop Distributed File System (HDFS).

2.1 History

Few years before the first ,,Big Data" popularity peaks, Google, at that time already fast expanding technological company, had to tackle the problem of rapidly growing volume of web content, in order to index it and provide search engine users with desired results in a reasonable time.

First revolutionary paper was by Ghemawat et al. from 2003 which presented a scalable distributed file system with high aggregate performance meeting the Google's data processing needs [5]. The general concept was similar to other systems - data is split into chunks and stored inside the cluster of commodity machines. The given solution treated components failures as something anticipated rather than the exception and therefore provided the fault tolerance by monitoring, replication of crucial data and recoveries.

One year later, Dean and Ghemawat presented a new model of data processing across large clusters, called MapReduce. Article was the first introduction of this currently very popular programming model, MapReduce, it introduced the new way of computation in terms of separated *map* and *reduce* functions. Authors proposed that programs written in that way are automatically parallelized and executed across the large-scale clusters. Despite small changes, MapReduce remained in the same form, which shows how big success the concept was in 2004 [3].

In parallel to Google research, Doug Cutting and Mike Cafarella later known as the co-founders of Hadoop, were working on improved open-source search

engine - project called Nutch, which was aimed to crawl and index hundreds of millions of webpages. After adding the underlying file systems, the open-source version of MapReduce with Nutch on the top, the project got the new name Hadoop (after Cutting's son yellow plush toy elephant) [7].

The next stages of Hadoop development were connected with company Yahoo where Cutting started to work in 2006. The Nutch web crawler remained as a separate project whereas Hadoop as an open-source Apache Software Foundation project. The first official version 0.1 was released in April 2006 [4] and firstly was deployed on Yahoo servers. Even though it was being developed inside Yahoo, Hadoop stayed an open-source project maintained and licensed by Apache Software Foundation - non-profit corporation which can be described as decentralized community of developers.

2.2 Architecture

After the first official release Hadoop authors were continuously working on its development and finally Hadoop 1.0.0 was released in December 2011. This major release Apache Hadoop consisted of so called ,,Two Pillars" of Hadoop 1.x - file storage HDFS and MapReduce on top of it. It considerably popularized MapReduce concept and presented the potential of distributed data processing.

From the architecture point of view, Hadoop uses so called master/slave architecture where one device has control over others. As shown on Fig. 2.1 both in MapReduce and HDFS components one can distinguish masters and slaves [8, 1].

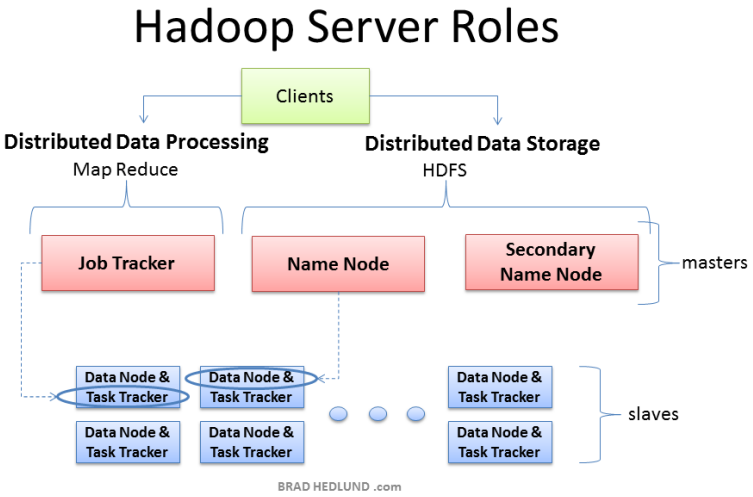

Figure 2.1: Hadoop master-slave architecture[8]

Alongside maintaining Hadoop 1 (which was enhanced only till version 1.2.1) authors were working on different branch which was released in May 2012 as Hadoop 2.0.0. Being continuously in development project's newest available stable version is 2.7.2 announced in January 2016. Following releases are published every few months.

With switching to the next major release Hadoop's primary components were re-written to add new functionalities. Fig. 2.2 presents architecture changes between Hadoop 1 and 2. The main difference was dividing MapReduce functionalities and decoupling separate module YARN (sometimes called also MapReduce 2.0), which took place of managing resources task. Also HDFS architecture was slightly improved. These changes eliminated such limitations of Hadoop 1 like problems with horizontal scalability and cluster restrictions (only 4000 nodes) [4].

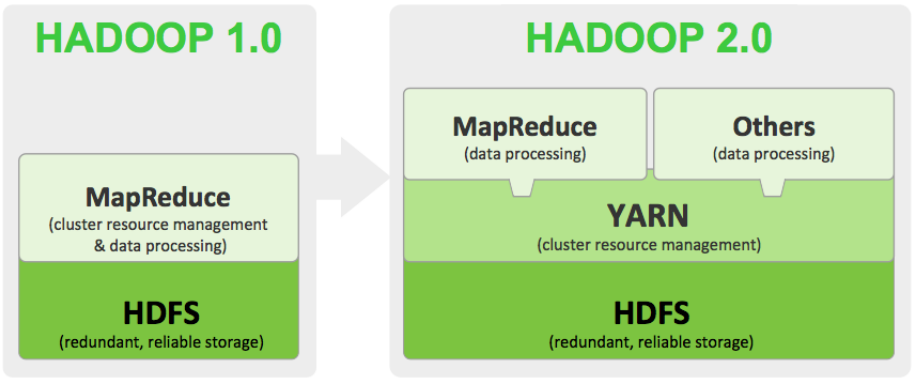

Figure 2.2: Hadoop 1 vs Hadoop 2 architecture[1]

2.2.1 MapReduce

MapReduce is the programming paradigm enabling the massive scalability across the cluster nodes. The whole concept assumes the same data flow and stages since its introduction in 2004 [3].

Although the Apache Hadoop project is just one of the MapReduce implementations, MapReduce plays the key role in the framework. Using it, users can write applications to process big amounts of raw data on large clusters with a parallel, distributed algorithm what improves speed and reliability of the system [4]. Jobs submitted to the cluster are called map and reduce tasks.

Fig. 2.3 presents the high-level model of MapReduce algorithm consisting of two stages [9]:

- Map stage
  Input data is read from HDFS and divided into data blocks. Each block is

[1]Source: Hortonworks `http://hortonworks.com/`
[2]Source: IBM `http://www.ibm.com/`

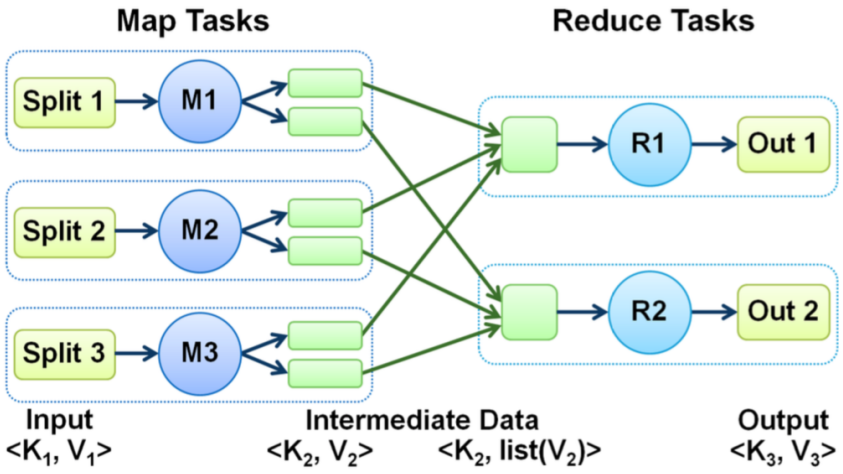

Figure 2.3: MapReduce architecture[2]

is processed by a map task line by line. Then the map function submitted by user generates the output data (in form of *key-value* pairs). Intermediate data collected in buffer is sorted, written to local disk as many file spills and merged into a single map output file.

- Reduce stage
  After transferring data to the proper node, outputs of different mappers are grouped by the previously defined key. User reduce function produces the final data which is later compressed and written as output to HDFS.

MapReduce is broadly discussed and analyzed in many papers and online sources what facilitates studies on it. It is widely popular because it is scalable allowing processing huge amount of data stored in one cluster and relatively easy to use - developers can write applications in any of popular programming languages (like Java, Python, or Ruby) to run them as MapReduce jobs.

2.2.2 YARN

Apache Hadoop YARN (Yet Another Resource Negotiator) is also called MapReduce 2.0 as it was introduced with Hadoop 2 and took some of the MapReduce functionalities. This cluster management technology can be described as a large-scale, distributed operating system used in Apache Hadoop framework and separates resources and scheduling management from data processing.

In Hadoop 1 so called JobTracker in MapReduce was responsible for resource management, tracking resource consumption and job life-cycle management. The fundamental idea of YARN is based on splitting these functionalities into

global ResourceManager (responsible for resources) and per-application ApplicationMaster (job scheduling/monitoring).

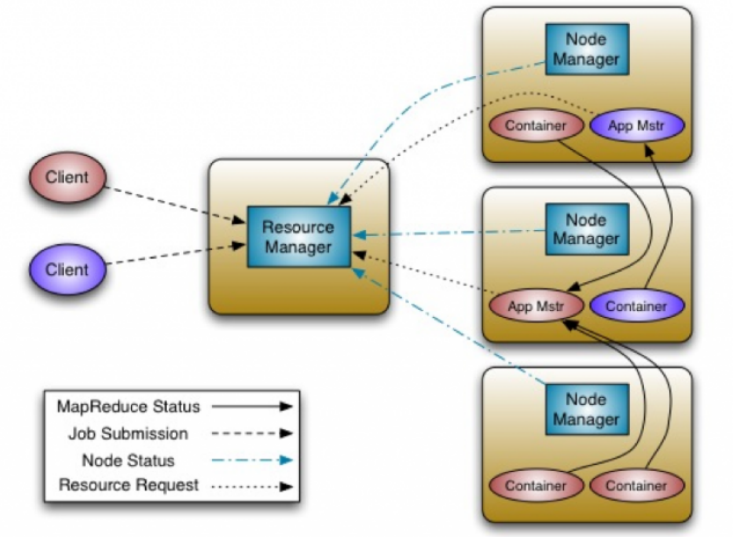

Figure 2.4: YARN architecture[3]

Fig. 2.4 illustrates the YARN architecture. The main ResourceMananger and per-node slave NodeManager are aimed to manage applications in a distributed manner. The per-application ApplicationMaster negotiates resources with the ResourceManager and collaborates with NodeMaanger in order to execute and monitor the tasks.

The ResourceManager is responsible for handling all available cluster resources among applications using two components - Scheduler and ApplicationsManager. The first one allocates resources based on the applications needs without monitoring, tracking status or restarting the failed tasks, the latter - accepts job-submissions directing them to the specific per-application ApplicationMasters.

The per-machine NodeManagers are responsible for monitoring their resources usage, tracking and reporting this information to the ResourceManager[4, 20].

2.2.3 HDFS

Hadoop Distributed File System is the primary distributed storage used in Hadoop - scalable and reliable, designed especially for large clusters of commodity servers, aiming to be fault-tolerant and running on low-cost hardware.

The HDFS architecture is presented on Fig. 2.5. Cluster with HDFS deployed over it consists of one main NameNode and DataNodes in the master-slave architecture (respectively NameNode and DataNodes in Fig. 2.1) [19, 4].

[3]Source: Hortonworks

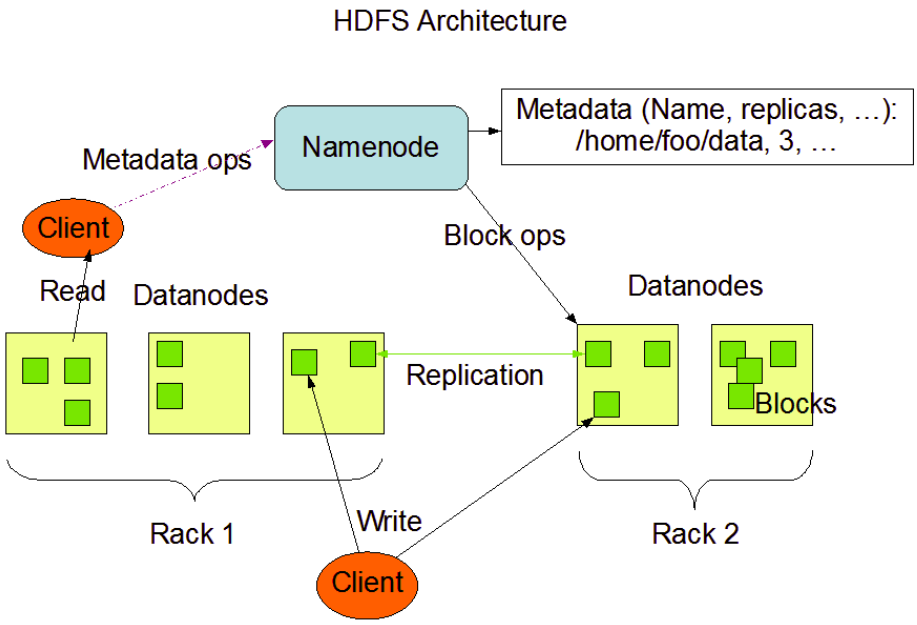

Figure 2.5: HDFS architecture[4]

System containing NameNode behaves as a master server. A single NameNode is responsible for managing the file system namespace and regulating user's access to files. Also, it performs the typical file system operations (like copying, moving) on stored files.

Every node in a cluster has (typically one) DataNode responsible for its data storage where files divided into one or more blocks are kept. They perform reading and writing requests as well as the operations on blocks ordered by NameNode.

Apart from the typical distributed file system features, HDFS is designed and developed to fulfill high efficiency goals. Hardware failure is not treated as exception but rather as a norm. Blocks are by default stored in more than one node in order to be easily recovered in case of the partial breakdown. Also, possible faults are monitored and quickly detected, so that nodes can be recovered.

Chapter 3

Studies on Data Locality

This chapter brings examples of studies on Data Locality in Hadoop, specifically, on MapReduce.

As very popular and having various implementations, parallel programming model MapReduce is also often discussed and widely elaborated in different scientific papers. Its open-source implementation - the Apache Hadoop framework has many contributors. During the years, there appeared many problems stopping them from improving the project performance.

Kalavri and Vlassov discus existing problems and limitations of MapReduce paradigm implemented in Hadoop. However, it leaves them open, without the proposed solutions [11]. Authors pay especially a lot of attention to the performance issues referring to the execution time and optimization techniques, as well as the complexity of the typical MapReduce jobs. Also, programming the MaprReduce jobs can be sometimes too low-level and require deep understanding of the system architecture.

Main subject of this study - Data Locality as the significant bottleneck in MapReduce implementations was also the topic of many articles investigating the different approaches and possible improvements in order to tackle the similar issues. Guo et al elaborate influence of cluster configuration on data locality and proposes new scheduling algorithm [6]. However, authors were more focused on distribution of blocks between nodes, rather than later impact of that on running tasks. Good distribution of data across nodes may help to reduce later cross-switch network traffic which is the common problem in data-intensive computing systems. Authors of this paper were analyzing the default Hadoop scheduling in terms of its optimality in comparison to their proposed called Linear Sum Assignment Problem.

Wang et al. focuse on data locality from the stochastic network perspective and balance between locality and load-balancing [21]. Next, it presents the new queueing architecture and map task scheduling algorithm. Their solution is aimed to asymptotically minimize the number of backlogged tasks.

Indeed the first idea was to base extra replication and moving data on MapReduce. As an older module, MapReduce is very well documented, also on other

sources than official documentation [1]. However, modifications of MapReduce cause only different assigning resources to jobs so the solution is not to move the computation closer to data but opposite - to modify HDFS instead of MapReduce. It also implies the possibility to use the added feature in other applications build on to of HDFS, like Spark, Pig or Hive.

Currently, tasks which are going to be executed require data which is downloaded from remote clusters. The process of collecting data needed for current task starts just before that execution. Instead transferring data could be done in parallel to the previous tasks. As HDFS actively uses replication of data, the additional transferring could be done executing replication with specified favoured nodes.

The main task of this thesis is to design and develop solution which enables the force replication which can be used by scheduler for HDFS as well as for forcing more balanced data distribution across the cluster.

Chapter 4

Code analysis

This chapter firstly presents how to build Hadoop source code and how to prepare cluster in order to start development and contribution process. Next it compares the possible approaches aiming to achieve the desired solution of on-request moving blocks between nodes.

4.1 Software preparation

In order to study HDFS, there was a need to work with the Hadoop source code.

Apache Software Distribution provides numerous releases of Hadoop Software, starting from 0.10.1 up to the newest 2.7.2 released in January 2016. Every version after 2.0 can be downloaded both as binary and source code archives. Apache also provides the detailed instruction how to contribute to the project - starting from possible improvements which could be implemented, through setting environment up to actual changes, testing them and sharing with the community [4].

Every contributor should start from compiling the available source code before changing it. Although there are numerous tutorials apart from the official one, including blogs and technical forums, it turned out to be very challenging due to many requirements and specific configuration of operation systems. Also, apparently it was a common problem as many people were looking for help.

During the whole process of working on this master thesis, all the involved machines were running a Debian-based Linux operating system Linux in version 14.04 LTS (Long Term Support). Therefore, the compilation process is described specifically for Linux system.

4.1.1 Building Hadoop source code

Required software
Hadoop is prepared to be built with the build automation tool Apache Maven.

There are plenty of dependencies to be installed:

- Java JDK (1.7 or later)

- ProtocolBuffer 2.5.0

- native Linux libraries - cmake, zlib, openssl, ssh and many others

Source code content
Unpacked archive (on Fig. 4.1)with source code contains many folders with specific modules.

```
j@j-Vostro:~/hadoop-2.7.2-src$ ls -1
BUILDING.txt
dev-support
hadoop-assemblies
hadoop-client
hadoop-common-project
hadoop-dist
hadoop-hdfs-project
hadoop-mapreduce-project
hadoop-maven-plugins
hadoop-minicluster
hadoop-project
hadoop-project-dist
hadoop-tools
hadoop-yarn-project
LICENSE.txt
NOTICE.txt
pom.xml
README.txt
```

Figure 4.1: Structure of Hadoop source code

Apart from the most siginificant ones in this case `hadoop-hdfs-project` and `hadoop-common-project` (which contains common utilities and libraries used within the whole project), archive ready to be built contains source code of remaining components - YARN and MapReduce - as well as the other elements used in compilation.

Compiling source code
Firstly, maven plugins need to be installed in order to generate the snapshot (an actual copy of the state) version of them. It should be done with standard command `install` which installs plugins to package to be used in other local projects:

```
$ cd hadoop-maven-plugins/
$ mv clean install
```

Then, the proper project with right parameters can be compiled from the main folder containing `pom.xml` file. The final command used here was created after combining many suggested solutions and arduous studying the maven documentation.

```
$ mvn clean install -Pdist -Dtar -Dmaven.javadoc.skip=true
↪  -DskipTests
```

It contains the following parameters:

`clean` handles project cleaning
`install` installs the package into the local repository
`-Pdist` creates binary distribution
`-Dtar` creates a TAR archive with distribution
`-Dmaven.javadoc.skip=true` omits generating JavaDoc
`-DskipTests` skips tests as they are not the priority currently and execution time would be much longer

As a result, folder `hadoop-dist/target/` contains expected archive `hadoop -X.Y.Z.tar.gz`, folder with Hadoop distribution `hadoop-X.Y.Z` as well as `hadoop -dist-X.Y.Z.jar` package.

A appendix A shows the whole code used to extract and compile the Hadoop source code archive.

The next step was to set up a single Hadoop node.

4.1.2 Setting up a single node cluster

The Apache Hadoop framework can be used on a cluster with one or more nodes. The most basic configuration allows already to perform simple operations using Hadoop MapReduce and Hadoop HDFS. In order to work on the source-code, the multi-node configuration should be set up. However, the first step to get familiarized with Hadoop from the client-user point of view, can be to experiment with the simple installation.

Pseudo-distributed, single-node Hadoop cluster can be set up using one of available Hadoop releases or, like here, from the previously compiled source code [4, 16].

Some tutorials suggest to create a special group and user in order not to interfere with the existing users permissions. In simple version it is not needed.

The basic configuration includes modifying three files:

- `etc/hadoop/hadoop-env.sh` where variable JAVA_HOME should be set according to the system configuration (e.g. to `/usr/lib/jvm/java-8-oracle`)

- `etc/hadoop/core-site.xml` where address of the default file system `fs.default.name` is specified

- `etc/hadoop/hdfs-site.xml` indicates the replication factor `dfs.replication` (on how many nodes every block should be replicated, by default 1)

Then, a new distributed-file system can be formatted and started.

```
$ bin/hadoop namenode -format
$ sbin/start-dfs.sh
```

4.1.3 Hadoop Cluster Setup

After setting up a single node cluster the next step was to prepare multi node network. Configuration of cluster looks similar in case of few nodes like thousands of nodes.

Fig. 4.2 shows the simplest case with one node acting as a master and one as a slave. As both MapReduce and HDFS have the master-slave architecture, one can separate the particular components (Job- and TaskTrackers are here indicated as MapReduce, in Hadoop 2 they are already included in YARN). The master machine deploys as NameNode and JobTracker (these roles can be also split up in two masters) while slaves - "worker nodes" as both DataNodes and TaskTrackers.

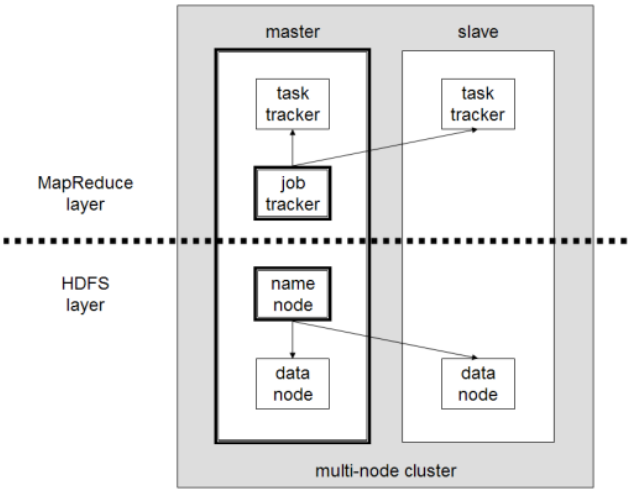

Figure 4.2: Structure of multi node cluster[15]

All machines should be connected to the same network and accessible from each other by SSH.

Configuration is the same like in case of the single node cluster apart from files `conf/masters` and `conf/slaves` on master indicating respectively the proper roles. `dfs.replication` parameter can be now set higher than 1 as files can be finally replicated on more nodes. All commands managing cluster have to be run from the NameNode [15].

Such configuration (of master and two slaves) was used through the whole development and testing process.

4.2 Different approaches within HDFS

The first step in preparing the desired solution described earlier was to broadly analyse structure of the whole Apache Hadoop project and more deeply - the

HDFS part. It was especially challenging part as being developed as an open-source software, framework lacks in some parts the detailed documentation. It makes familiarizing with particular components really difficult. The whole project is very complex containing many modules with more than 11 000 classes and almost 2 millions lines of code (with not even 400 000 lines of comments)[1].

During studies on the Apache framework, there appeared different approaches how to solve the problem of in-advance data shipping. The ideal way, according to the reusability principle, was to take advantage of existing code and reuse it. It aimed to identify parts of source code where the necessary changes should be made to provide the desired solution but also to make it reusable later for further improvements.

4.2.1 Modifying DFSOutputStream

`DFSOutputStream.java` is the file which contains direct output stream to the datanodes. First attempts to modify the source code included interference with this class. However, it is used by every HDFS writing module and therefore also changing it would influence many other components.

`DFSOutputStream` is a very low-level class, while the idea was rather to work to re-use some of higher classes. Then they invoke not only direct connection to the datanode but also take care of nodes policy, checksums of files and others.

4.2.2 Mover functionality

Mover is described as a new data migration tool since Hadoop 2.6.0 (November 2014). It officially supports the Storage Policies - specific rules in which storage type (ARCHIVE, DISK, SSD or RAM_DISK) files are allowed to be saved. However, here the most important is that Mover class apart from moving blocks between different storage types, transfer them between datanodes.

As visible in Fig. 4.3 the whole module starting from Mover contains also other tools like Dispatcher. In the end the DataOutputStream class is running in order to move single blocks of data.

[1] In June 2016, source: OpenHub `https://www.openhub.net/p/Hadoop`

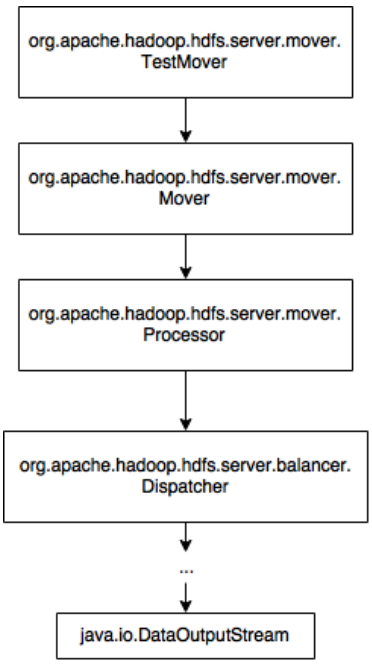

Figure 4.3: Mover class diagram

Chapter 5

Implementation

As described in the previous chapter the idea was to base a new functionality on the existing class Mover. It is more efficient to take and advantage an reuse the existing code than to re-implement the whole functionality from the scratch like described in different approaches (based on DFSOutputStream).

Mover was designed in order to support storage policy requirements but its basic tool works on single blocks of data. Therefore, it supplements the new functionality called here the *forced replication*. Replication specifies rather how many instances of the given block should be stored across the cluster. However, user can here force the replica to be stored in the exact datanode. Then, the forced replication was chosen as the working name.

Firstly, wWe will describe the architecture of our solution. The class diagram Fig. 5.1 is divided into components which were significant in this implementation. Then, the activity diagram Fig. 5.4 presents how the command is processed. The source code is shown in Appendix B.

5.1 The implementation architecture

Fig. 5.1 shows classes used in the implementation. These marked yellow are the new ones. The other ones were modified by adding new methods if needed. Analysis should be started from the the `ForceReplication` class which allows user to execute the command and specify parameters of force replication.

The basic way user can manage the cluster is through commands executed on the NameNode. These shell-like commands directly interact with HDFS and allow user manipulate files, analogously to the UNIX commands.

Main class supporting all commands operating on files is `FSCommand`. Then also class `ForceReplication` invoking the force replication extends it. `FSCommand` provides common functions for all commands like operations on extra parameters and checking whether file exists.

Command to call the force replication:

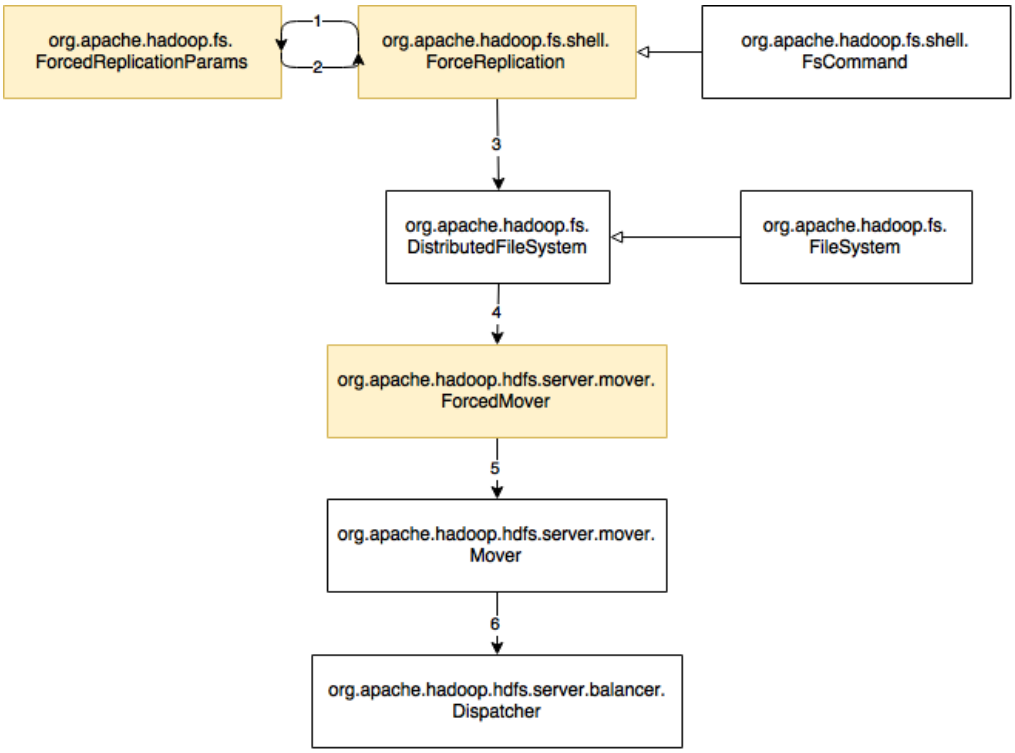

Figure 5.1: ForceReplication architecture diagram

```
$ bin/hadoop -forceReplication <blockID> <srcDatanode>
↪  <destDatanode> <path>
```

User has to determine the specific parameters:

- blockID - every block is defined by its parameters, including ID

- srcDatanode - DataNode where currently the block is located

- dstDatanoce - DataNode where block should be moved

All these informations can be obtained from HDFS, e.g. from command fsck (Fig. 5.2 where blockID and datanodes are marked).

In case user tries to execute forceReplication without proper parameters he gets the warning (like in Fig. 5.3)

If parameters are specified correctly they are passed as istance of class `ForcedReplicationParms` (steps 1 and 2 in Fig. 5.1). It is a simple file containing parameters of the given force replication job, i.e. blockId, srcDatanode and dstDatanode. Its instance is called later whenever these parameters must be used.

Step 3 in Fig. 5.1 shows that HDFS uses implementation of `FileSystem` called `DistributedFileSystem`. `forceReplication` calls it directly to proceed with the operations.

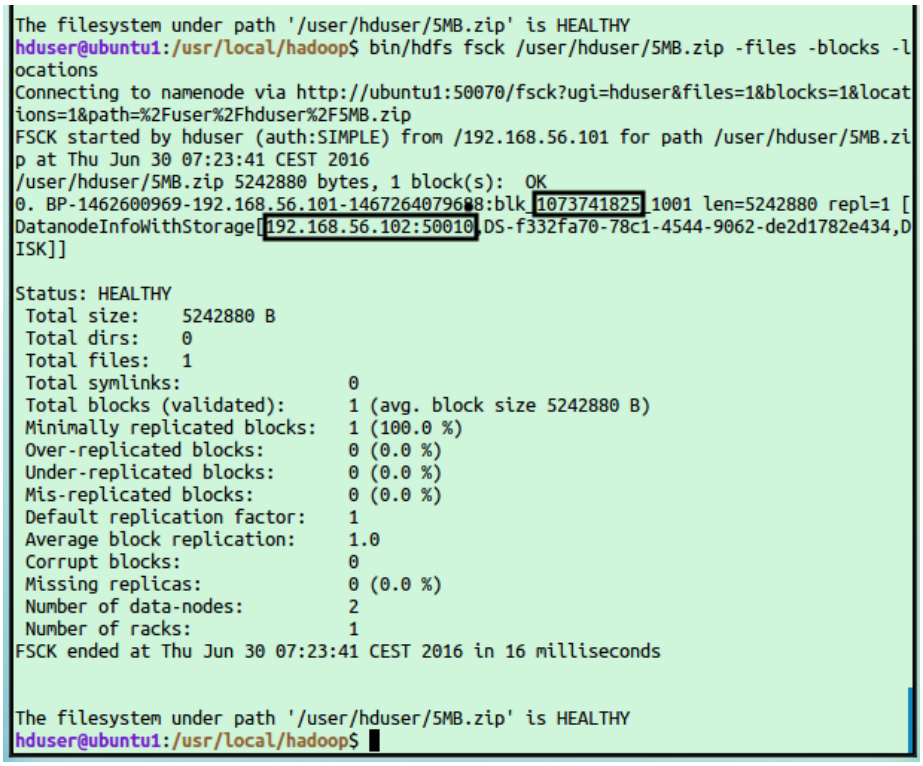

Figure 5.2: DFS Command fsck

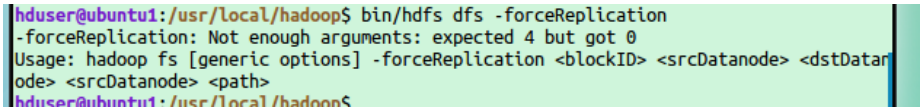

Figure 5.3: forceReplication command warning

In case force replication should be used not through HDFS command, class `Forced Mover` plays the role of connector. It is designed based on existing tests using the class `Mover`

5.2 Forced replication functionality

Fig. 5.4 shows the flow diagram.

Firstly users specifies the parameters of forced replication. `forceReplication` class as FS shell command provides checking whether file exists and if parameters were defined properly, otherwise it throws the exception. Then inside `DistributedFileSystem` it is analysed if given block is a part of specified file, as well as whether both datanodes exist. In case any of these tests is not passed,

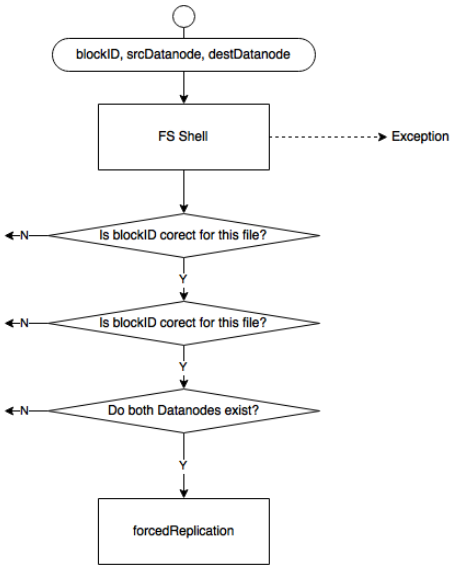

Figure 5.4: ForceReplication activity diagram

the whole command fails and user is notified why. After, the `ForcedMover` is involved.

Chapter 6

Conclusions

This master thesis considered firstly the theoretical part, which included studies on data locality in the open-source framework, Apache Hadoop, and analysis of its architecture. As an outcome of this research, we developed a new functionality for the Hadoop Distributed File System for instructing the redistribution of data across cluster nodes and improving the performance of running applications. Intuitively, the functionality we have added allows to move single blocks between datanodes in order to redistribute data in cluster and bring data in advance closer to tasks that need them.

This project involved working with the open-source software. Therefore, apart from the official documentation, framework is broadly studied in different sources, like academic papers, but also online magazines and private blogs. Also, technical forums and mailing lists give the opportunity to communicate and contact with other contributors. On the other hand, working on open-source projects can cause struggling with complex source code, developed simultaneously by many volunteers. Although it is rather challenging and requires more patience, contributing to open-source community and engagement in the community give lot of satisfaction.

6.1 Future perspectives

Even though the developed solution may have some limitations, it can be a great basis for future Hadoop enhancements. It would be enough to connect it programmatically with any further modifications working with instructed redistribution of data, such as in-advance scheduler. Then, it could significantly improve the built-in data locality solution and therefore, the resource utilization and the whole system performance.

Appendix A

Bash scripts

Compilation Bash script used to compile Hadoop from source the source code. It support both local (on a single node cluster) and cluster installation - prepares files to automate and speed up later configuration.

```
#!/bin/bash

if [[ $1 = "cluster" ]]; then
        MODE="cluster"
elif [[ $1 = "local" ]]; then
        MODE="local"
else
        echo -e "Usage:\nsudo compile.sh local|cluster"
        exit
fi

VERSION="2.7.1"
MODIFIED_FOLDER=hadoop-$VERSION-src_modification


echo -e "Starting maven compilation for "$MODE"\n"
cd $MODIFIED_FOLDER
cd hadoop-maven-plugins/
mvn clean install
cd ..
mvn clean install -Pdist,native -Dtar -Dmaven.javadoc.skip=true -DskipTests
cd ..
echo -e "Hadoop"$VERSION" compiled\n"

echo -e "Backing up...\n"
COMMON=$MODIFIED_FOLDER/hadoop-common-project/hadoop-common/src/main/java/org/
 ↪  apache/hadoop/fs
HDFS=$MODIFIED_FOLDER/hadoop-hdfs-project/hadoop-hdfs/src/main/java/org/apache/
 ↪  hadoop/hdfs
BACKUP=modified
```

```
29  CLOUD=~/Dropbox/IFE_CSIT/mgr/files_backup/
30  cp $COMMON/shell/ForceReplication.java $BACKUP
31  cp $COMMON/ForcedReplicationParams.java $BACKUP
32  cp $COMMON/shell/CommandWithDestination.java $BACKUP
33  cp $COMMON/FileSystem.java $BACKUP
34  cp $HDFS/DistributedFileSystem.java $BACKUP
35  cp $HDFS/server/mover/ForcedTestMover.java $BACKUP
36  cp $HDFS/server/balancer/Dispatcher.java $BACKUP
37  cp $HDFS/protocol/datatransfer/Sender.java $BACKUP
38  cp compile.sh $BACKUP
39  cp $BACKUP/* $CLOUD
40  echo -e "All modified files backed up in "$BACKUP" and "$CLOUD"\n"
41
42  #locally
43  if [[ $MODE = "local" ]]; then
44          LOCALLY=/usr/local
45          # cp -r £MODIFIED_FOLDER/hadoop-dist/target/hadoop-£VERSION.tar.gz .
46          rm -r $LOCALLY/hadoop
47          sudo rm -r /tmp/h*
48          cp -r $MODIFIED_FOLDER/hadoop-dist/target/hadoop-$VERSION $LOCALLY/hadoop
49          cp local_tobechanged/* $LOCALLY/hadoop/etc/hadoop/
50          sudo chown -R hduser $LOCALLY/hadoop
51          sudo subl $LOCALLY/hadoop/
52          echo -e "Hadoop"$VERSION" available in folder "$LOCALLY"/hadoop\n"
53  #cluster
54  else
55          CLUSTER=~/shared_VB
56          rm -r $CLUSTER/hadoop
57          cp -r $MODIFIED_FOLDER/hadoop-dist/target/hadoop-$VERSION $CLUSTER/hadoop
58          sudo chown -R j $CLUSTER/hadoop
59          cp $CLUSTER/tobechanged/* $CLUSTER/hadoop/etc/hadoop/
60          subl $CLUSTER/hadoop/
61          echo -e "Hadoop"$VERSION" available in folder "$CLUSTER"/hadoop\n"
62  fi
```

Appendix B

Code

ForcedReplication.java

```
1  package org.apache.hadoop.fs.shell;
2
3  import java.io.IOException;
4  import java.util.LinkedList;
5  import java.util.List;
6  import java.util.Arrays;
7
8  import org.apache.hadoop.classification.InterfaceAudience;
9  import org.apache.hadoop.classification.InterfaceStability;
10 import org.apache.hadoop.fs.FSDataOutputStream;
11 import org.apache.hadoop.fs.ForcedReplicationParams;
12 import org.apache.hadoop.fs.PathIsDirectoryException;
13 import org.apache.hadoop.fs.PathNotFoundException;
14
15 /**
16  * Forces replication of one block
17  */
18 @InterfaceAudience.Private
19 @InterfaceStability.Unstable
20
21 public class ForceReplication extends FsCommand {
22     public static void registerCommands(CommandFactory factory) {
23         factory.addClass(ForceReplication.class, "-forceReplication");
24     }
25
26     public static final String NAME = "forceReplication";
27     public static final String USAGE = "<blockID> <srcDatanode> <dstDatanode>
    ↪  <srcDatanode> <path>";
28     public static final String DESCRIPTION =
29             "Moves one block of the given file from from one datanode to another
    ↪  one.\n";
30
```

```
31      private long blockId = 0;
32      private String srcDatanode = "";
33      private String dstDatanode = "";
34
35      @Override
36      protected void processOptions(LinkedList<String> args) throws IOException {
37          CommandFormat cf = new CommandFormat(4, 4);
38          cf.parse(args);
39
40          try {
41              blockId = Long.parseLong(args.removeFirst());
42          } catch (NumberFormatException nfe) {
43              throw nfe;
44          }
45          srcDatanode = args.removeFirst().toString();
46          dstDatanode = args.removeFirst().toString();
47          displayWarning("blockID " + blockId + "; srcDatanode " + srcDatanode + ";
    ↪  dstDatanode " + dstDatanode);
48          super.processOptions(args);
49      }
50
51      @Override
52      protected void processPath(PathData item) throws IOException {
53          if (item.stat.isDirectory()) {
54              throw new PathIsDirectoryException(item.toString());
55          }
56          item.fs.setVerifyChecksum(true);
57          ForcedReplicationParams params = new ForcedReplicationParams(blockId,
    ↪  srcDatanode, dstDatanode);
58          displayWarning("forceReplication - " + params);
59          item.fs.setWriteChecksum(true);
60          item.fs.forcedReplication(item, params);
61      }
62  }
```

ForcedReplicationParams.java

```
1   package org.apache.hadoop.fs;
2
3   import org.apache.hadoop.classification.InterfaceAudience;
4   import org.apache.hadoop.classification.InterfaceStability;
5
6   @InterfaceAudience.Public
7   @InterfaceStability.Evolving
8   public class ForcedReplicationParams {
9       private long blockId;
10      private String srcDatanode;
11      private String dstDatanode;
12
```

```
    public ForcedReplicationParams(long blockId, String srcDatanode, String dstDatanode) {
        this.blockId = blockId;
        this.srcDatanode = srcDatanode;
        this.dstDatanode = dstDatanode;
    }

    public long getBlockId() {
        return blockId;
    }

    public String getSrcDatanode() {
        return srcDatanode;
    }

    public String getDstDatanode() {
        return dstDatanode;
    }

    @Override
    public String toString() {
        return "ForcedReplicationParams-" + blockId + "-" + srcDatanode + "->" + dstDatanode;
    }

}
```

DistributedFileSystem.java

```
  public boolean forcedReplication(PathData f, final ForcedReplicationParams params) throws IOException {
    DatanodeStorageReport[] storageReport = dfs.namenode.getDatanodeStorageReport(DatanodeReportType.LIVE);
    LocatedBlocks blocks = dfs.namenode.getBlockLocations(f.toString(), 0, Long.MAX_VALUE);

    LocatedBlock forcedBlock = null;
    DatanodeInfo forcedSrcDatanode = null;
    DatanodeInfo forcedDstDatanode = null;

    for (LocatedBlock block: blocks.getLocatedBlocks())
      if (block.getBlock().getBlockId() == params.getBlockId())
        forcedBlock = block;
    if (forcedBlock == null) {
      DFSClient.LOG.info("Wrong blockID");
      return false;
    }

    for (DatanodeStorageReport report: storageReport) {
      DatanodeInfo datanodeInfo = report.getDatanodeInfo();
```

```
19        if (datanodeInfo.getXferAddr().equals(params.getSrcDatanode()))
20          forcedSrcDatanode = datanodeInfo;
21        if (datanodeInfo.getXferAddr().equals(params.getDstDatanode()))
22          forcedDstDatanode = datanodeInfo;
23      }
24
25      if (forcedSrcDatanode == null || forcedDstDatanode == null ||
    ↪  forcedSrcDatanode == forcedDstDatanode) {
26        DFSClient.LOG.info("Wrong datanodes");
27        return false;
28      }
29
30      DFSClient.LOG.info(f.toString() + ": " + forcedSrcDatanode + "->" +
    ↪  forcedDstDatanode);
31      ForcedMover mover = new ForcedMover(this, f.toString());
32      boolean result = mover.scheduleMovingBlock(forcedBlock, forcedSrcDatanode,
    ↪  forcedDstDatanode);
33      return result;
34    }
```

ForcedMover.java

```
1   package org.apache.hadoop.hdfs.server.mover;
2
3   import com.google.common.collect.Maps;
4   import org.apache.hadoop.conf.Configuration;
5   import org.apache.hadoop.fs.FSDataOutputStream;
6   import org.apache.hadoop.fs.Path;
7   import org.apache.hadoop.fs.StorageType;
8   import org.apache.hadoop.hdfs.*;
9   import org.apache.hadoop.hdfs.protocol.DatanodeInfo;
10  import org.apache.hadoop.hdfs.protocol.LocatedBlock;
11  import org.apache.hadoop.hdfs.protocol.LocatedBlocks;
12  import org.apache.hadoop.hdfs.server.balancer.Dispatcher.DBlock;
13  import org.apache.hadoop.hdfs.server.balancer.NameNodeConnector;
14  import org.apache.hadoop.hdfs.server.mover.Mover.MLocation;
15
16  import java.io.IOException;
17  import java.net.URI;
18  import java.util.*;
19  import java.util.concurrent.atomic.AtomicInteger;
20
21  public class ForcedMover {
22
23    private static DistributedFileSystem dfs;
24    private static String file;
25
26    public ForcedMover(DistributedFileSystem dfs, String file) {
27      this.dfs = dfs;
28      this.file = file;
```

```
29    }
30
31    static Mover newMover(Configuration conf) throws IOException {
32      final Collection<URI> namenodes = DFSUtil.getNsServiceRpcUris(conf);
33      Map<URI, List<Path>> nnMap = Maps.newHashMap();
34      for (URI nn : namenodes) {
35        nnMap.put(nn, null);
36      }
37
38      final List<NameNodeConnector> nncs = NameNodeConnector.newNameNodeConnectors(
39          nnMap, Mover.class.getSimpleName(), Mover.MOVER_ID_PATH, conf,
40          NameNodeConnector.DEFAULT_MAX_IDLE_ITERATIONS);
41      return new Mover(nncs.get(0), conf, new AtomicInteger(0));
42    }
43
44
45    public static boolean scheduleMovingBlock(LocatedBlock block, DatanodeInfo
    ↪  srcDatanode, DatanodeInfo dstDatanode)
46          throws IOException {
47      dfs.LOG.info("start TEST");
48      final Configuration conf = new HdfsConfiguration();
49
50      final Mover mover = newMover(conf);
51      mover.init();
52      final Mover.Processor processor = mover.new Processor();
53
54      final LocatedBlock lb = block;
55      final List<MLocation> locations = MLocation.toLocations(lb);
56      final MLocation mlS = new MLocation(srcDatanode, lb.getStorageTypes()[0],
    ↪  lb.getBlockSize());
57      final MLocation ml = new MLocation(dstDatanode, lb.getStorageTypes()[0],
    ↪  lb.getBlockSize());
58      final DBlock db = mover.newDBlock(lb.getBlock().getLocalBlock(), locations);
59
60      final List<StorageType> storageTypes = new ArrayList<StorageType>(
61          Arrays.asList(StorageType.DEFAULT, StorageType.DEFAULT));
62      processor.scheduleMoveForcedReplica(db, ml, mlS, lb.getStorageTypes()[0]);
63      dfs.LOG.info("finish TEST");
64      return true;
65    }
66  }
```

List of Figures

Bibliography

[1] *Apache Hadoop (MapReduce) Internals - Diagrams*. http://ercoppa.github.io/HadoopInternals/.

[2] David Bollier. *The Promise and Peril of Big Data*. Tech. rep. The Aspen Institute, 2010.

[3] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Commun. ACM* 51.1 (2008), pp. 107–113. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[4] Apache Software Foundation. *Apache Hadoop*. URL: http://hadoop.apache.org/docs/current/.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*. 2003, pp. 29–43. DOI: 10.1145/945445.945450. URL: http://doi.acm.org/10.1145/945445.945450.

[6] Zhenhua Guo, Geoffrey Fox, and Mo Zhou. "Investigation of Data Locality in MapReduce". In: *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13-16, 2012*. 2012, pp. 419–426. DOI: 10.1109/CCGrid.2012.42. URL: http://dx.doi.org/10.1109/CCGrid.2012.42.

[7] Derrick Harris. *The history of Hadoop: From 4 nodes to the future of data*. URL: https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/.

[8] Brad Hedlund. *Understanding Hadoop Clusters and the Network*. Sept. 2011. URL: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/.

[9] Herodotos Herodotou. "Hadoop Performance Models". In: *CoRR* abs/1106.0940 (2011). URL: http://arxiv.org/abs/1106.0940.

[10] Petar Jovanovic et al. "H-WorD: Supporting Job Scheduling in Hadoop with Workload-driven Data Redistribution". In: *Advances in Databases and Information Systems - 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28 – 31, 2016, Proceedings*. 2016.

[11] Vasiliki Kalavri and Vladimir Vlassov. "MapReduce: Limitations, Optimizations and Open Issues". In: *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International*

Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013. 2013, pp. 1031–1038. DOI: 10.1109/TrustCom.2013.126. URL: http://dx.doi.org/10.1109/TrustCom.2013.126.

[12] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. META Group, Feb. 2001. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[13] James Manyika et al. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. Tech. rep. McKinsey Global Institute, June 2011.

[14] Bernard Marr. *Why only one of the 5 Vs of big data really matters*. Mar. 2015. URL: http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters.

[15] Michael G. Noll. *Running Hadoop on Ubuntu Linux (Multi-Node Cluster)*. URL: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/.

[16] Michael G. Noll. *Running Hadoop on Ubuntu Linux (Single-Node Cluster)*. URL: http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/.

[17] Kevin Normandeau. *Beyond Volume, Variety and Velocity is the Issue of Big Data Veracity*. Sept. 2013. URL: http://insidebigdata.com/2013/09/12/beyond-volume-variety-velocity-issue-big-data-veracity/.

[18] Mark van Rijmenam. *Why The 3V's Are Not Sufficient To Describe Big Data*. Aug. 2007. URL: https://datafloq.com/read/3vs-sufficient-describe-big-data/166.

[19] Konstantin Shvachko et al. "The Hadoop Distributed File System". In: *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*. 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972. URL: http://dx.doi.org/10.1109/MSST.2010.5496972.

[20] Vinod Kumar Vavilapalli et al. "Apache Hadoop YARN: yet another resource negotiator". In: (2013), 5:1–5:16. DOI: 10.1145/2523616.2523633. URL: http://doi.acm.org/10.1145/2523616.2523633.

[21] Weina Wang et al. "MapTask Scheduling in MapReduce With Data Locality: Throughput and Heavy-Traffic Optimality". In: vol. 24. 1. 2016, pp. 190–203. DOI: 10.1109/TNET.2014.2362745. URL: http://dx.doi.org/10.1109/TNET.2014.2362745.