



College of Engineering, Design and Physical Science
Electronic and Computer Engineering

Assignment Java Testing and Measuring

Distributed Computing Systems Engineering Msc

Author: Christoph Gschrey

Lab-Partner: Matthias Gebert

Date: 22. September 2017

Supervisor: Prof. Dr. Peter Väterlein

ASSIGNMENT SUBMISSION FORM

Please note: that no course work will be accepted without this cover sheet.

Please ensure: that you keep a copy of work submitted and retain your receipt in case of query.

Student Number:	SPO ID Number (Office use only):	
Course:		Level:

MODULE	
Module Code:	Module Title:
Lab / Assignment:	Deadline:
Lab group (if applicable):	Date Stamp (Office use only):
Academic Responsible:	
Administrator:	

Please note: that detailed feedback will be provided on a feedback form.

✂.....

RECEIPT SECTION (Office Copy)	
Student Number:	SPO ID Number (Office use only):
Student First Name:	Student Last Name:
Module Code:	Module Title:
Lab / Assignment:	
Lab group (if applicable):	Deadline:
Academic Responsible:	Number of Days late:

DECLARATION	
I have read and I understand the guidelines on plagiarism and cheating in the Handbook and I certify that my contribution to this report fully complies with these guidelines. I confirm that I have kept a copy of my work and that I have not lent my work to any other students.	
Signed:	Date Stamp (Office use only):

✂.....

RECEIPT SECTION (Student Copy)	
Student Number:	Student Name:
Lab / Assignment:	
Lab group (if applicable):	Module Title:
Academic Responsible:	Deadline:
Module Code:	Date Stamp (Office use only):

The University penalty system will be applied to any work submitted late.

IMPORTANT: You **MUST** keep this receipt in a safe place as you may be asked to produce it at any time as proof of submission of the assignment. Please submit this form with the assignment attached to the Department of Design Education Office in the Michael Sterling Building, room MCST 055.

Contents

1	Introduction	1
2	High Performance Computing With MapReduce and Hadoop	2
2.1	MapReduce	2
2.2	Hadoop	3
2.2.1	Hadoop Distributed File System (HDFS)	4
3	Hadoop's Parameters	6
4	HSim	8
5	Evaluation	11
5.1	Changing the dataSize	11
5.2	Changing the sortFactor	12
5.3	Changing the requiredMappers and requiredReducers	12
6	Conclusion	14
	Bibliography	15

1 Introduction

As the last few years and decades have seen ever-increasing amounts of data analyzed and managed in ever shorter periods of time, a challenge faced by the IT industry has been the limited computing power of individual machines and simple networks. Tasks are constantly evolving, so they need more and more computing power to solve them.

In order to solve these problems, various techniques have been developed that help to continuously improve the performance of the machines. These include supercomputers, computer clusters, and various methods and algorithms that can be summarized under the term High Performance Computing.

This workshop introduces the programming model **MapReduce** and a simulation of the model based on the **Hadoop** framework. The aim of this workshop was to use the Hadoop Simulation HSim to gain a basic understanding of the functionality of MapReduce.

2 High Performance Computing With MapReduce and Hadoop

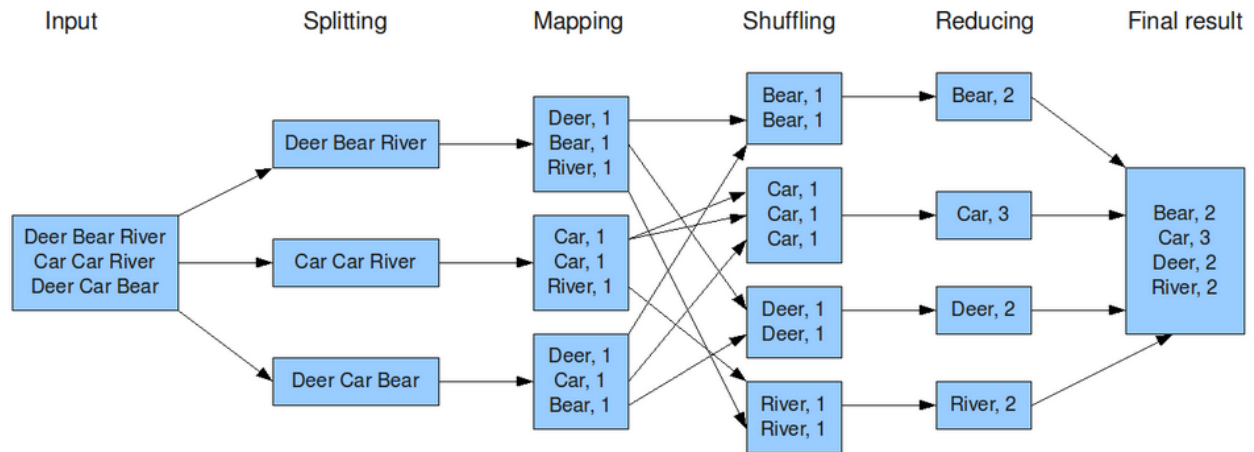
As already mentioned in the introduction, MapReduce was introduced as a programming model to process large amounts of data (Big Data) in parallel on several machines. This reduces processing time, since the load can be distributed over multiple machines.

The following chapters briefly explain the basics of this workshop. First, MapReduce and its functionality are described. This is followed by a short introduction to Hadoop.

2.1 MapReduce

Mapreduce is a programming model that addresses the processing and analysis of large amounts of data using a distributed, parallelized approach [Dg0]. The data to be processed is divided into several data blocks of the same size. These data blocks are processed in parallel and independently of each other. These different blocks are the inputs for a **mapping** function that performs filtering and sorting [Bi0]. Each block receives a unique key. The resulting key value pairs now enter a process called "**shuffling**". Therefore, the key value pairs are assigned to the correct nodes that are defined by the key. In other words, the shuffling process sorts the data from the separate mapping blocks into new blocks according to the key. Each block contains pairs with the same key but different values. These are required for the next to last step of the MapReduce process: **reducing**. The reduction phase cannot begin until the mapping phase is complete. In this phase, each value of a block is combined to its key so that you get a new key value pair, where the new value is a list of the old values. The last step is to combine all the pairs in a data block. This is the final result of the MapReduce process [Whi15]. Large amounts of data are thus divided into much smaller blocks, which are processed in parallel and independently from each other and finally reassembled. MapReduce simplifies and accelerates parallel processing and analysis of large amounts of data and is therefore very popular in the big data area. Figure 2.1 shows an example with all required steps for the MapReduce programming model with simple datasets

There are several implementations of the MapReduce programming model. One of these is described in the next chapter.

Figure 2.1: The MapReduce model¹

2.2 Hadoop

Apache Hadoop is a framework that allows distributed processing of large amounts of data across clusters of computers with simple programming models. It supports scalability from individual servers to thousands of computers, each with local calculations and storage [Apa17]. Hadoop is based on Java and consists of four main components:

- Hadoop Common
- Hadoop Distributed File System (HDFS)
- Google's MapReduce-algorithm
- Yet Another Resource Negotiator (YARN)

Hadoop Common provides the basic functions and tools for the other modules of the software. The HDFS will be described in the next sub-section. The MapReduce-algorithm is based on the MapReduce programming Model (s. Chapter 2.1) and offers the same functionality. The Yet Another Resource Negotiator (YARN) can manage the resources in a computer cluster and dynamically allocate the resources of a cluster to different jobs. YARN uses queues to determine the capacities of the systems for the individual tasks [Lit16].

Hadoop uses a master-slave architecture to distribute the data to be processed on several machines inside a cluster. A client transfers a job to the master node that is connected to the slaves in the cluster. The JobTracker of the master node controls the MapReduce job and reports it to a TaskTracker of a slave node. Within a slave node there are one or more data

¹image-source: <https://cs.calvin.edu/courses/cs/374/exercises/12/lab>

nodes as well as one or more Mappers and Reducers, which execute the MapReduce algorithm. In the event of an error, the JobTracker of the master node will reschedule the task to the same or a different slave node, depending on what is most efficient. This makes Hadoop very fault-tolerant and ensures that the entire process does not have to be aborted and restarted in the event of an error. [Tec12]

However, this alone is not enough to prevent the process from being terminated. If a master-node fails, all MapReduce processes that are processed by its associated slaves are also affected. Therefore, the master defines checkpoints after completion of single processing steps. In the event of an error, another master node can then continue where the failed master stopped and the process does not have to be rolled back completely. Figure 2.2 shows the Hadoop's Master-Slave architecture:

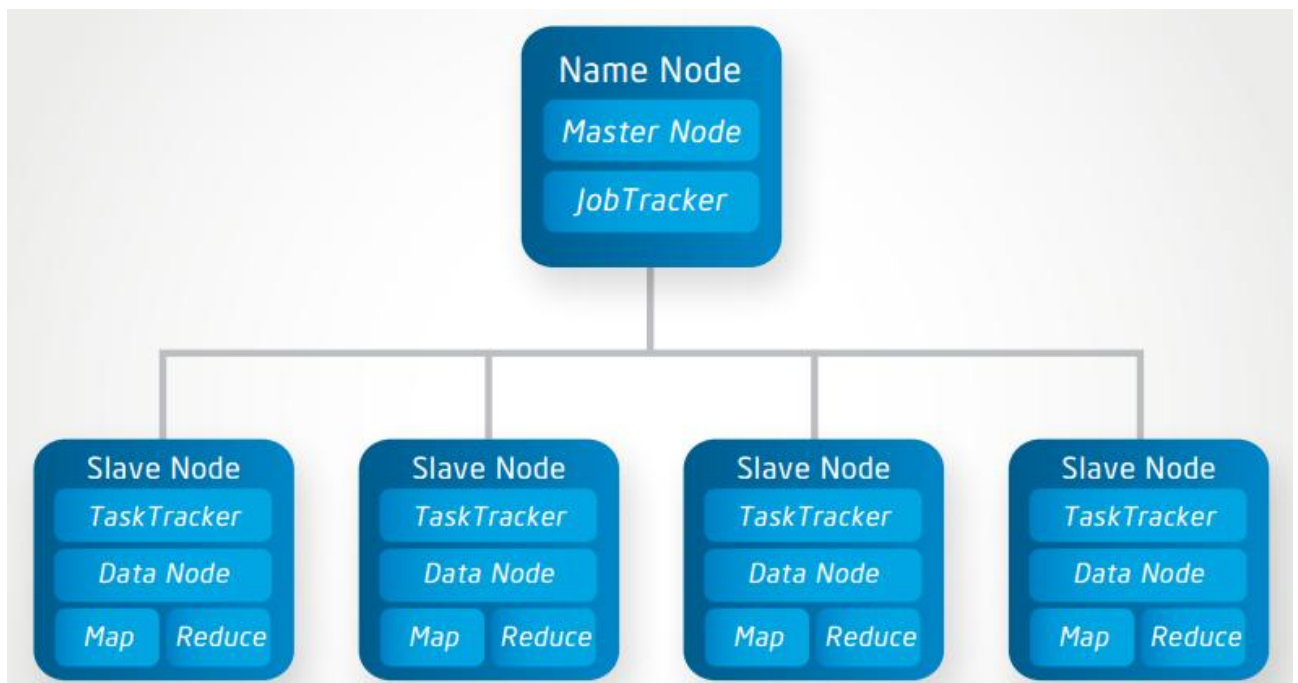


Figure 2.2: The Hadoop Master-Slave Architecture²

2.2.1 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications [see Rou18]. It is a distributed file system that handles large data sets running on commodity hardware. HDFS splits the input data into data blocks and distributes them to independent nodes so that the data can be processed in parallel. For fault tolerance reasons, HDFS copies each piece of data several times and sends these copies to different racks

²image-source: <http://www.rosebt.com/blog/hadooparchitecture-and-deployment>

(**replication**). This ensures the accessibility to the required data elsewhere in this cluster if a job node fails.

The structure consists of a name node and several data nodes according to the master-slave principle. The name node thus assumes the role of the master and is responsible for managing and coordinating the data in the clusters. The name-node contains metadata including the data replicas, checkpoints (s. Chapter 2.2) and other important information, while the data nodes contain the actual data to be processed. Figure 2.3 shows the Hadoop Distributed File System Architecture and the ongoing read and write processes to the file system.

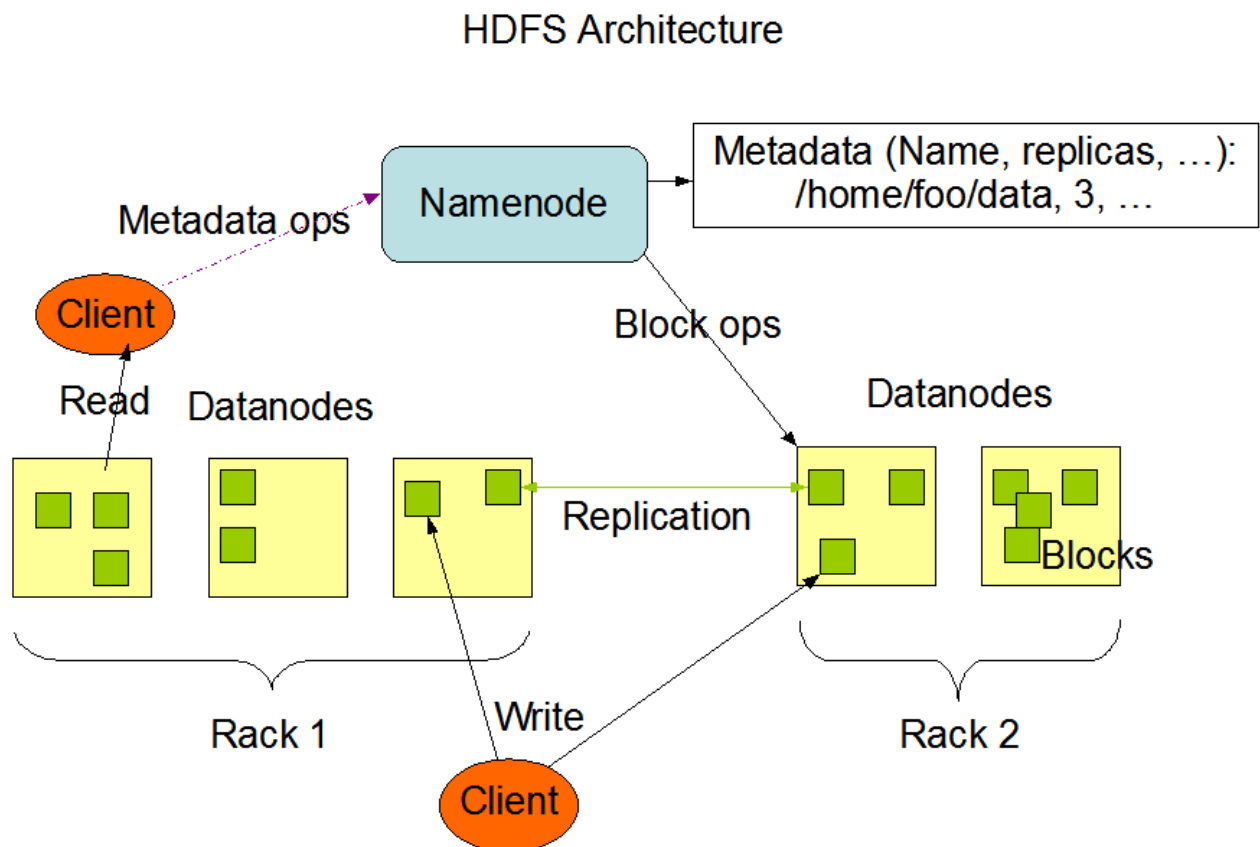


Figure 2.3: The Hadoop Distributed File System Architecture³

³image-source: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

3 Hadoop's Parameters

Before starting the simulation of Hadoop in the next chapter, a closer look is taken at the parameters of Hadoop, which can be manipulated in the simulation to get different results. Table ?? shows an overview over the most important Hadoop parameter:

Configuration Parameters	Default Values
io.sort.factor	10
io.sort.mb	100
io.sort.spill.percent	0.8
mapred.reduce.tasks	1
mapreduce.tasktracker.map.tasks.maximum	2
mapreduce.tasktracker.reduce.tasks.maximum	2
mapred.child.java.opts	200
mapreduce.reduce.shuffle.input.buffer.percent	0.7
mapred.reduceparallel.copies	5
mapred.compress.map.output	False
mapred.output.compress	False

Table 3.1: Most important Hadoop parameters

Each of these parameter can be changed to change the way Hadoop is working on large datasets. The following descriptions for each parameter are directly taken from Apache's official documentation⁴:

io.sort.factor: This is the number of streams Hadoop uses to sort the files in parallel. Keep note that this determines the number of open files that have to be handled.

io.sort.mb: This is the total amount of buffer memory Hadoop sorts the files in MB. The default value per merge stream accounts for 1MB, which should minimize seeks. **io.sort.spill.percent:** The soft limit in either the buffer or record collection buffers. Once reached, a thread will begin to spill the contents to disk in the background. Note that this does not imply any chunking of data to the spill. A value less than 0.5 is not recommended.

mapred.reduce.tasks: The default number of reduce tasks per job. Typically set to 99% of the cluster's reduce capacity, so that if a node fails the reduces can still be executed in a single wave. Ignored when mapred.job.tracker is "local".

⁴<https://hadoop.apache.org/docs/r1.0.4/mapred-default.html>

mapreduce.tasktracker.map.tasks.maximum: The maximum number of map tasks that will be run simultaneously by a task tracker.

mapreduce.tasktracker.reduce.tasks.maximum: The maximum number of reduce tasks that will be run simultaneously by a task tracker.

mapred.child.java.opts: Java opts for the task tracker child processes. The following symbol, if present, will be interpolated: @taskid@ is replaced by current TaskID. Any other occurrences of '@' will go unchanged. For example, to enable verbose gc logging to a file named for the taskid in /tmp and to set the heap maximum to be a gigabyte, pass a 'value' of: -Xmx1024m-verbose:gc -Xloggc:/tmp/@taskid@.gc The configuration variable mapred.child.ulimit can be used to control the maximum virtual memory of the child processes.

mapreduce.reduce.shuffle.input.buffer.percent: The percentage of memory to be allocated from the maximum heap size to storing map outputs during the shuffle.

mapred.reduceparallel.copies: The default number of parallel transfers run by reduce during the copy(shuffle) phase.

mapred.compress.map.output: Should the outputs of the maps be compressed before being sent across the network. Uses SequenceFile compression.

mapred.output.compress: Should the job outputs be compressed?

These are just a few selected parameters that can be changed to influence Hadoop's runtime behavior. The complete list can be found here: [AP03]. Now that the theoretical basics have been sufficiently explained and the parameters for the simulation described, it is time to move on to the actual topic of this assignment. The next chapter will introduce HSim and show how to change the parameters listed above to achieve different results in the simulation.

4 HSim

As already mentioned in the introduction, no complete cluster is being set up for this workshop, as the effort would have been far too much. Instead the provided software *HSim* was used, which provides a simulated Hadoop environment with several machines, routers and nodes. *HSim* is based on Java and Hadoop v2 and it offers the possibility to adjust different Hadoop parameters and to display the effects on the simulation realistically. Figure 4 shows *HSim*'s main user interface while running a simulation.

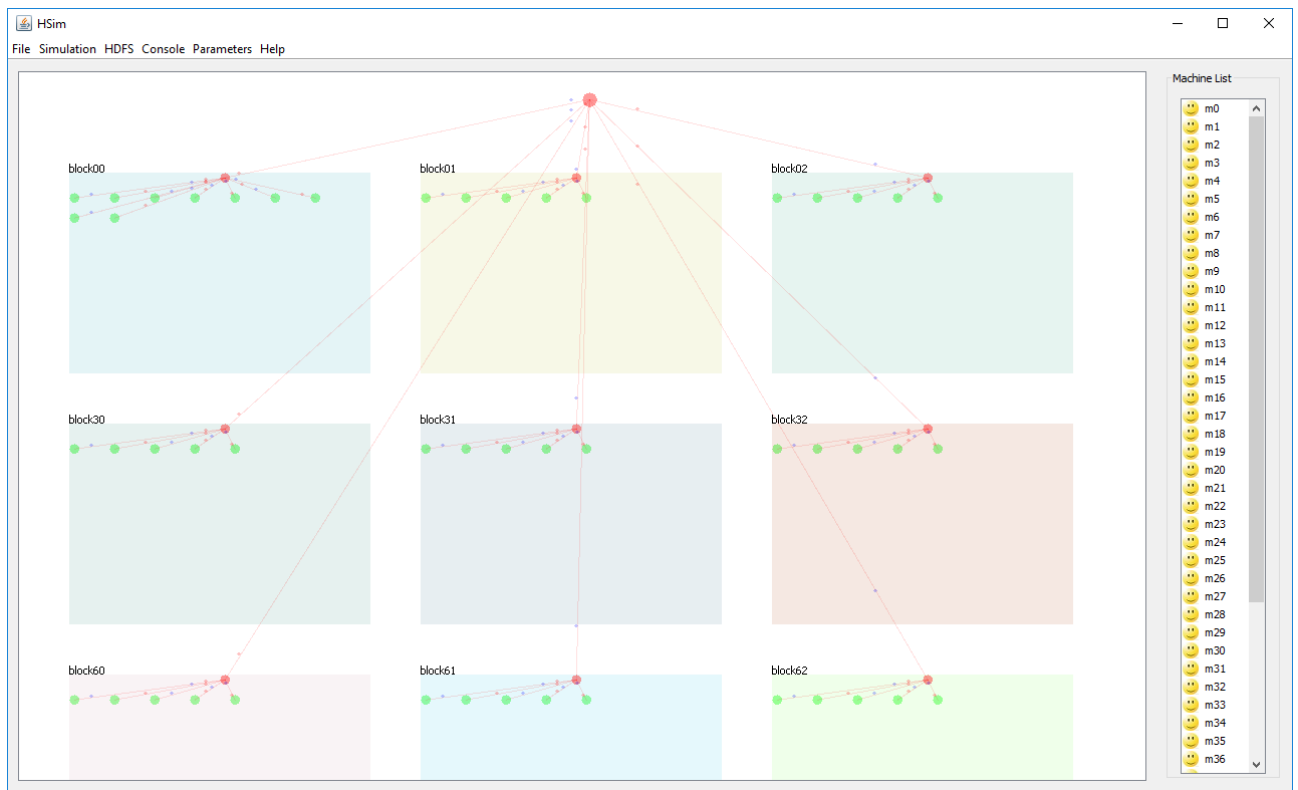


Figure 4.1: Screenshot of *HSim*'s User Interface during a running Simulation

The rectangles in different colors represent the individual racks of the simulated Hadoop cluster. Each rack has multiple machines that are displayed as green dots. The red dots, of which each rack has one, are the routers. The red dot at the top edge is the master node, which splits the input data and distributes it to the routers of the racks. On the right hand side there is a list of all machines. The next figure 4.2 shows the machine state window that appears when double clicking on one of the nodes.

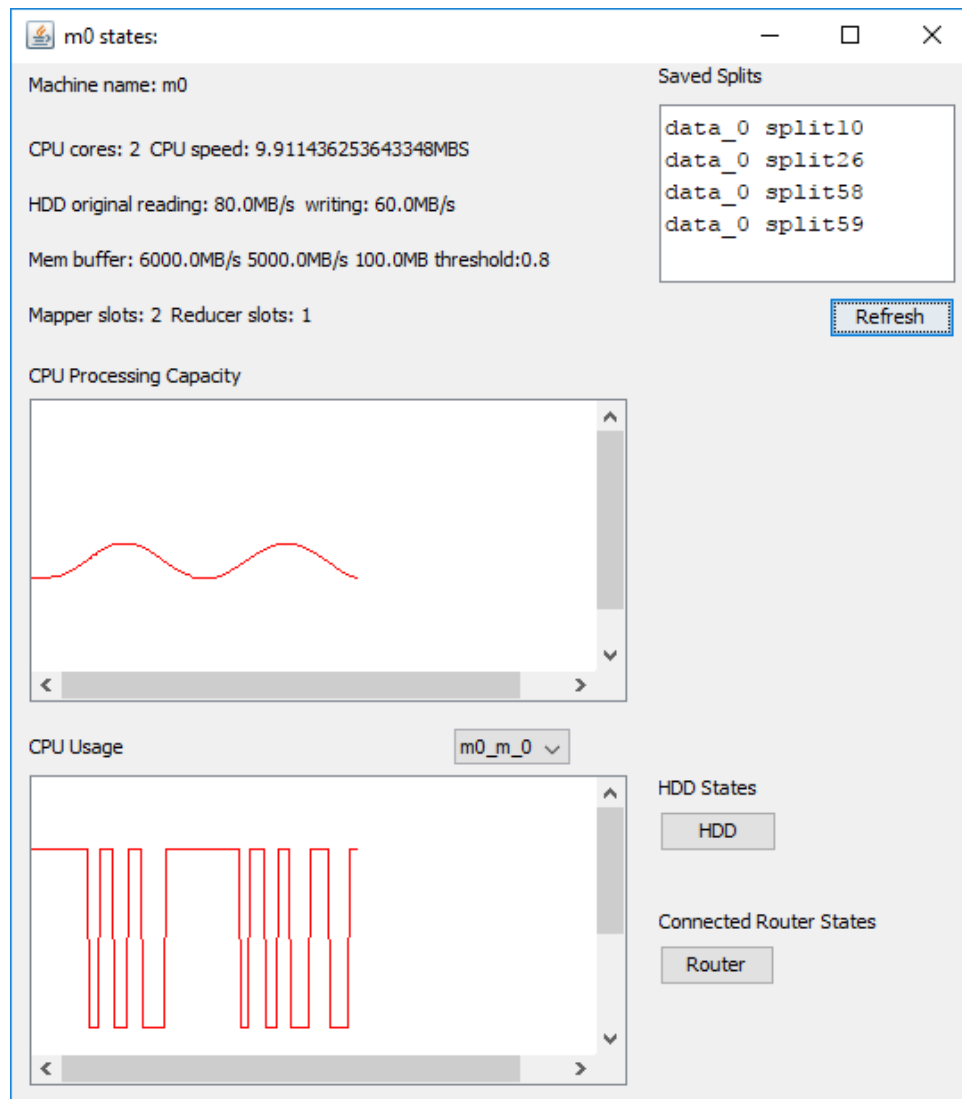


Figure 4.2: Screenshot of the first machine's state window during a running Simulation

The performance of the individual machines can be observed here at the runtime of the simulation. The next figure 4.3 shows the console that can be opened from HSim's main UI (see Figure 4.1).

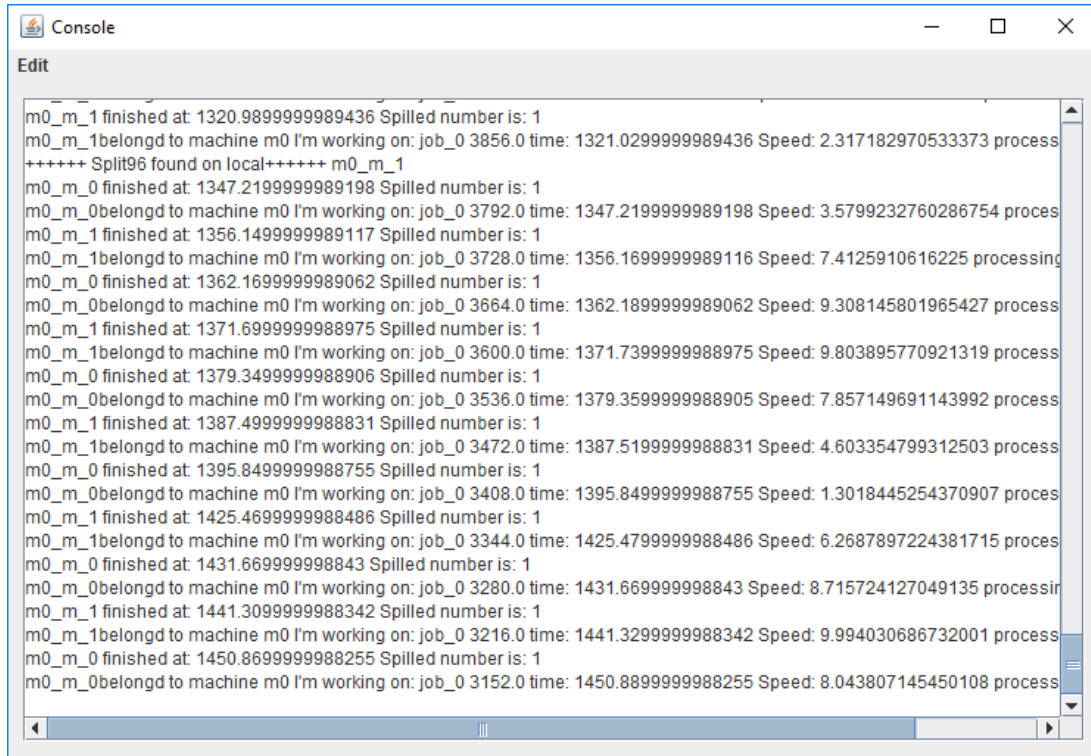


Figure 4.3: Screenshot of HSim's Console during a running Simulation

From here, the individual processes for the individual jobs running on the various machines can be tracked at runtime. The runtimes required by the individual machines for a job are measured and displayed.

The different Hadoop parameters (see chapter 3) can be set using a configuration file. For the following simulation, the presetting with a Hadoop cluster of 49 individual machines was used. In the next chapter, the influence of four different Hadoop parameters on the runtime of the simulation will be investigated.

5 Evaluation

5.1 Changing the dataSize

Data Size	Runtime for Upload	Runtime for Mapping	Runtime for Job Execution	Total Runtime
1000	25,253	66,275	80,638	172,166
2000	50,126	134,22	157,284	341,63
4000	97,97333333	213,7944444	260,3766667	572,1444444
6000	146,288	305,504	381,654	833,446
8000	194,306	387,245	469,059	1050,61
10000	244,437	491,713	585,049	1321,199

Figure 5.1: DataSize/RunTime table

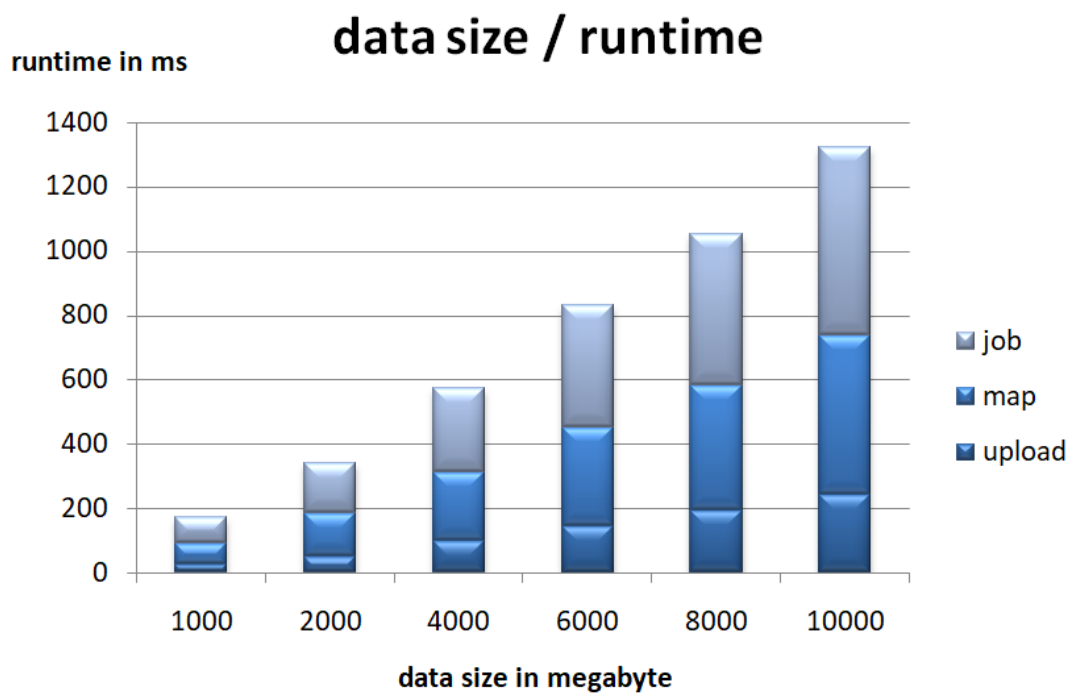


Figure 5.2: DataSize/RunTime chart

5.2 Changing the sortFactor

Sort Factor	Runtime for Upload	Runtime for Mapping	Runtime for Job Execution	Total Runtime
2	243,995	490,107	521,518	1255,62
4	244,552	490,874	520,539	1255,965
6	243,725	489,469	584,025	1317,219
8	243,85	494,744	589,945	1328,539
10	244,626	490,13	584,699	1319,455

Figure 5.3: Sortfactor/RunTime table

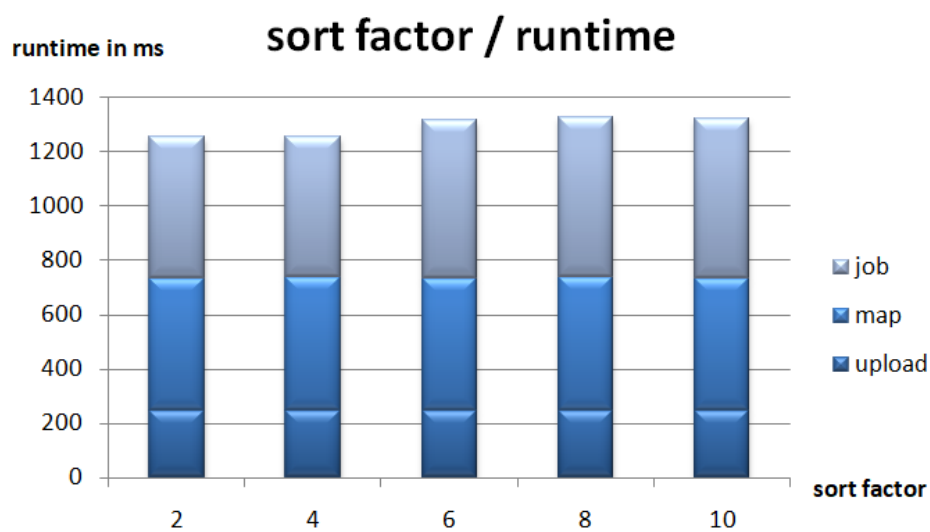


Figure 5.4: Sortfactor/RunTime chart

5.3 Changing the requiredMappers and requiredReducers

Mappers	Reducers	Runtime for Upload	Runtime for Mapping	Runtime for Job Execution	Total Runtime
2	1	248,892	1960,489	2243,065	4452,446
4	2	246,275	1066,72	1140,888	2453,883
6	3	244,157	790,193	823,579	1857,929
8	4	243,653	660,776	674,94	1579,369
10	5	244,142	608,608	635,875	1488,625
12	6	244,124	572,157	642,629	1458,91
14	7	244,035	529,861	611,206	1385,102
16	8	243,96	514,104	601,053	1359,117
18	9	243,907	497,659	589,805	1331,371
20	10	243,375	492,929	585,462	1321,766

Figure 5.5: Number of Mappers and number of Reducers/RunTime table

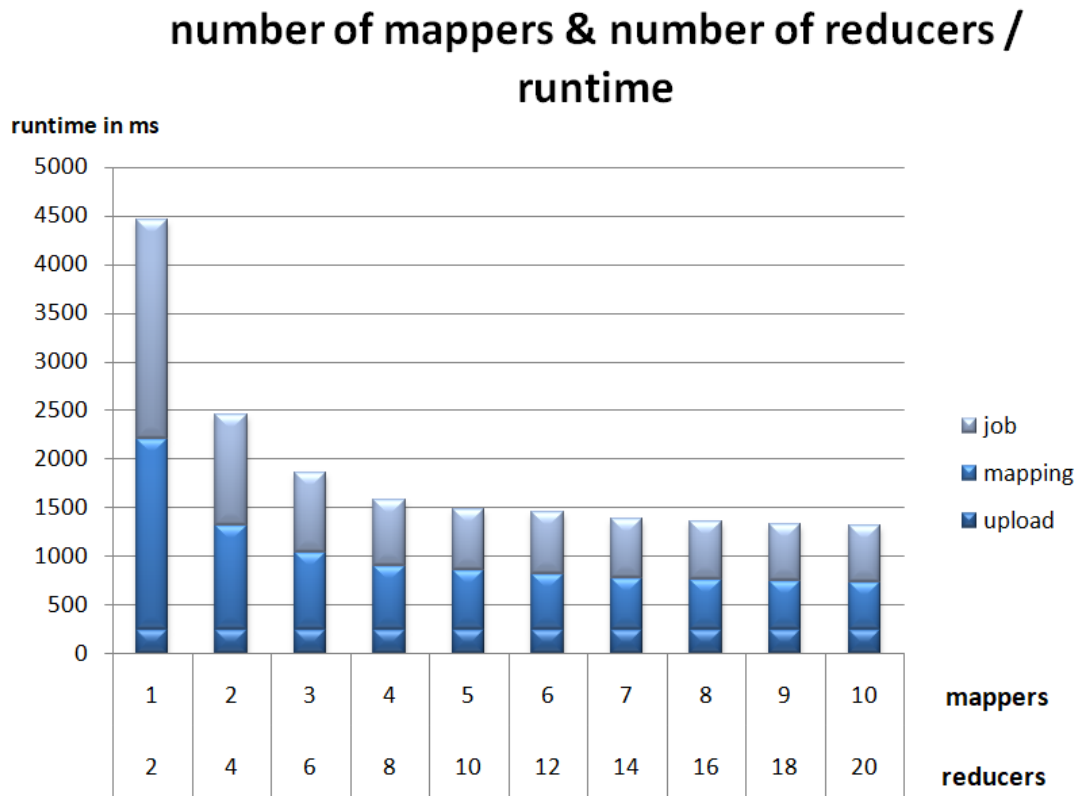


Figure 5.6: Number of Mappers and number of Reducers/RunTime chart

6 Conclusion

Bibliography

- [Dg0] *MapReduce: Simplified Data Processing on Large Clusters*. 2004. URL: <http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf> (cit. on p. 2).
- [Bi0] *Big-Data-Technologien - Wissen for Entscheider*. URL: <https://www.bitkom.org/noindex/Publikationen/2014/Leitfaden/Big-Data-Technologien-Wissen-fuer-Entscheider/140228-Big-Data-Technologien-Wissen-fuer-Entscheider.pdf> (cit. on p. 2).
- [Whi15] Tom White. “Hadoop: The Definitive Guide”. In: *Hadoop: The Definitive Guide*. Cambridge: O’Reilly, 2015. URL: http://javaarm.com/file/apache/Hadoop/books/Hadoop-The.Definitive.Guide_4.edition_a_Tom.White_April-2015.pdf (visited on 03/03/2018) (cit. on p. 2).
- [Apa17] Apache. “Hadoop Official Documentation”. In: *Hadoop Official Documentation*. The Apache Software Foundation, 2017. URL: <http://hadoop.apache.org/> (visited on 03/03/2018) (cit. on p. 3).
- [Lit16] Nico Litzel. “Definition: What is Hadoop?”. In: *Big Data Insider*. Vogel Business Media, 2016. URL: <https://www.bigdata-insider.de/was-ist-hadoop-a-587448/> (visited on 03/03/2018) (cit. on p. 3).
- [Tec12] Rose Business Technologies. “Hadoop Architecture and Deployment”. In: New York, NY, 10020: Rose Business Technologies, 2012. URL: <http://www.rosebt.com/blog/hadooparchitecture-and-deployment> (visited on 03/03/2018) (cit. on p. 4).
- [Rou18] Margaret Rouse. “Hadoop Distributed File System (HDFS)”. In: 2018. URL: <http://www.rosebt.com/blog/hadooparchitecture-and-deployment> (visited on 03/04/2018) (cit. on p. 4).