Hadoop in the browser

Simulating MapReduce in the browser

H. Bilen & M. Zwart

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Hadoop in the browser

Simulating MapReduce in the browser

by

H. Bilen & M. Zwart

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,

Supervisor:	dr. T. Abeel
Client:	dr. C. Hauff
Bachelor coordinator:	O. Visser

TU Delft Delft University of Technology

Preface

This report details the development and research done during the 'Hadoop in the browser' project for the course: TI3806 Bachelorproject. It was commissioned by Assistant Professor Claudia Hauff from the Web Information Systems group (WIS) at the faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at the Delft University of Technology.
The report documents the 10 weeks that were spent researching the most optimal solution and the development of this solution, Trifle. Trifle is a web based platform for practicing with toy MapReduce problems to improve the quality of teaching in the Big Data Processing course in the Computer Science curriculum. It allows for dynamic lectures, has high accessibility and user testing indicated that it in fact does improve the understanding of MapReduce. The goal of the report is to show the reader how the application works from both a student and lecturer perspective, which design choices were made and how we recommend the system to be deployed.
We would like to thank dr. Thomas Abeel for his regular and constructive feedback on our report, dr. Claudia Hauff for offering us this unique opportunity and attending our user test session. Lastly special thanks go out to our user test group who offered their time to help us test the platform and get some valuable feedback from them.

H. Bilen & M. Zwart
Delft, June 2016

Abstract

With the current increase of user generated data, the need for tools to process large quantities of data is increasing. One of the courses of the Computer Science BSc curriculum is the Big Data Processing course. The Big Data Processing course teaches students ways of doing so. A popular and teached method is using MapReduce, a programming model to process large quantities of data.

Big Data Processing students currently write their implementations for MapReduce related assignments of the lab in the Cloudera Virtual Machine (VM). This VM is slow, cannot be used interactively and it cannot be used to teach all MapReduce principles like memory separation.

Since there are no existing solutions that solves these problems we have decided to write our own. Writing our own solution required diverse knowledge including but not limited to JavaScript, ECMAScript 6, WebWorkers, HTML5 (&CSS), PHP, MySQL, API development, Wordpress, Linux, QUnit and Selenium. Our solution, Trifle, overcomes previously mentioned and other shortcomings. Trifle is a web-based solution that simulates MapReduce within the browser. The framework uses JavaScript together with WebWorkers and our front-end is written using Wordpress. By simulating MapReduce within the browser we managed to create a solution that requires no cluster, is easy to use, works multi-platform and most importantly enables lecturers to teach some MapReduce principles that could not be taught before. Furthermore, we integrated a submissions system that can be used to do interactive lectures in which the lecturer can see problems in real-time and explain obstacles. A user test we have done verifies that Trifle helps to better grasp the idea behind MapReduce.

Contents

1

Context

This section provides context relevant to our research. We give an introduction to the Big Data Processing course (BDP) and its main teaching goals. Furthermore, we give an overview of tools currently used to teach the MapReduce paradigm.

1.1. About the Big Data Processing course

The course focuses on teaching the students the principles of processing Big Data and discovering patterns in the (semi) unstructured data. The goals of this course include: explaining the difference between OO-programming and functional programming, explaining the major components of the Hadoop framework, demonstrating the interplay between the different Hadoop components, creating Hadoop-based algorithms for novel (unseen) practical problems and analyze MapReduce algorithms for their feasibility in practice [1]. The BDP course introduces methods of processing such as streaming and the MapReduce paradigm. A big part of this course is teaching the students the principles of MapReduce in the Hadoop framework [1, 2], which we elaborate on in the next subsection.

1.2. MapReduce

MapReduce is a programming model developed by Google that provides a simple API for the distributed processing of massive amounts of data in a cluster [3]. MapReduce handles the distribution of the data that needs to be processed so the developers can focus on the processing of the data. This way the same code can run on a single machine or a cluster consisting of 20,000 machines. For a simple wordcount of a dataset, just two methods have to be written: map and reduce. The developer implements a mapper that splits a given dataset into key-value pairs. The MapReduce framework then splits the tasks such that the values of identical keys are given to a single reducer. The reducers output the final result back to the framework which in place will write it to the filesystem.

1.3. Hadoop

Hadoop is an open-source implementation of the MapReduce programming model used by many big corporations for the distributed processing of large datasets in a cluster [4].

We provide a typical wordcount example in the MapReduce model below:

- The framework executes the mapper for each line with the document id as key and line as value.
- The mapper emits each word in the line as key and value 1 (ex. {dog, 1}).
- The MapReduce framework combines the values of each key and distributes the keys over reducers (one reducer might receive {dog, [1, 1, 1]}).
- The reducer sums up the values of each key and outputs the key together with the sum of the values (one reducer might output {dog, 3}).
- The output of the framework are file(s) with each word and their frequency (the output of a reducer might be [{dog, 3}, {cat, 2}]).

1.4. Current MapReduce teaching methods

During the BDP course, the professor explains concepts and gives examples of MapReduce algorithms of simple problems such as the earlier mentioned WordCount example. Given this knowledge, students implement more advanced problems in a Cloudera VM that is provided. Cloudera VM is a Virtual Machine Image based on CentOS that has Hadoop pre-installed. Students of the BDP-course install this VM and can start writing the required code after a setup (like adding the input files for the lab assignment to this machine) [5]. After implementation solutions are sent to the teacher. The only way students are actually interacting with the principles of MapReduce is by working in this provided environment. However this environment has a few flaws as a learning environment. In the next chapter we will elaborate on this.

2

Problem definition

Our client, the BDP-course lecturer, would like to have a tool to teach the principles of MapReduce during a lecture but the only option at hand would be the Cloudera VM that students already use for the lab assignments. In order to provide her with the most optimal solution we will research the following questions and subquestions during the pre-implementation fase:

- Can an interactive and easy to use tool help a lecturer teach the principles of MapReduce?
  - What are the shortcomings of current course setup?
  - What is an effective model for teaching programming paradigms?
  - What is the most suitable tool that has these properties and overcomes these shortcomings?

2.1. Shortcomings Cloudera VM

Cloudera VM has several shortcomings that make it unsuitable as a tool to use during lectures. Cloudera runs Hadoop jobs as a single-node cluster which means that the mapper and reducer get executed on the same machine. For this reason memory between the mapper and reducer is not separated, something that is not the case in a real cluster. Bad solutions of students might work in a VM but would not in a real cluster. It is hard to teach students the principle of separated memory without the behaviour of a cluster.

Moreover, specifying the number of mappers and reducers is not possible in the VM. This means the lecturer cannot let students experience the difference between a single-node-cluster and a cluster with multiple mappers and reducers. Furthermore, Cloudera takes time to download, install, configure and learn to use. Precious time that could be used for a more in depth look at important aspects of MapReduce such as separate memory for nodes. Finally, professors are unable to track progress of implementations written by students in the Cloudera VM, making the VM unsuitable for interactively teaching students MapReduce during lectures.

2.2. Effective teaching model

Research shows that students involved in an active learning environment understand and remember the material in a course better and less frequently drop out of a course [6]. Another study shows that active learning results in up to 72% improved long-term retention of knowledge compared to passive learning methods [7]. Active learning environments are an effective style of teaching. Research suggests that active learning is especially effective for Computer Science students who tend to be visual/intuitive learners [8]. Given the opportunity to interact with the material, students of most disciplines demonstrate better understanding of the material. Computer Science students especially demonstrate this improvement as suggested by research [9]. Furthermore, collecting and then grading assignments is a tedious process that takes away time from teaching. When an online system is used, these issues could be resolved [10]. These results found in other researches allows us to conclude that an active teaching environment is the best approach to help students improve their understanding of MapReduce. A possible active environment is, as our client also suggested, one that allows students

to work on exercises during lectures. Where the lecturer can also gain insights in the solutions of the students, so he/she can explain topics that students find difficult or have a hard time getting their heads around.

2.3. Requirements

There is a need for a tool that helps students get a better understanding by having ”activity breaks” during a lecture. Doing an assignment about the material discussed in the lecture students tend to be more engaged and thus learn more [11]. The usefulness of the tool would greatly improve if it were to have a two-way communication and educator-student interaction as these aspects are deemed as characteristics of effective lectures [12]. Therefore a high accessibility tool that can be used during the lectures to teach the students the principles of MapReduce would be a suitable solution. The description of this project can be found in Appendix E.

The solution that we introduce will be used to improve teaching of the BDP course. However the solution also has certain requirements that it has comply with. There is no access to a real cluster for the BDP students and for this reason all lab assignments are currently conducted using the Cloudera VM. Therefore the solution must work without a cluster, just like the VM. A solution is sought that enables important cluster-like conditions that the VM lacks, like a scalable amount of nodes and separate runtime memory between nodes. It would enable students to learn how issues like separate runtime memory influence the way one programs in Hadoop, in addition to teaching students the MapReduce paradigm in general. Furthermore, it would be desirable to configure the amount of mappers and reducers dynamically. As implied in this paragraph the goal of this research is not to implement a simulation that does everything that (real) Hadoop offers but rather offer what the Cloudera VM does *not* offer. The goal is to create a tool that complements the current teaching methods and preferably enable the lecturer to use the tool in-class exercises. In order to create a tool that suits the use-case, it will be made accessible and interactive. The teacher will be able to see submissions in real-time, the student must be able to anonymously login and submit solutions. Besides this it should be easy to configure for each lecture. Finally, it has to work on each popular OS (Windows, OSX and Linux) and there may be no requirement to install anything besides a modern browser like Firefox or Google Chrome. Below we include an overview of requirements we will be taking into account while weighing the possible solutions we could implement in the next section.

Category	***Requirement***
Must have	1. must not depend on a cluster
	2. students must be writing code in a language they learned in one of their courses
Should have	3. memory separation of mappers and reducers
	4. ability to define amount of mappers/reducers
	5. ability to see submisions in real time
	6. multi OS support
	7. high accessibility
Could have	8. practical unit tests for students (defined by lecturer)
	9. memory monitoring/limitation
Won’t have	10. the requirement to install Hadoop or a VM

Table 2.1: Generic MoSCoW overview

2.4. State of the art

There are many possible solutions that could be applied to help students grasp the principles of MapReduce faster and better. We will discuss the most promising ideas that we have found or came up with and argue why we have or have not chosen them as our approach to solve our challenge. For a quick overview of this section please see Table 2.2 included at the end of this section.

2.4.1. Website with backend that runs Hadoop jobs

One possible path some programming courses take is to offer a web based solution in which jobs are ran on a (clustered) backend. One of the TU Delft Bachelor Computer Science courses that does this is 'Concepts of Programming Languages' using WebLab [13]. "WebLab, an e-learning platform that lets students write, execute and test Scala programs entirely in their browser" [10] Students are prompted with a simple web based IDE that they can use to write solutions in. When they press run, the code is sent to the server where it is compiled and executed. The output of the run is shown to the student. A similar approach for MapReduce would be practical in a technical sense. "The proofs of concept serve as early evidence for the usefulness of web IDEs as a teaching aid and the feasibility executing advanced editor services in the browser." [10] The implementations of students would be tested using real world conditions: a real cluster in combination with Hadoop. Additionally, students don't have to install anything and solutions are kept online. However, this solution is impossible without the proper financial resources. It is given that limited resources are available to the BDP course which would make the financial aspect a big disadvantage of this solution. An outdated but similar solution has been created before, called WebMapReduce [14].

2.4.2. Multi VM approach

Another possibility would be to have each student install multiple instances of the Cloudera VM and cluster these. This way each student will have his/her own 'cluster' that they can run their MapReduce problems on. This will solve the shared memory problem and can teach students how the system behaves when multiple mappers and reducers are present. However this means that if a professor wants the students to run a job with for instance 10 reducers and 1 mapper, that the student will have to install 11 instances of the Cloudera VM. Having this many active VMs is a big strain on the machine of the student, a slightly older laptop might not even be able to deal with this many VMs. It is also very time consuming to install all these VMs and add them to their simulated cluster. Sadly, this approach still does not allow dynamic lectures in which the professor gets insights in how the students approach given problems, simply because the students' solutions are kept on their local machines. Regardless of the fact that this would be a very realistic simulation for the students these last three drawbacks are simply too big from a teaching perspective for this approach to be practical.

2.4.3. P2P VM approach

A very interesting approach would be to incorporate a Peer to Peer (P2P) network into the Cloudera VM where each student would represent a node in the cluster. Allowing them to use their peers machines as extra mappers/reducers. This way every student would only have to install one instance of the Cloudera VM in which we would incorporate the software that sets up the P2P network. Another advantage is that even though the student only installs 1 VM they can still run very realistic MapReduce jobs that do not have the shared memory problem and the amount of required mappers of reducers can be upscaled up to the size of the network. However this approach also has its drawbacks. For example the amount of mappers/reducers cannot exceed the size of the network. Lets say there is an example which uses 100 reducers while there are only 30 students present then this cannot be simulated. If a peer disconnects while another user is waiting for a mapper/reducer that was running on this peers machine a part of the job will have to be re-run, which causes extra load on the rest of the cluster. If multiple users are running jobs the machines in the cluster might become overloaded with jobs, resulting in a very slow cluster causing all students to be unable to continue their work properly. Another edge case would be if a student does not have an active internet connection or just happens to be working on his own he still has the same situation as before, in which he has shared memory between the mapper and reducer and cannot increase the amount of mappers/reducers. Last but not least it still does not allow for dynamic lectures in which the professor gets insights in how the students approach given problems, which from a teaching perspective is a major drawback.

2.4.4. Client-side simulation

Rather than running jobs in Hadoop in a cluster, one could write and use a framework or tool that runs jobs in a simulated environment possibly with cluster like behaviour. There are multiple ways of doing this and certainly different programming languages can be used but the solutions can be categorised into two very different categories: program based and website based.

Program based

A program based solution has to be installed by the student before it can take advantage of it. For example a Java Swing based application or a plugin that students should install in their webbrowser. Either way, this solution might be platform or browser dependent and cannot be used instantly. However, this is the best performing client-side simulation solution, regarding execution speed and maximum input size. It would also allow the students to write Java (just like in Hadoop).

Website based

As opposed to program based solutions, a website based solution does not need any installation (apart from the browser itself). Such solution would typically use JavaScript. Using JavaScript along with WebWorkers would allow us to configure the amount of mappers & reducers and also separate the memory of the mappers and reducers. JavaScript and most WebWorkers features are known to be supported by all modern browsers [15, 16] (a solution might support a limited set of browsers due to time contraints or less than full browser compatibility). Those characteristics alone are a big plus. By contrast, mapper and reducer code could practically only be written in JavaScript to be evaluated within the browser. Moreover, since everything runs within the browser, the input size of the dataset is limited. As mentioned in section *Website with backend that runs Hadoop jobs*, web IDEs are useful as a teaching aid [10]. The most significant disadvantage of a client-side simulation solution is that a simulation does not have the same conditions as a real Hadoop cluster. For that reason, a website based solution would probably only be useful for educative purposes and only for use cases the solution can accurately simulate conditions.

Solution	Advantages	Disadvantages
The current Cloudera VM	Runs the actual Hadoop framework Cheap to maintain (does not cost anything at all) Well documented	Shared memory between mapper/reducer Cannot simulate behaviour of framework with multiple mappers/reducers Takes a lot of time to install Does not allow for dynamic lectures
Website with clustered backend	Very realistic simulations No shared memory issue Students do not have to install any software besides browser	Hosting the cluster required for the backend is very expensive
Multi VM approach	Very realistic simulations No shared memory issue Can set the amount of mappers/reducers to be simulated	Does not allow for dynamic lectures Takes a lot of time to set up Limits amount of mappers/reducers to amount of VMs the host machine can handle
P2P VM approach	Very realistic simulations No shared memory issue	Does not allow for dynamic lectures Still takes at least as much time to set up as the original VM Limits amount of mappers/reducers to size of P2P network (could be 1) Network could get overloaded resulting in all students being affected by slowness
Client-side simulation using a program	Allows for dynamic lectures Fixes the shared memory issue Can set the amount of mappers/reducers	Requires some sort of installation before it can be used Tends to become platform dependent
Client-side simulation using a website	Allows for dynamic lectures Fixes the shared memory issue Requires no installation whasoever Platform independent Can set the amount of mappers/reducers Cheap hosting costs	Simulations can not be completely realistic Input size is limited

Table 2.2: An overview of the advantages and disadvantages of the discussed solutions

2.5. Choice motivation

In the table below we show which requirements are met by which possible solution, the requirements can be found in Table 2.1, the requirement numbers match with the numbers in this table. A checkmark means the requirement is satisfied, a cross means that the requirement is not satisfied.
Weighing all aspects of each potential solution we quickly arrived at the conclusion that a client side

Solution	**R.1**	**R.2**	**R.3**	**R.4**	**R.5**	**R.6**	**R.7**	**R.8**	**R.9**	**R.10**
The current Cloudera VM	✓	✓	×	×	×	×	×	×	✓	×
Website with clustered backend	×	✓	✓	✓	✓	✓	×	✓	✓	×
Multi VM approach	×	✓	✓	✓	×	×	×	×	✓	×
P2P VM approach	×	✓	✓	✓	×	✓	×	✓	✓	×
Client-side simulation using a program	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
Client-side simulation using a website	✓	✓	✓	✓	✓	✓	✓	✓	×	✓

Table 2.3: An overview of which requirements are met by which possible solution

simulation would be the most feasible solution. We were left with the choice between a client-side simulation using a program or using a website. We could choose between a tool with lower accessibility and memory management or a tool with high accessibility and no memory management. Since we feel like accessibility is an important aspect in this challenge we have chosen for the client-side simulation using a website.

A website with a cluster in the backend would be perfect but is due to the lack of financial resources unachievable. The multi VM approach could work, but puts a big strain on the students machines, putting those with older machines at a disadvantage. Besides that, it still does not allow very interactive lectures and the user experience would probably suffer a lot as well.

The P2P VM approach sounds rather promising but comes with a risk of the network getting overloaded. Furthermore, you cannot simulate more mappers/reducers than there are available machines. Finally, this method still does not allow very interactive lectures. These facts combined made us not choose for this solution as well.

Developing a program that students could install on their machines would have all the advantages the client side webbrowser except for the fact that students would have to install this program. This finalised our decision to implement the client side webbrowser solution. It requires very little resources, provides all features requested by our client and has the best accessibility of all possible solutions.

3

Technical choices & Implementation

This chapter introduces the Technical choices which were made before and during the implementation fase. It elaborates on the requirements, provide an overview of the system's design and finally shortly discuss the implementation.

3.1. Solution specific MoSCoW overview

We start this chapter by defining the exact requirements for website based simulation solution. These requirements are based on meetings we had with both our client and supervisor. We will use this overview as a reference to motivate our (technical) design choices. The overview of the requirements can be found in table 3.1.

3.2. Use cases

Together with our client dr. Claudia Hauff we have defined the following cases the system should certainly be able to simulate:

- WordCount to count the total amount of words in a document.
- WordCount (count the frequencies of each word).
  - Same scenario with 2 mappers and 2 reducers.
- Inlink count (output the number of times each webpage is referred)
- List documents and their categories occurring 2+ times
- Simple WordCount with a combiner summing the total term frequencies in documents (here the result with and without combiner are the same)
- Simple WordCount with a combiner that computes the average term frequencies in documents (here the result with and without combiner are different)
- WordCount example that initializes a dictionary in the setup of the mapper. The mapper would then only map words that occur in the dictionary.
- WordCount example with a setup and cleanup. WordCount would emit the 10 most occurring words.
- WordCount example with a setup and cleanup. This example would emit how many words start with the same letter.
- WordCount example that uses Counters to determine total amount of words (of amount of unique words), essentially different from case 1 where no counter is used.

Category	*Requirement*
Must have	not depend on a cluster
	run completely in a browser
	students must write JavaScript code
	has to be platform independent
	students should be able to write mapper and reducer code
Should have	an interface for the lecturer to see submissions in real-time
	an interface for the students to code in (in JavaScript)
	back-end for lecturer to define exercises in
	separate memory for mapper and reducer
	amount of mappers/reducers should be configurable
	ability to simulate setup for both mapper and reducer
	ability to simulate combiner and cleanup
	simulation of counters
	ability to simulate partitioner
Could have	an interface for the students to see an overview of all exercises
	unit testing (definable in back-end)
	memory management/limitation
Won't have	required installation of plugins/tools

Table 3.1: Requirements defined according to the MoSCoW method

3.3. High level system design

In this section we define a high level system overview and then proceed to explain more in detail how each component should work. Our system consists of four general components: the server, the students' browser, the MapReduce framework and the lecturer's browser (where exercises can be defined and real-time submissions of students can be seen).
Below you will find a graphic representation of this system, we will then proceed to elaborate on each component.

3.3.1. Server

The server serves partially static data (such as code for running the MapReduce jobs), manages (anonymous) sessions, saves submissions in a database, keeps a collection of exercises and renders pages for the lecturer and students. The system we have chosen to serve this content and handle web accessibility is Wordpress.

Wordpress

This research mainly focuses on the framework rather than the website hosting it. Therefore we have chosen not to write a Content Management System (CMS) from scratch, something that is already done several times by others. A well known and documented CMS is Wordpress [17]. Its good documentation and large community are our primary reasons for choosing this solution.

3.3.2. Browser (Lecturer)

The lecturer should be able to login into the admin area, create exercises (and possibly add unit tests for them) and see an overview of the submissions on those exercises. The admin area and exercise creation will be integrated in Wordpress, since this is easily configured in this CMS. The overview of

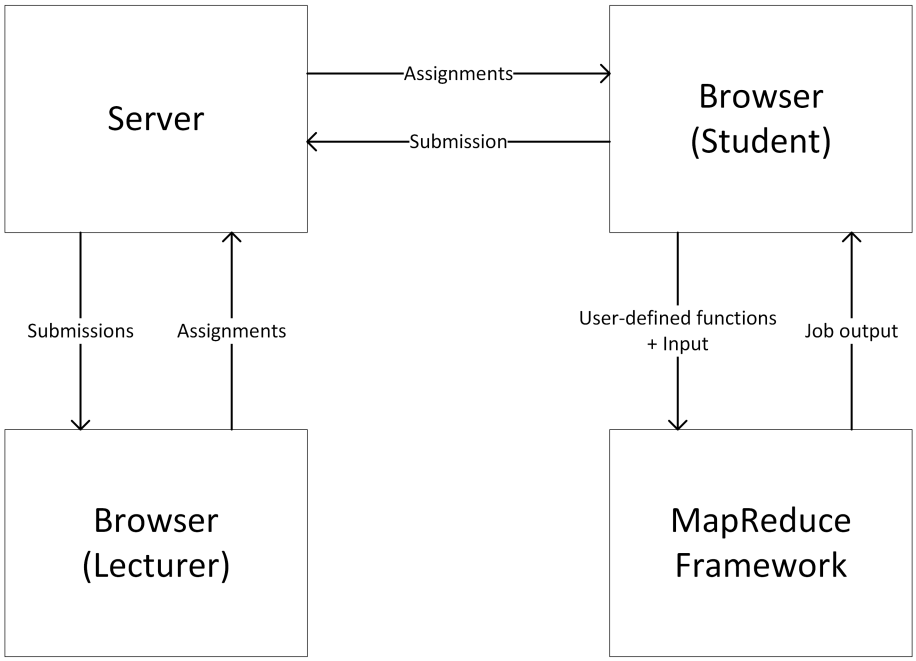


Figure 3.1: a high level overview of the system

submissions will be a separate page on the website which only the lecturer can see. When the lecturer clicks on a submission a student view is opened with the requested submission so the lecturer can directly run it, making it easy to show to the students what needs to be changed to fix an error in a given submission.

3.3.3. Browser (Student)

Students should be able to anonymously use the tool and view the overview of exercises. They should be able to write implementations of mapper/reduce code, run and submit their implementations. The anonymity of the students is taken care of by the server. Each assignment will have a page with several fields in which the students can write mapper/reducer code in JavaScript, buttons to run/submit the solution, a console to see the logs of the simulation and finally the final output of the reducers.

3.3.4. MapReduce framework

The framework should accept input and user-defined mapper/reducer (and possibly setup/combine/-cleanup) functions. Then it should use them to run a MapReduce job and return its output. The framework should use separate memory for different nodes and between mapper/reducer.

State of the art

MapReduce is a popular algorithm that many people have written in several languages including JavaScript. We have looked at multiple Open Source projects written in JavaScript. We will give a small list of the most noteworthy ones and explain why they are good and what lacks them.

- https://github.com/f1ames/maprereduce
  Very simple implementation that works in the browser and able to feed an array as input. However does not use WebWorkers (so memory separation is not possible). Furthermore it is not very advanced, it lacks functions as setup, combine and cleanup. Thus it would not be able to handle several of our use cases defined in section 3.2 and therefore also not able to meet our should have requirements in table 3.1.

- https://github.com/wenkesj/mapreducejs
  Runs in the browser but depends on a server side master, furthermore it only offers map and reduce functions. Because of this limitation this framework is unable to meet one of our must

have requirements defined in table 3.1. Neither is it able to handle all of our use cases defined in section 3.2.

- https://github.com/eterna2/MRCluster
  Able to configure amount of mappers & reducers and able to define map/reduce/partition/combine/cleanup functions. However, it is written for NodeJs (so it requires some refactoring), does not use WebWorkers (so memory separation is not possible) and only accepts files as input. So this framework meets all our requirements but it is written in NodeJS (a server side JavaScript-based language), therefore we would have to modify it to work in native JavaScript. However, due to the fact that modifying this solution has very high potential to take more time than writing a custom solution ourselves we have chosen not to use this framework.

3.3.5. Custom MapReduce framework

We have chosen for a custom solution because we expect it would require a rather limited amount of time to develop and we would get extra features that the state of the art frameworks does not offer us.

WebWorkers

In order to achieve true memory separation between mapper & reducer *and* between different (simulated) nodes one could only use WebWorkers [15]. It would be impossible to truly separate the mapper and reducer memory if we were to use a native approach or any framework approach (like JQuery) due to the fact that global variables could be declared in the mapper which would then be available in the reducer. Furthermore, by moving the computation to WebWorkers you can keep the webpage responsive during code execution.

MapReduce framework Sequence Diagram

In order to simulate MapReduce jobs, one has to follow the MapReduce paradigm. First the input is partitioned and the input is split over the available mappers. The mappers should map the input into key value pairs. Thereafter, the pairs are aggregated and sent to the available reducers. Finally, the reducers send their output back and is displayed to the user. In the following sequence diagram we show this process. Each mapper and reducer will be a WebWorker instance running in the background. This method allows to create multiple WebWorkers if we want to simulate more than one mapper/reducer. The master will instantiate, command and eventually terminate the mapper/reducer workers and provide the user with the output of the workers.

Memory management

We have looked into possibilities of memory management and/or memory limitation. Such features would allow lecturers to teach the difference between memory efficient and inefficient algorithms. However, JavaScript and WebWorkers do not offer memory management or limitation out of the box. In Google Chrome there are ways to request the used JS heap size but requires a non-default enabled flag. Furthermore, it is not possible to specifically request or limit memory usage of a single WebWorker.

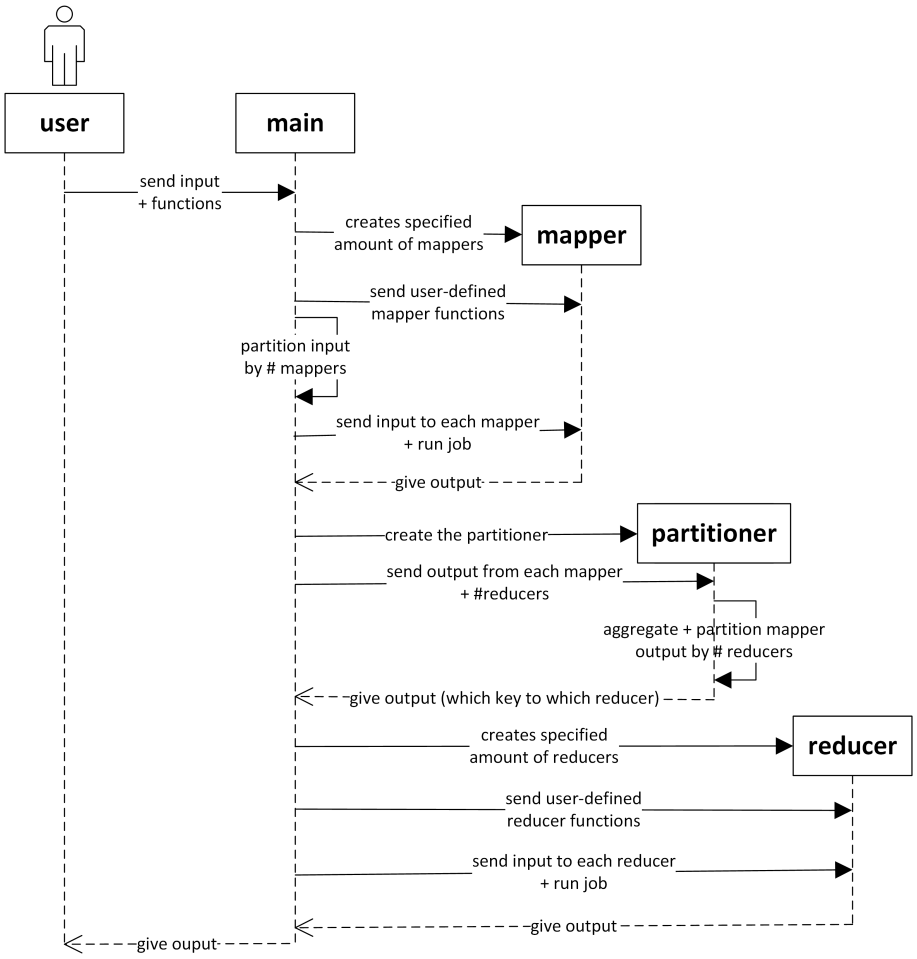


Figure 3.2: MapReduce framework sequence diagram

3.4. Implementation

Our solution, called Luna, was built from scratch and consists of a JavaScript frontend with a PHP backend. The MapReduce code a user can write in this system has to be JavaScript and resembles the core of the Hadoop 2.6.3 API as closely as possible [18] so the user does not have to become familar with two different APIs while learning MapReduce. Luna's most important features are as follows:

- Intuitive interface with easy to use JavaScript code editors.
- Simulation of all functions from the Mapper class [19].
- Simulation of all functions from the Partitioner class [20].
- Simulation of all functions from the Reducer class [21].
- Shows preview of input on the page (full input can be downloaded).
- Shows preview of output on the page after a job is done (full output can be downloaded).
- Allows lecturer to see students submissions in (near) real-time.
- Students submissions are tested against tests predefined by the lecturer. The results are shown to both lecturer and student.
- Accessible since it requires no installation nor registration.

The frontend is responsible for storing (anonymous) authentication details, simulating MapReduce jobs and fetching job data such as input and tests from the server. Communication with the server is done through HTTP requests to our REST API [22] where most of the data is represented in JSON format [23]. The serverside implementation was kept relatively simple, a Wordpress installation which incorporates our custom theme and REST API plugin forms the spine of the backend. The theme includes the MapReduce framework and the student testing framework besides the template and styling definitions. New exercises can be made by creating new exercise pages, for more details on using this system please see D. For further implementation details please see Appendix C.

3.5. Software methodology

During our implementation phase we implemented scrum in order to increase our productivity [24]. We have had daily meetings in which we discussed our previous sprint, whether we were having any problems, how to tackle these and finally plan our next sprint. This way we avoided doing duplicate work, we did not stand in each others way and we solved problems more quickly. Furthermore, we used an online scrum board on trello.com. With a scrum board we could prioritize tasks and see our progress anytime.

4

Research & results

This chapter discusses the conducted research and the results found during this research. We discuss the benchmarking of the system and its results, after which we discuss the conducted user test and its results.

4.1. Benchmarks

We conducted benchmarks using our framework in order to determine performance and the (practical) input size limit. Benchmarks were done using Chrome and Firefox on three different machines using four different cases. Raw results can be found in Appendix F

4.1.1. Setup

The benchmarks were conducted on 3 machines, two laptops and a custom build desktop. Below we include the relevant specifications of these machines. We will refer to these machines as M1, M2 and M3 as described below.

- Machine 1 (M1): Thinkpad Yoga 15 (8gb ram + i7 @ 2.4ghz)
- Machine 2 (M2): Custom Desktop computer (8gb ram + i5 @ 3.8ghz)
- Machine 3 (M3): Acer Aspire V Nitro (8gb ram + i7 @ 2.5ghz)

4.1.2. Benchmarked use cases

The cases we have benchmarked were picked from the list of use cases that were defined in Section 3.2. We chose the following 4:

- Case 1: A simple WordCount example to test simple implementation performance
- Case 2: A more complex WordCount example to test performance
- Case 8: 10 most occurring words example in which workers temporally save data in memory to test memory usage and performance
- Case 10: A counter example in which performance of internal communication in the framework is tested

4.1.3. Graphical results

In the following graphical results you will see the correlation between input size and runtime/memory usage. For runtime you can clearly see a linear correlation. In Figure 4.1 you can see that in one benchmark the partitioner throwed an error due to being out of memory (100MB input, case 1, Firefox, machine 1). For every runtime result we have done the benchmark three times and averaged how long it took to finish by outputting data when it is done running. For every memory usage we have done the benchmark three times as well and averaged the peak memory usage by looking at the native Windows Task manager. We looked at peak browser memory usage to also take into consideration the browsers overhead compared to executing the script separately.

Benchmark results case 1

The benchmark results measuring input size in MB vs runtime in ms were very nice. Generally speaking Firefox had less overhead than Chrome did, resulting in faster runtimes in most cases. However in one case Firefox did go out of memory on Machine 1 (M1). The browser clearly has a big overhead regarding memory usage. Since the browsers memory usage exceeds 1GB with an input size of 100MB we strongly discourage anyone who deploys this platform to simulate cases with this input size. 50MB input size also tends to have memory usage close to 1GB. We recommend input sizes of smaller than 10MB since this input size will result in simulations with runtimes lower than 6-7 seconds.

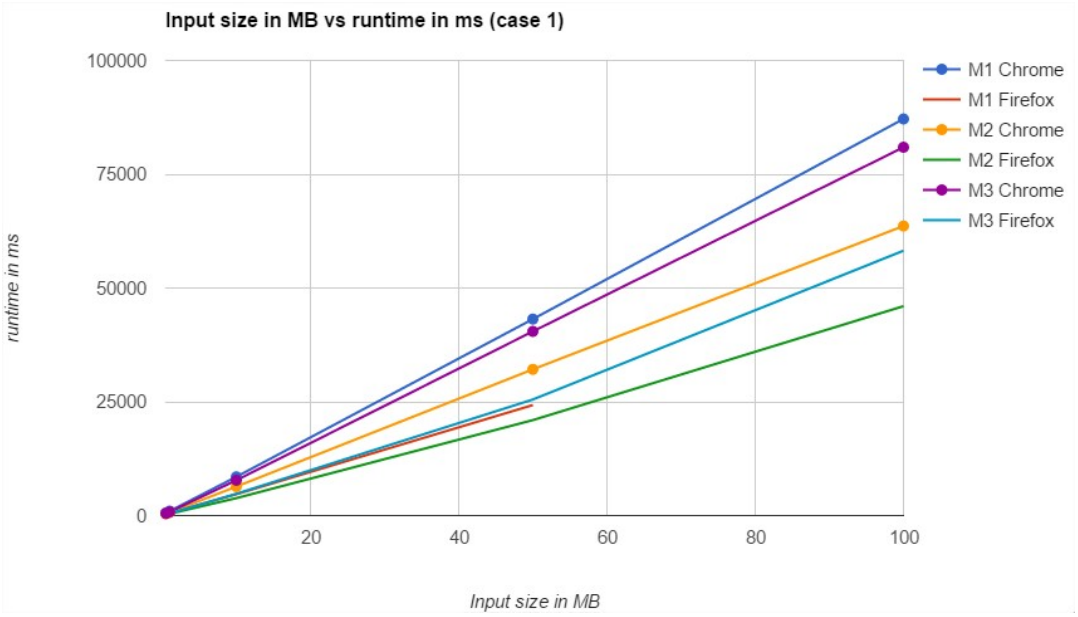


Figure 4.1: Graphical representation of runtime in use case 1

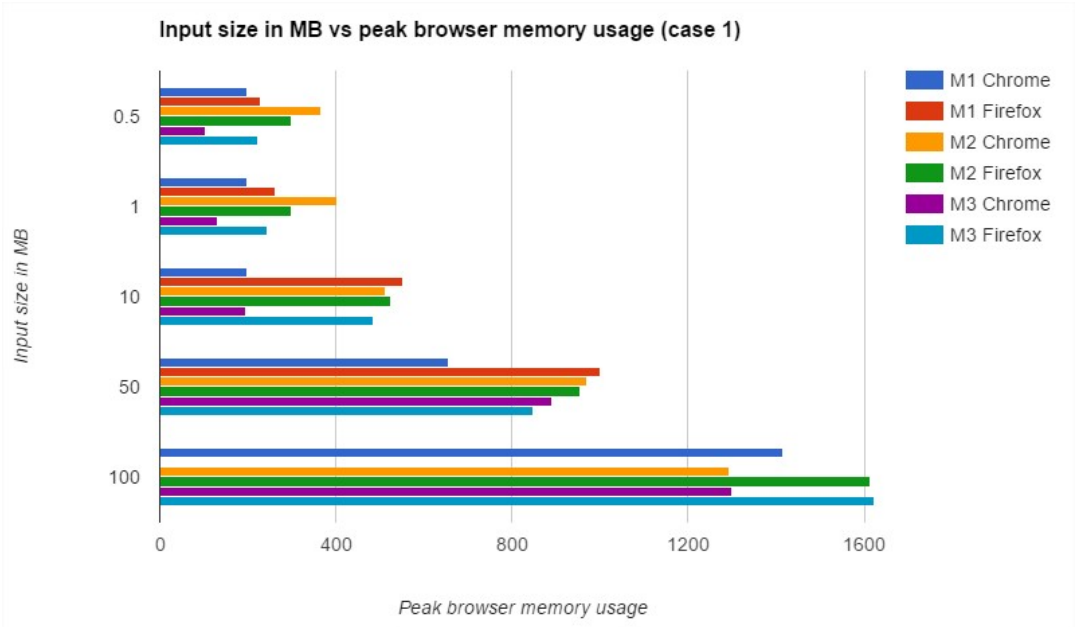


Figure 4.2: Graphical representation of memory usage in use case 1

Benchmark results case 2

Case 2 had severely lower running times, simulations processing about 50MB would take anywhere from 4-6.5 seconds. In Firefox however this input size resulted in a peak browser memory usage of over 900MB, we still recommend inputting 10MB or less when simulating this case.

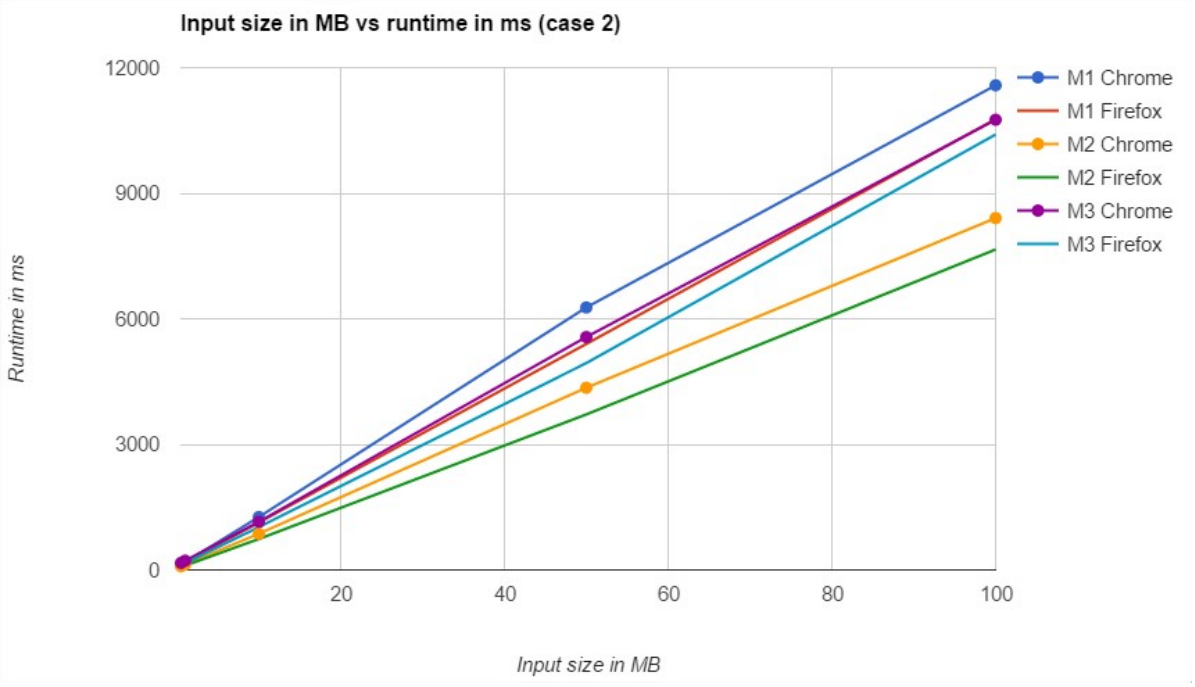


Figure 4.3: Graphical representation of runtime in use case 2

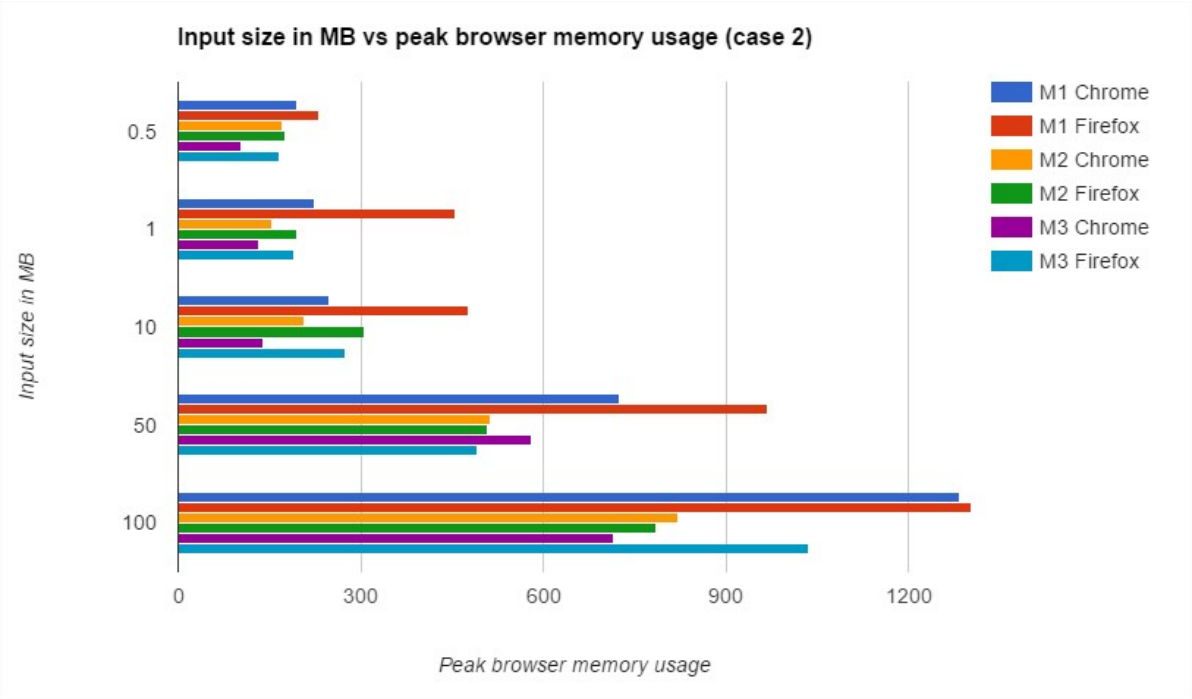


Figure 4.4: Graphical representation of memory usage in use case 2

Benchmark results case 8

Case 8 had quite low runtimes and memory usage, peaking at a runtime of 12 seconds and memory usage of just slightly over 1GB with the largest input size. This case included some more complex computations, which makes the results even more interesting. As with the previous two benchmarked cases we recommend not exceeding 10MB input size, in order to keep the simulations fast not use excessive amounts of memory.

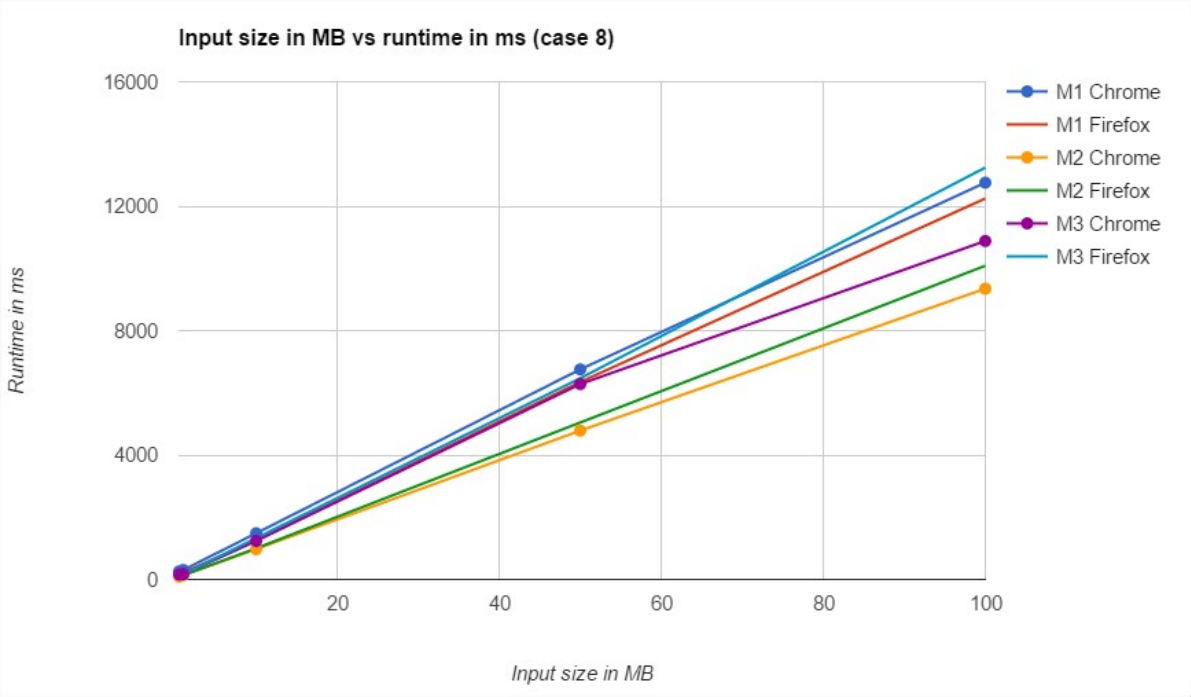


Figure 4.5: Graphical representation of runtime in use case 8

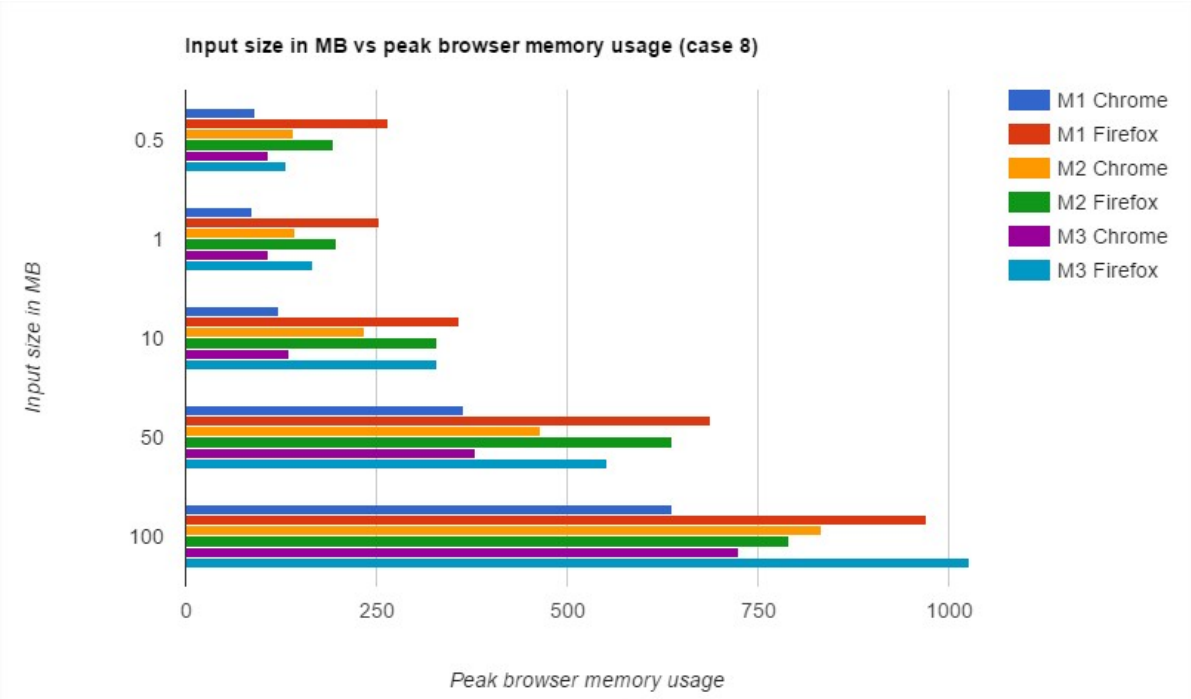


Figure 4.6: Graphical representation of memory usage in use case 8

Benchmark results case 10

The last case we benchmarked, case 10, was primarily to test internal communication within the framework. More specifically, communication between webworkers and the webpage's JavaScript. This appears to be very efficient, processing 100MB of input only takes at most 13 seconds. The browser does however still use close to 1GB of memory while processing this data. So like the other cases we recommend not exceeding 10MB of input for a simulation.

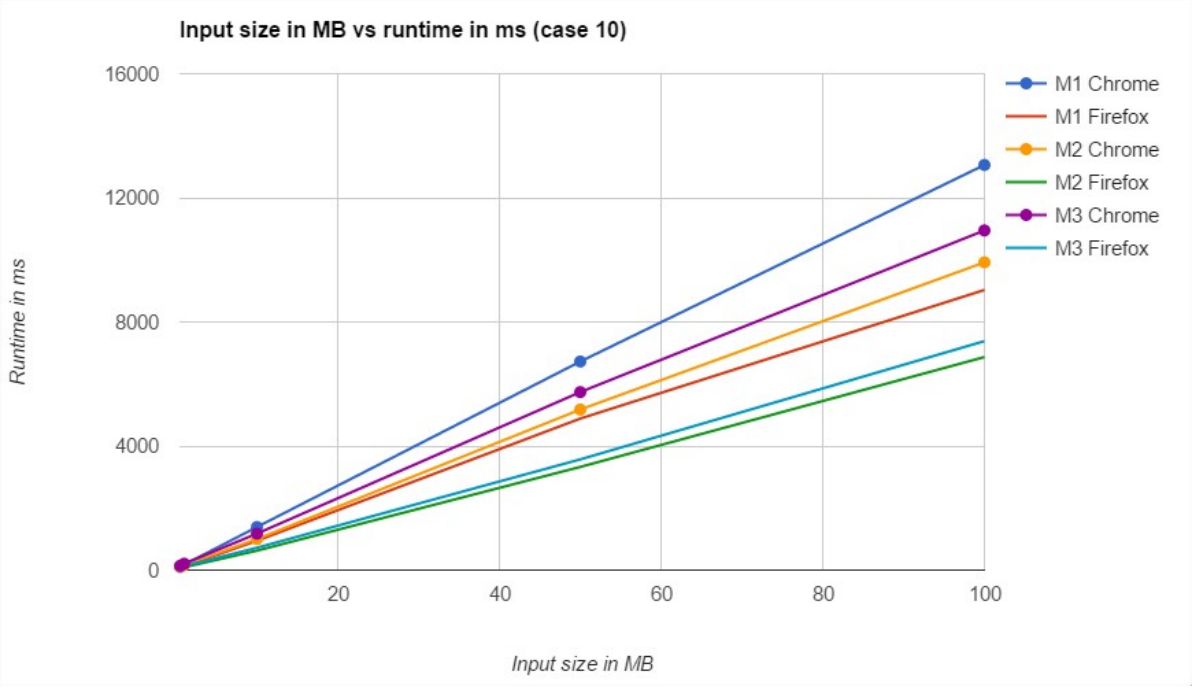


Figure 4.7: Graphical representation of runtime in use case 10

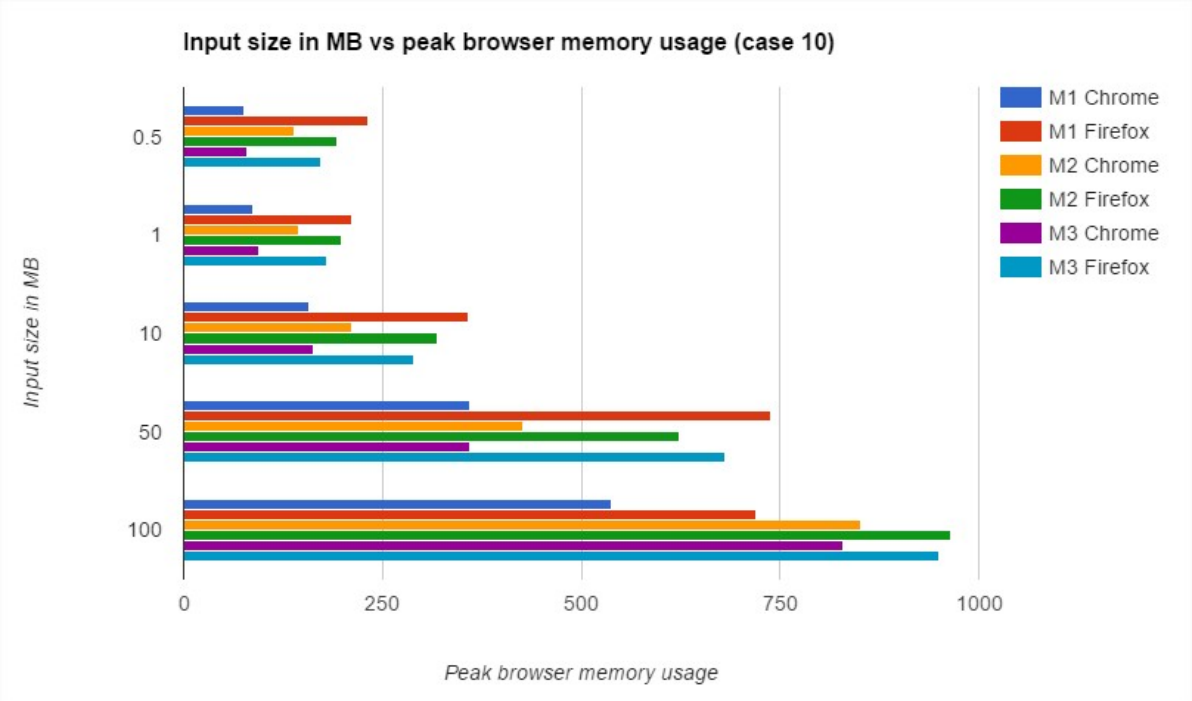


Figure 4.8: Graphical representation of memory usage in use case 10

4.2. User test

In order to streamline our solution we employed an user test with fellow students, some of which did and some of which did not follow the Big Data Processing course. Because there might be a lack of knowledge about MapReduce, we gave students a short introduction lecture. The lecture explained the following topics:

- What Trifle is.
- What MapReduce is.
- How MapReduce works.
- Explanation of assignment 1: WordCount (Participants are asked to implement WordCount that outputs each word with its frequency)
- Explanation of assignment 2: Inlink count (Participants are asked to implement Inlink count that outputs the amount of references to each referenced document)
- Explanation of assignment 3: Counters (Participants are asked to implement a counter that outputs the total amount of words using the context.getCounter(name).increment(amount) API)

After the introduction we explained some simple assignments that are also part of the BDP course and asked the participants to implement these assignments in Trifle. After each assignment we would explain the next one untill all three assignments were completed. After the participants completed the test they were asked to fill out a questionnaire. To see the full questionnaire see Appendix G. The questions were asked about usability of the system and if the participants felt like it had helped them understand the MapReduce programming model.

4.2.1. General results

The most interesting results we found during the test is that JavaScript knowledge is the biggest issue to getting started with the system. Most testers quickly grasped the idea, but made mostly syntax errors or did not know how to implement certain functionalities (like regex matching) in JavaScript. The first assignment took most users the longest, after that, more complicated assignments would be completed rather quickly. So if the system gets deployed and used in an actual lecture of the BDP course it is to be expected that the first (and probably the second) exercise will take above average in time to implement, mainly due to the fact that students will have to refresh their JavaScript knowledge. Furthermore initially the 'context' variable was unclear, we concluded this was mainly due to the fact that we explained MapReduce based on an old Hadoop API, which did not have the 'context' variable. Explaining MapReduce using the Hadoop 2.6.3 API [18] should eliminate this unclarity.

4.2.2. Questionnaire results

The results of the questionnaire will be discussed in the following sections. Each question's results will be visualised using a pie chart. The most interesting results from the questionnaire were found in the questions about error displaying.

Experience with MapReduce/Hadoop

Our testgroup was evenly partitioned, half of the group was somewhat familar with MapReduce while the other half was not. This gave us the opportunity to also filter down other question's results to see how users that were familiar with MapReduce experienced our system differently from users that were not familiar with MapReduce.

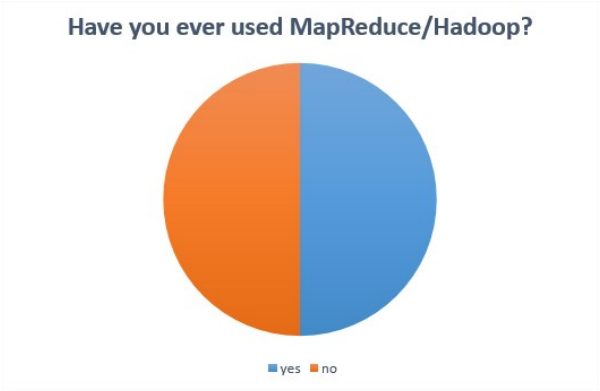


Figure 4.9: User test questionnaire answers as a Pie chart

Navigation and intuitivity of the interface

We started off with some more superficial, nevertheless important questions. No users experienced any problems with navigating him or herself through the website. The users also voted unanimously yes to the question whether the interface was intuitive.

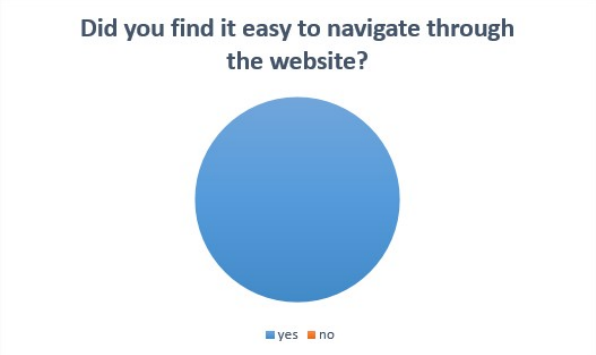


Figure 4.10: Answers to the question whether the website was easy to navigate through

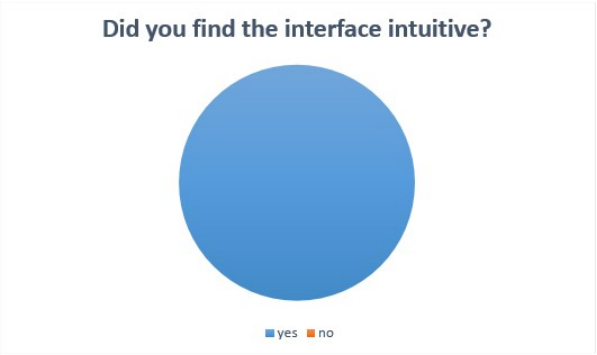


Figure 4.11: Answers to the question whether the interface was intuitive

Tooltips and tab structure

The following two questions were also about the interface, more specifically about the tooltips and tab structure on the exercise pages. The tooltips were found to be very usefull, one user had preferred if the tabs were displayed next to each other. However in the test we had only used two of the 7 editors, the others were hidden. So if only two editors are present it would be a good idea to display the editors besides or below each other, but when there is a total of 7 editors then this is not a very effective way to display them.

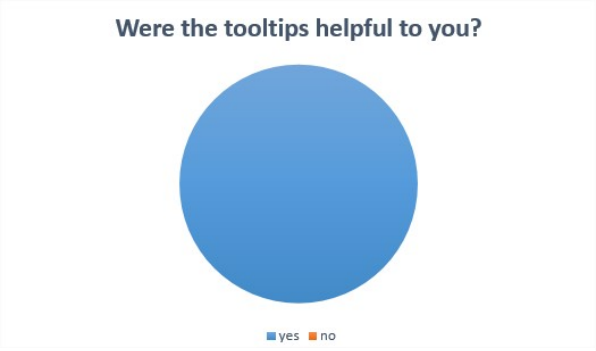


Figure 4.12: Answers to the question whether the tooltips were helpfull

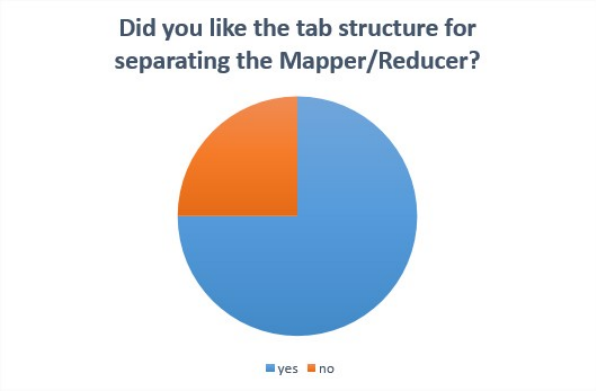


Figure 4.13: Answers to the question whether the user liked the tab structure

Error messages

The question whether the error messages were displayed showed some interesting results, half of the participants thought the error messages were not displayed clearly. Even though the question was focused on the visual aspect, most users answered no due to the fact that the messages themselves were not very clear. Our system in its current setup is only able to determine whether a function written by the user is valid or not. It is unable to tell the user which specific line is syntactically wrong. Even though the Ace editors come with some build in syntax validation, some edge cases are overlooked. For instance in Java one could write:

```
for([Type] element : list){
    //do something
}
```

While in JavaScript this is syntactically wrong. Ace fails to recognise this syntax error, so the user tends to think it is a valid expression. When running a job with a syntax error like described above, our system will show the user an exception with a message "this is not a function". Leaving the user confused what is wrong with their code. We wanted to see if it made any difference whether the user

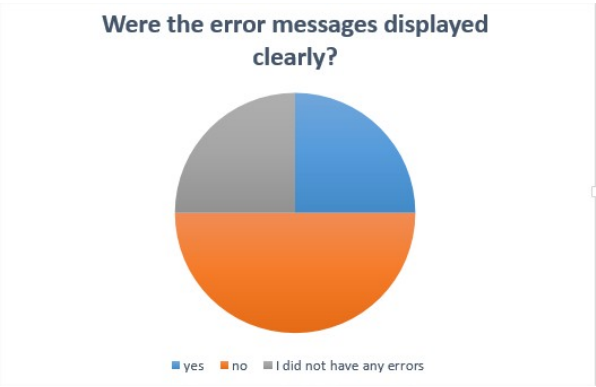


Figure 4.14: Answers to the question whether the error messages were displayed clearly

had some experience with MapReduce beforehand regarding the answers to this question. It turned

out it did not, half the users of each group answered 'no'. Currently we cannot change the way errors

Figure 4.15: Answers from users who did know MapReduce beforehand

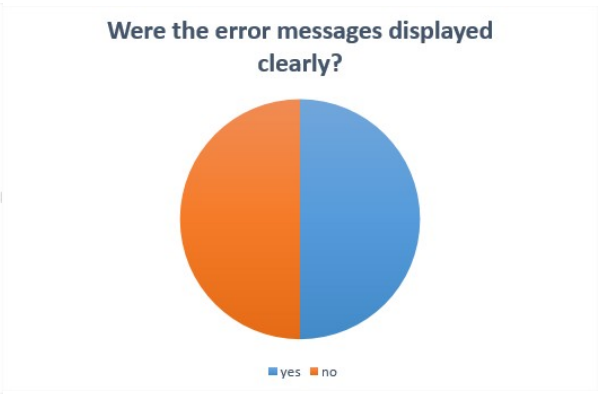


Figure 4.16: Answers from users who did not know MapReduce beforehand

are shown to the user, if a syntax error passes through Ace's syntax validation then there is nothing we can currently do about that. We can only show which function contains an error, but not which line in that function is wrong. For this we would have to make our own custom syntax validator which is far outside the scope of this project. It might however be a good addition to implement in a future work.

Performance

Our test group was very statisfied with the performance of the system. No user had any remarks about long running times. Performance was one of our main concerns, especially cross browser, so its good to have verification that the system runs as efficiently on others machines as well.

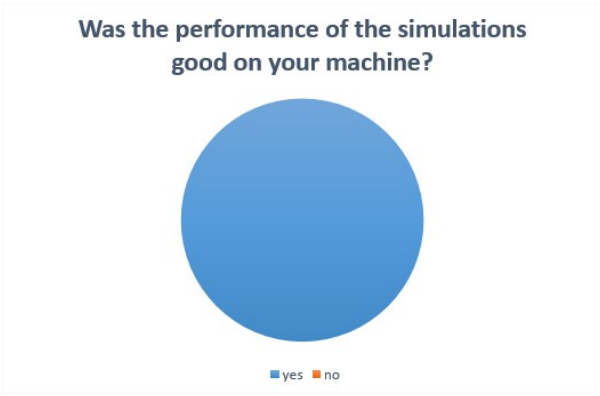


Figure 4.17: Answers to the question whether the performance was good

Trifling around

The users felt like working with the platform helped them grasp the idea behind MapReduce. Being able to quickly dive into the material and run simulations in rapid succession helped better understanding what was going on. One user felt like it would not improve the spead of understanding how MapReduce works, since the answers were anonymous and the person who answered no did not reveil himself we do not know why this user thought this way about the platform.

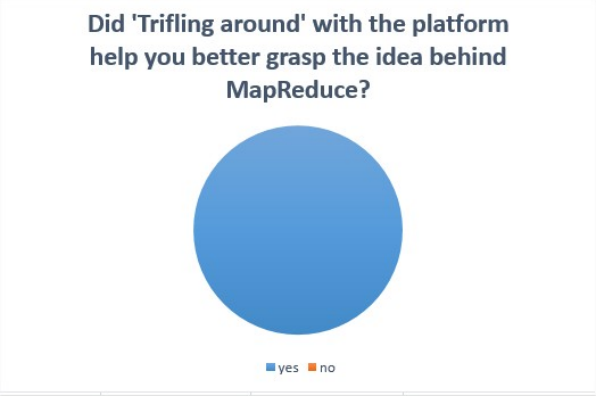


Figure 4.18: Answers to the question whether trifling around helped understanding MapReduce

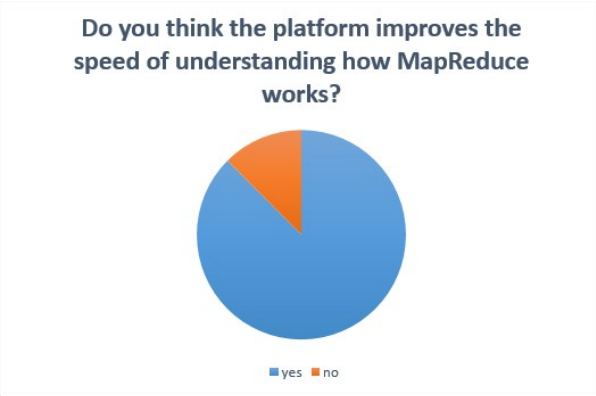


Figure 4.19: Answers to the question whether the user thought the platform improves the speed of understanding MapReduce

Trifle during the BDP course

All users found that the platform is a good addition to the material and would like to use it during the lectures. These questions were asked with the context that they follow the BDP course.

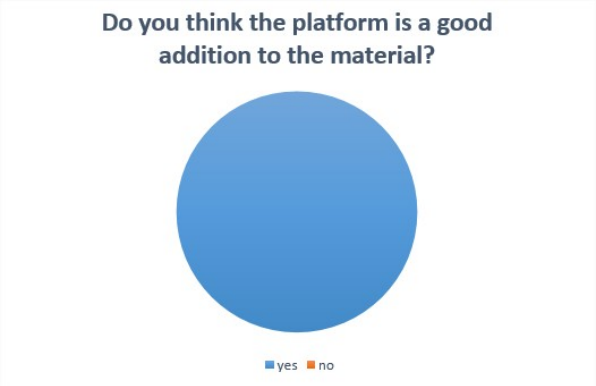


Figure 4.20: Answers to the question whether the platform is a good addition to the material

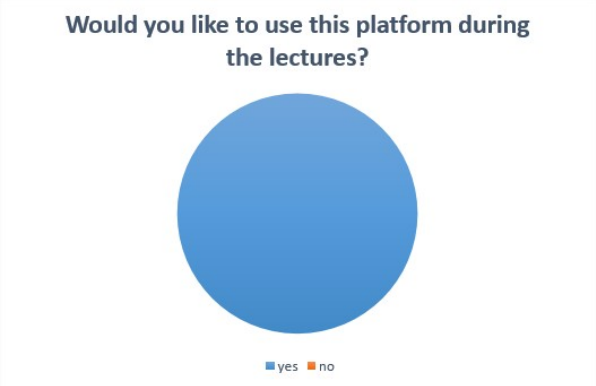


Figure 4.21: Answers to the question whether the user would like to use the platform during lectures

4.2.3. Suggestions from participants

During the user test some participants gave us some suggestions that could improve the platform. We have noted those suggestions down and aggregated them into the following list:

- Rather than only giving the name of the test that fails, also explain why it fails
- Showing to the user that you already submitted an assignment
- Run button looks disabled when grey
- Explain what the context parameter does and when it should be used
- Give better feedback to the user when something is submitting, for example going to the next assignment (if any)
- Explain that students should write JavaScript so they do not get mistaken with syntax of other languages

4.2.4. Remarks client

Our client who attended our user test had some remarks:

- The majority of the participants completed the first two assignments and all participants completed the first assignment.
- It is confusing how the input translates to key/value pairs and whether the value parameter of map is a single line or whole document.
- Context parameter lacks documentation
- It is unclear what the output of the Mapper looks like
- Some participants were not overly familiar with JavaScript syntax
- Submission confirmation should be more obvious

In order to remedy these issues we have done the following things:

- Make the assignment description describe how the input gets split and fed to the mapper
- Explain what the context parameter does and when it should be used in the assignment description
- Add a link to the JavaScript documentation together with explaining that the student code should be fully written in JavaScript

5

Discussion

In the following sections we discuss the results of the project. The target was to implement a high accessibility tool that can be used during the lectures to teach students the principles of MapReduce.

5.1. The resulting implementation

The platform has a high accessibility, as it only requires you to go to a website, no installation needed whatsoever. From the user tests we concluded that it does help understand the principles of MapReduce. Furthermore it overcomes the many shortcomings such as memory separation. Therefore we can answer our research question mentioned in chapter 2. An interactive and easy to use tool *can* help a lecturer to help the principles of MapReduce. The platform was implemented using HTML5, CSS, JavaScript (ECMAScript 6) and uses WebWorkers to do computations in the background without the website becoming unresponsive. As a backend we chose Wordpress for its good documentation and to provide a clean interface to our client to define the exercises in. Every single requirement as defined in section 3.1 is met in the current implementation.

5.1.1. Recommendations based on benchmarks

From the benchmark we can clearly see that the runtime scales linearly with the input size. Furthermore, Firefox computes the result much faster, in some cases twice as fast but that comes with a bigger memory usage compared to Chrome. Due to browser set maximum JavaScript heapspace size, the maximum input size is about 100MB but practically the input size is lower for several reasons. First of all, since the solution will be used in a lecture environment, solutions must run fast so the lecturer can interactively teach the subject. Computing jobs with inputs of 100MB could take several minutes client-side and serving a 100MB input file for hundred students is far from ideal when a low-end server is used. Benchmarking using different input sizes has led us to conclude that the practical limit for input size is somewhere around 1MB, never exceeding 10MB. Only having input files of 1MB or less makes it cheap to host, fast to compute and interactive to use.

Small input sizes used solely to teach students the concept of MapReduce are best suited for our solution. Our solution has minimal memory and cpu usage for those inputs. Our assumption that performance wise, large quantities of data would be impractical to be done in a browser is verified with our benchmark. However, in 2.3 we have noted that ”the goal of this research is not to implement a simulation that does everything that (real) Hadoop offers but rather offer what the Cloudera VM does not offer”. In other words, the goal was to create something that performs well with small assignments that teach concepts like memory separation. Something that we have proven working and therefore benchmark wise our requirements are met.

5.1.2. Feedback from testers

The user test gave us some insights from a different perspective. Students thought the interface was easy to navigate and intuitive to use. Students also mentioned that error messages were confusing and they did not know what was wrong with their code. As discussed in subsection 4.2.2 this was mainly due to the fact that some syntax errors were not picked up by Ace's syntax validation. Furthermore there were some good suggestions regarding user feedback. Some felt like it was unclear wether their submission had succeeded. Another noted that showing multiple notifications at once was not possible. There was hardly any negative feedback on the interface itself, besides that the run button, which is grey, gave the impression it was disabled. Therefore we have decided to change this buttons colour.

5.2. Recommendations for the BDP course

From the results of the user test we have several recommendations for the BDP course. First of all we suggest the students refresh their JavaScript knowledge at home before they work with the system, in order to more quickly start implementing the MapReduce problems introduced in the course. Another recommendation is to explain MapReduce using code examples that follow the Hadoop 2.6.3 API, since our platform also follows this API. During the user test we received quite a few questions regarding this API differing from the code examples we showed on the slides. Therefore keeping the slides in line with this API would make it even easier to start working with Trifle.

5.3. Future work

A nice addition to this system would be a custom syntax validator that does not leave edge cases like described in subsection 4.2.2. This will allow to give more detailed error reports to the users.

Another next step is to take a look at possible improvements for other topics of the Big Data Processing course. For instance incorporating Streaming Algorithms, which is another part of the course, into Trifle.

We think it would be another nice feature to also show students that increasing your cluster size also comes with a speedup instead of just separated output. The current simulation will have to be modified to show this speedup, but it would be a nice addition to the platform.

Currently, users log in anonymously, but a possible extension to the platform is to have users create an account on Trifle, and implement actual lab exercises in Trifle instead of just toy examples during the lectures.

A

Defined custom fields

On the next page we have included a table showing the relevant information regarding the custom fields defined in Wordpress.

Field label	Field name	Field type	Noteworthy attributes	Required	Default value
Week number	week_no	Select	Provides choices 'Week 1' to 'Week 10'	Yes	Week 1
Amount of mappers	mapper_no	Number	Can not be smaller than 0	Yes	1
Amount of reducers	reducer_no	Number	Can not be smaller than 0	Yes	1
Input	input	File	Should be a .zip file containing .txt files which will be used as input for the exercise.	Yes	
Description	description	Wysiwyg Editor		No	
Show editors	show_editors	Checkbox	unchecking one of the editors boxes will hide it from view on the exercisepage but the code provided in the matching editors Text area field will be used during the exercise.	No	all options checked
Mapper.setup	mapper_setup	Text Area	The default value can be changed to provide help or different kind of exercises to the student.	Yes	function setup(context){}
Mapper.map	mapper_mapper	Text Area		Yes	function map(key, value, context){}
Mapper.combine	mapper_combiner	Text Area	Defaults to the elementary combiner (outputs each key-value pair again).	Yes	function combine(key, values, con-text){,while(values.hasNext()) {,context.write(key, values.next());,}}
Partitioner.getPartion	partitioner_partitioner	Text Area	Defaults to a partitioner that splits keys evenly over reducers.	Yes	function getPartition(key, value, numPartitions){,var hashCode = function(s){,return s.split("").reduce(function(x,y){x= ((x<<5)- x)+y.charCodeAt(0);return x&x},0);,},return Math.abs(hashCode(key)) % numPartitions;}
Reducer.setup	reducer_setup	Text Area		Yes	function setup(context){}
Reducer.reduce	reducer_reducer	Text Area		Yes	function reduce(key, values, context){}
Reducer.cleanup	reducer_cleanup	Text Area		Yes	function cleanup(context){}

Table A.1: An overview of the custom fields defined in Wordpress

B

Use cases code examples

This appendix provides code examples for our framework that solves use cases defined in 3.2.

B.1. Use case 1

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').toLowerCase().split(' ');
    for(let word in words){
        context.write(key, 1);
    }
    context.getCounter('example1').increment(1);
}

function combine(key, values, context){
    while(values.hasNext()){
        context.write(key, values.next());
    }
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        count += values.next();
    }
    context.write(key, count);
}
```

```
function cleanup(context){

}
```

B.2. Use case 2

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').toLowerCase().split(' ');
    for(let word of words){
        context.write(word, 1);
    }
}

function combine(key, values, context){
    var emitValue = 0;
    while(values.hasNext()){
        emitValue += values.next();
    }
    context.write(key, emitValue);
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        count += values.next();
    }
    context.write(key, count);
}

function cleanup(context){

}
```

B.3. Use case 3

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z0-9.: ]/g, '').split(' ');
```

```
    for(let word of words){
        let index = word.indexOf(":");
        if(index>-1){
            context.write(word.substring(0,index), key);
        }
    }
}

function combine(key, values, context){
   while(values.hasNext()){
        context.write(key, values.next());
    }
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        values.next();
        count += 1;
    }
    context.write(key, count);
}

function cleanup(context){

}
```

B.4. Use case 4

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z \: 0-9]/g, '').split(' ');
    for(let i in words){
        if(words[i].startsWith('category')){
            let category = words[i].split(':')[1];
            context.write(category, key);
        }
    }
}

function combine(key, values, context){
    while(values.hasNext()){
        context.write(key, values.next());
```

```
    }
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    if(values.hasNext()){
        let first = values.next();
        if(values.hasNext()){
            context.write(first, key);
            while(values.hasNext()){
                context.write(values.next(), key);
            }
        }
    }
}

function cleanup(context){

}
```

B.5. Use case 5

```
function setup(context){

}

function map(key, value, context){
    const words = value.split(' ');
    let termFreq = {};
    for(let i in words){
        let word = words[i];
        if(!termFreq[word]){
            termFreq[word] = 1;
        } else {
            termFreq[word]++;
        }
    }

    for(let word in termFreq){
        context.write(word, termFreq[word]);
    }
}

function combine(key, values, context){
    var emitValue = 0;
    while(values.hasNext()){
        emitValue += values.next();
```

```
    }
    context.write(key, emitValue);
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let total = 0;
    while(values.hasNext()){
        total += values.next();
    }
    context.write(key, total);
}

function cleanup(context){

}
```

B.6. Use case 6

```
function setup(context){

}

function map(key, value, context){
    const words = value.split(' ');
    let termFreq = {};
    for(let i in words){
        let word = words[i];
        if(!termFreq[word]){
            termFreq[word] = 1;
        } else {
            termFreq[word]++;
        }
    }

    for(let word in termFreq){
        context.write(word, termFreq[word]);
    }
}

function combine(key, values, context){
    let emitValue = 0;
    let count = 0;
    while(values.hasNext()){
        count++;
        emitValue += values.next();
    }
```

```
    context.write(key, emitValue / count);
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let total = 0;
    let count = 0;
    while(values.hasNext()){
        count++;
        total += values.next();
    }
    context.write(key, total / count);
}

function cleanup(context){

}
```

B.7. Use case 7

```
function setup(context){
    this.animals = ['cat', 'dog', 'mouse'];
}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').toLowerCase().split(' ');
    for(let i in words){
        let word = words[i];
        if(this.animals.indexOf(word) >= 0){
            context.write(word, 1);
        }
    }
}

function combine(key, values, context){
    var emitValue = 0;
    while(values.hasNext()){
        emitValue += values.next();
    }
    context.write(key, emitValue);
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x
    }
    return Math.abs(hashCode(key)) % numPartitions;
```

```
}

function setup(context){

}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        count += values.next();
    }
    context.write(key, count);
}

function cleanup(context){

}
```

B.8. Use case 8

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').toLowerCase().split(' ');
    for(let i in words){
        context.write(words[i], 1);
    }
}

function combine(key, values, context){
    var emitValue = 0;
    while(values.hasNext()){
        emitValue += values.next();
    }
    context.write(key, emitValue);
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){
    this.tenMostOccurring = {};
}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        count += values.next();
    }
    this.tenMostOccurring[key] = count;
```

```
    let _self = this;
    let sortedWords = Object.keys(this.tenMostOccurring).sort(function(a,b){
        return _self.tenMostOccurring[b] - _self.tenMostOccurring[a];
    });

    if(Object.keys(this.tenMostOccurring).length > 10){
        delete this.tenMostOccurring[sortedWords.pop()];
    }
}

function cleanup(context){
    for(let word in this.tenMostOccurring){
        context.write(word, this.tenMostOccurring[word]);
    }
}
```

B.9. Use case 9

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').split(' ');
    for(let i in words){
        context.write(words[i], 1);
    }
}

function combine(key, values, context){
    while(values.hasNext()){
        context.write(key, values.next());
    }
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){
    this.characters = {};
}

function reduce(key, values, context){
    while(values.hasNext()){
        let value = values.next()
        let character = key.charAt(0);
        if(!this.characters[character]){
            this.characters[character] = value;
        } else {
            this.characters[character] += value;
        }
    }
}
```

```
function cleanup(context){
    for(let character in this.characters){
        context.write(character, this.characters[character]);
    }
}
```

B.10. Use case 10

```
function setup(context){

}

function map(key, value, context){
    const words = value.replace(/[^a-zA-Z ]/g, '').toLowerCase().split(' ');
    context.getCounter('total').increment(words.length);
}

function combine(key, values, context){
    while(values.hasNext()){
        context.write(key, values.next());
    }
}

function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0
    }
    return Math.abs(hashCode(key)) % numPartitions;
}

function setup(context){

}

function reduce(key, values, context){
    let count = 0;
    while(values.hasNext()){
        count += values.next();
    }
    context.write(key, count);
}

function cleanup(context){

}
```

C

Implementation

C.1. MapReduce Framework

In this chapter we will discuss how our implementation works. The API we have written for our MapReduce framework is as similar as the Hadoop (v2.6.3) [18] framework so students don't need to get familiar with two completely different API's. We will start this section by describing the Languages and Technologies used to implement the MapReduce framework, after that we will proceed to describe the components of the framework and the signatures of the functions that are part of the input. Finally we will explain the data flow in the framework and point to some examples that show that our framework can simulate at least the required scenarios.

C.1.1. Languages

Our framework is purely written in JavaScript and uses WebWorkers to simulate a mapper or reducer machine. By using WebWorkers we keep the memory between all mappers and reducers separated. Mapper and reducer implementations are defined in separate files and WebWorkers are run with only either of those, this means a single simulated machine can only be a mapper or reducer and never both. Communication between master and WebWorker happens through standard WebWorker messages [15].

C.1.2. Components

Framework context

All user-defined functions except the partitioner come with a context parameter. This can be used to call framework functions.
The api is as follows:

```
context.getCounter('exampleCounterName').increment(1);
```

Master

The master components accepts input, configuration and callback functions. The master initializes WebWorkers, configures them with user defined functions and distributes the work load. Once all reducers are done it returns the final result but it also returns logs and errors during the processing itself. The data is returned by calling specified callback functions.

Mapper

The mapper accepts three user-defined functions: setup, map and combine. First the setup, thereafter map for each key value pair and finally the combiner is called. Map and combine functions have a 'values' parameter which is an iterator: once a value is read it cannot be read again. When the mapper is done, the result is returned back to the master through a message.
The mapper API is as follows:

```
function setup(context){
```

```
    // define variables that can be used later in map or combine
}

function map(key, value, context){
    //context.write(a_key, a_value) to emit pairs
}

function combine(key, values, context){
    //context.write(a_key, a_value) to emit pairs
}
```

Partitioner

The partitioner is called for each key value pair emitted by mappers to determine to which reducer the pair should be send.
The partitioner API is as follows:

```
function getPartition(key, value, numPartitions){
    // return reducer id within interval [0, numPartitions)
}
```

The standard Hadoop partitioner is the HashPartioner [25] which distributes keys uniformly over the available reducers. We have implemented a similar partitioner to test if our framework works as expected [26].

```
function getPartition(key, value, numPartitions){
    var hashCode = function(s){
        return s.split("").reduce(function(x,y){
            x = ((x << 5) - x) + y.charCodeAt(0);
            return x & x;
        }, 0);
    }
    return Math.abs(hashCode(key)) % numPartitions;
}
```

Reducer

The reducer accepts three user-defined functions as well: setup, reduce and cleanup. The setup in the reducer also gets called first, secondly the reduce function for each pair and finally the cleanup function. Just like the mapper, 'values' in reduce and cleanup are iterators: once a value is read it cannot be read again.
The reducer API is as follows:

```
function setup(context){
    // define variables that can be used later in map or combine
}

function reduce(key, values, context){
    // context.write(key, value) to emit pairs
}

function cleanup(context){
    // context.write(key, value) to emit pairs
}
```

C.1.3. Data flow

A quick overview of the data flow of the framework:

- Define a configuration with input, user defined paritioner/mapper/reducer code and number of mappers/reducers to use.
- Create a MapReduce object with the configuration as argument
- The MapReduce framework will verify configuration and set internal variables if the configuration is valid
- Mapper and reducer WebWorkers are created and configured with the user defined functions. Furthermore, an event listener is attached to each WebWorker in order to retrieve result, log, error and counter data.
- Define and set log, error, counter and result callback functions.
- Call run function on MapReduce object
- The framework will split the input of each document (default line split)
- Mappers are fed with chunks of the input with key (document id) value (line data) pairs
- Mappers process the data and while doing so they will message logs, errors, counter increments and results back to the master. The master will in turn call its callback functions.
- The framework creates a partitioner WebWorker and provides this worker with the number of reducers, key value pairs from the mappers and the user defined getPartition function.
- The partitioner assigns a reducer id to the key value pairs that mappers created and returns this to the MapReduce object.
- The key value pairs now get assigned to their respective reducers based on the id returned from the partitioner.
- Reducers process the data and while doing so they will message logs, errors, counter increments and results back to the master. The master will in turn call its callback functions.

C.1.4. Simulating the required scenarios

In section 3.2 we discussed certain use cases that should certainly work using our solution. We have written the code for our framework that can simulate each of these use cases, this code is tested and gives the expected results. The implementations can be found in Appendix B.

C.2. Student testing framework

Grading student implementations is a tedious task and our client requested we would implement automatic validation of the output of jobs against a predefined expected result.

First of all we have defined a test configuration in JSON for each assignment. This JSON consists of the name of the test, the input, the number of mappers/reducers, and expected counters/result. An example of such JSON is as follows:

```
{
    ...
    "tests": [
        {
          "name": "One mapper, one reducer, single document test",
          "input": [
            {
              "name": "text.txt",
              "content": "The dog is in the house"
            }
```

```
          ],
          "numMappers": 1,
          "numReducers": 1,
          "counters": {},
          "output": [
            {
              "text.txt": 6
            }
          ]
        }
    ]
}
```


C.3. Front-end

For the front-end of the website we have decided to use Bootstrap, Ace, jQuery, custom css and of course HTML. In this section we shall elaborate on these frameworks and languages. The design of the website will also be introduced and discussed shortly.

C.3.1. Design

The website we have designed ended up having 4 types of pages: the homepage, exercise pages, an overview page that shows available exercise pages and the submission page where the lecturer can see the submitted solutions. We will shortly introduce each page type. The design explicitly does not take mobile devices into account, since this is a coding environment and we do not think students will be writing code on their phones during a lecture.

The homepage

We wanted to keep the homepage simple, since each visitor gets assigned a random identifier we did add the easy to use Google ReCaptcha [27] as we did not to risk our system being flooded by random IDs in case a bot spams the login form.

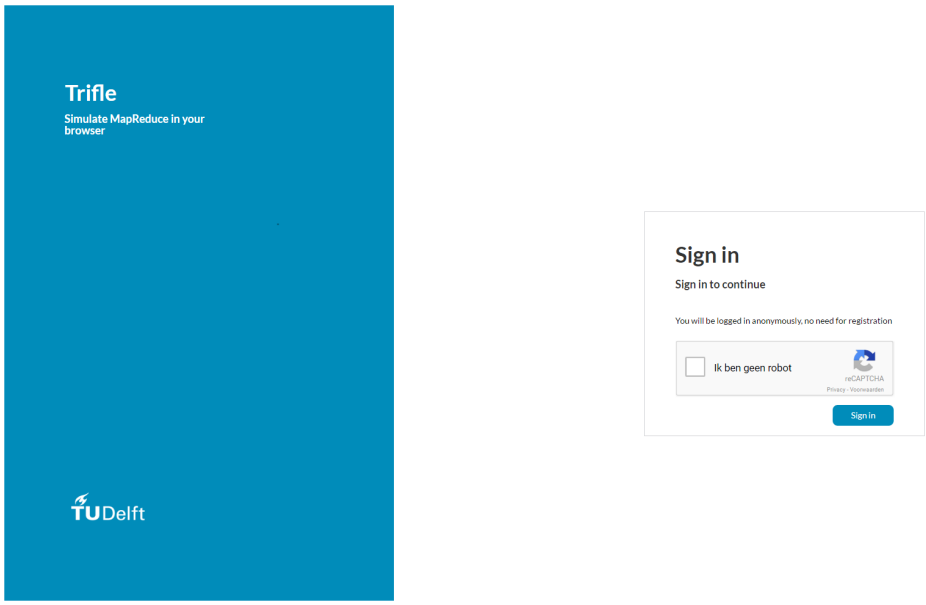


Figure C.1: The design of the homepage

The overviewpage

The overviewpage shows an accordion [28] where each week can expand and show the available exercises for this week, which are linking to the corresponding exercisepage.

Figure C.2: The design of the overviewpage

The exercisepage

In contrast to the simple home and overview page, the exercise page has a lot of content. At top of the page is the exercise description (which can be folded in to save space on the page). On the left side the MapReduce code can be written, we have split the classes mapper [19], partitioner [20] and reducer [21] as seen in Hadoop over three tabs with code editors to write the methods that need to be implemented.
On the right side a similar tab structure is used to display the input and output of the job. Both the input and output will be truncated on the page to a maximum of about 10 lines, however a download will be provided to see the full in- or output.

The lecturer page

When an active admin panel session is active, the submission button is present on the overview page and the lecturer can click on it to view the submissions. When one or more students submitted their code, submissions will be appear on the submissions page with an id and test score. The page groups submissions by assignment and also tells you how many submissions there are and how many active participants (at least one heartbeat in the last 60 seconds) there are. Furthermore, when the lecturer clicks on a submission, a new tab is openened with the exercise page and submission of the student filled in. This way the lecturer practically sees exactly what the student sees and could interactively help with problems students encounter.

C.3.2. Ace

We have decided to use the Ace code editor plugin [29] for the students to write their code in. Since we wanted students to be able to write code in the browser we wanted to give them a smooth coding experience. Ace is highly configurable, meaning you can include only what you need. For us this meant including the JavaScript mode, Textmate theme (to match the design we will discuss later) and the so called language tools which allowed us to add custom autocompletion to the editors. We chose Ace over other options like ICEcoder and CodeMirror because it is free, simple to implement, customisable (only include what you need) and it packaged versions of Ace can be loaded for free over Content Delivery Networks (CDN) like JsDelivr and cdnjs.

C.3.3. Bootstrap

To build our front-end we did not want to start completely from scratch, during design we explicitly kept in mind a clear row/column structure so it would be very easy to build using the Bootstrap framework. Bootstrap also has build in support for tabs and accordions which is made building the overview- and

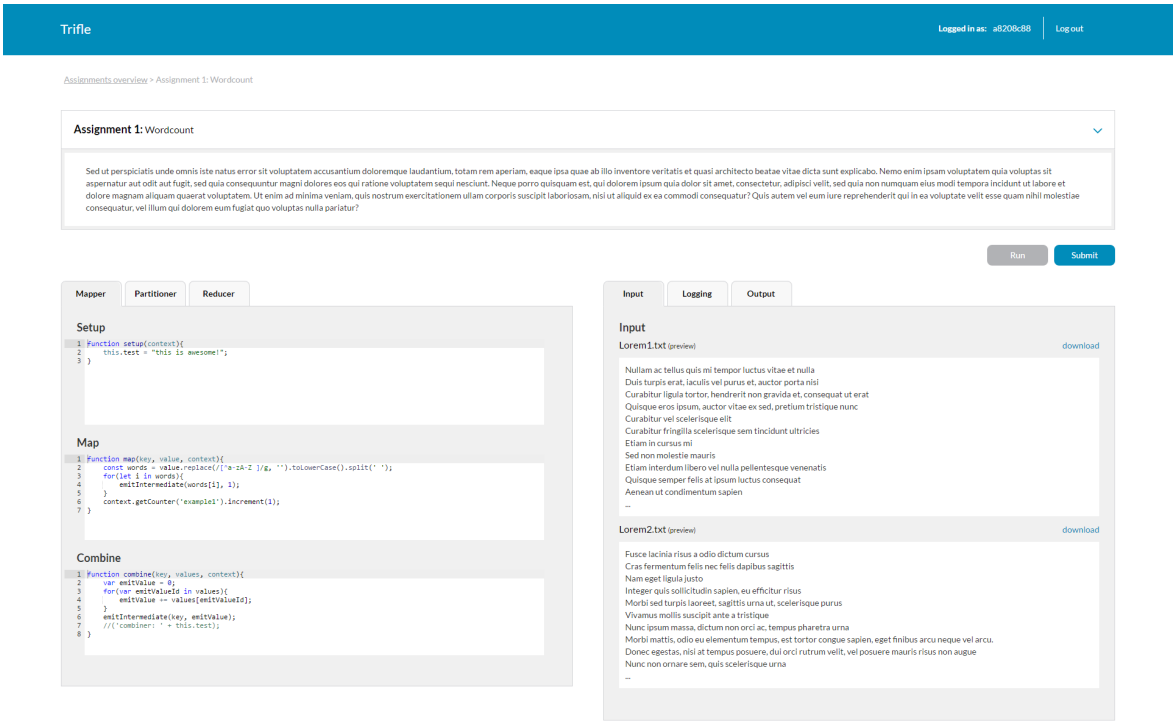


Figure C.3: The design of the exercisepage

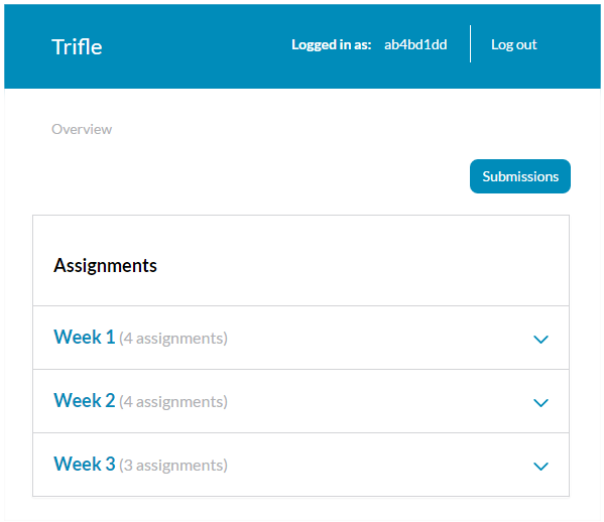


Figure C.4: The submissions button being displayed

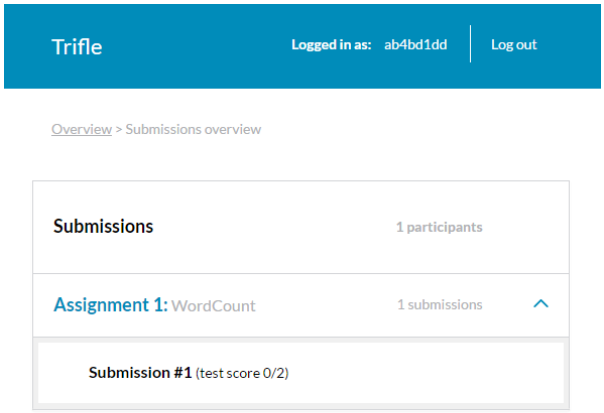


Figure C.5: The lecturer page with one submission

```
1 function getPartition(key, value, numPartitions){
2     var hashCode = function(s){
3         return s.split("").reduce(function(x,y){x=((x<<5)-x)+y.charCodeAt(0);return x&x},0);
4     }
5     return Math.abs(hashCode(key)) % numPartitions;
6 }
```

Figure C.6: The ace editor with the textmate theme

exercisepages a lot easier.

C.3.4. jQuery

Bootstrap builds upon the jQuery framework, therefore we decided to use it in our webpages' javascript as wel. jQuery makes DOM selections very easy and there was no need for any other framework as all the functionalities we needed were available in either jQuery or native javascript.

C.4. Back-end

The backend of our website will be written in PHP in combination with WordPress. The hosting machine will be very likely CentOS-based as this is typically the setup of a server that our client receives from the TU Delft. As defined in our system requirements in Table 3.1, our system has to be platform independent. Therefore we have chosen a WordPress backend, since this runs on both Windows and Unix-based Operating Systems. In this section we will elaborate on how we have set up the backend of the website.
As described in Subsubsection 3.3.1 we have chosen to use Wordpress as our CMS. Creating a custom Wordpress website requires a few things, a Theme, optionally one or more plugins for extra functionalities and some custom database tables in addition to the Wordpress tables.

C.4.1. Wordpress theme 'Trifle'

A Wordpress theme purely defines the 'skin' of the website, for us this means our custom design. We transformed the html we had build into templates, each pagetype getting its own template. For a description of all pagetypes see Section C.3. In the backend a template can then be chosen to define what type of page the administrator is creating (i.e an exercisepage or perhaps a new overview page). The theme also includes all css files, images and the javascript page controller to style and operate the pages correctly. However it also includes the JavaScript files of the MapReduce framework as this is technically part of the front-end and there was no cleaner way of incorporating the framework into

the website. The theme also includes a special functions file, here we have defined a function that is executed on the 'publish_page' hook. This function extracts the uploaded input zip and transforms it into a JSON format in combination with other fields filled in for the exercise and stores it into the database. This function simultanously extracts the tests zip and incorporates this into the assignment JSON before storing it in the database, the format of this zip is described in D.2.2. Where this is stored will be elaborated on in Subsection C.4.3.

C.4.2. Wordpress plugins

In our Wordpress installation we have included two plugins, the popular Advanced Custom Fields (ACF) plugin [30] and our own newly created plugin 'Custom API' which builds upon the Wordpress REST API. We will continue to elaborate on each plugin.

Advanced Custom Fields

The ACF plugin allows its users to define conditional logic to display certain input fields depending on which template has been selected. In this system this mainly affects the exercisepages as there is nothing to configure regarding the home, overview and submissions pages. We have used ACF to define all the fields we needed for the Exercise pages (e.g the exercise description or the input of the job). For an overview of the fields we have defined see Appendix A.

Custom API plugin

We have also implemented our own REST API in Wordpress, this API can be used for example to get exercise data or submit an assignment. The defined endpoints are:

`GET /wp-json/api/v1/register`

This endpoint takes a querystring parameter 'g-recaptcha-response' as input, first validates this with Google, if it is validated correctly (i.e the user is not a bot) then a cookie will be generated in the database and returned to the user. The cookie is required to get access to the rest of the endpoint.

`GET /wp-json/api/v1/submissions/<ID>`

This endpoint fetches a submission with a given identifier from the database. First checks the authentication cookie and will return a HTTP status 401 unauthorized in case the cookie is invalid.

`POST /wp-json/api/v1/submissions/create`

This endpoint takes a 'submission' parameter, which should be the properly formatted JSON containing the users functions. The JSON should be structured as seen below. It also checks the authentication cookie and will return a HTTP status 401 unauthorized in case the cookie is invalid.

```
{
    mapper_setup: <mapper.setup code>,
    mapper_mapper: <mapper.map code>,
    mapper_combine: <mapper.combine code>,
    partitioner_partitioner: <partitioner.getpartition code>,
    reducer_setup: <reducer.setup code>,
    reducer_reducer: <reducer.reduce code>,
    reducer_cleanup: <reducer.cleanup code>
}
```

`GET /wp-json/api/v1/assignment/<ID>`

This endpoint fetches an assignment with a given identifier from the database. As described in Subsection C.4.1 the assignments are stored in a JSON format in the database. It also checks the authentication cookie and will return a HTTP status 401 unauthorized in case the cookie is invalid.

C.4.3. Custom Database Tables

- Table wp_assignments for storing assignment data by id of wp_posts: id: INT(11), assignment: LONGTEXT

- wp_custom_users for keeping track of cookies and active participants: id: INT(11), cookie: VARCHAR(45), lastheartbeat: INT(11)
- wp_submissions for storing submissions of users by userid of table wp_custom_users and assignmentid of table wp_assignments: userid: INT(11), assignmentid: INT(11), submission: LONGTEXT, timestamp: INT(11)

C.4.4. Anonymous login

The anomymous login works relatively simple with the Google recaptcha implemented on the homepage. The user checks the "I am not a robot" box, perhaps has to perform one of the ReCaptchas puzzles and then presses 'login'. A GET request is done to the login form backend, where the google-recapcha-response parameter is passed on to the registration endpoint, which in turn validates the response and returns the cookie. The form backend then sets this cookie and redirects the user to the overview page. From this moment on the user has its valid cookie and can use this to get assignments and submit solutions.

C.5. Testing

In order to guarantee that our framework works as expected, we have employed two types of testing: QUnit [31] and Selenium [32]. QUnit is used to test that our framework runs MapReduce jobs with the correct output whereas Selenium is used to test whether our site works as expected. In other words we have separated website and framework testing. For QUnit we have individually tested framework components: mapper, reducer and mapreduce as a whole. For each component we have tested whether running a job leads to expected job output, counters and errors all individually. Furthermore, for mapreduce as a whole we have tested whether jobs with single reducer/mapper works, whether jobs with multiple mapper/reducers work and last but not least, whether all our 10 use cases run and give the expected result. For Selenium we have tested if the navigation and functionality of the website works as expected. For example, we have tested if clicking on an assignment actually redirects you to the exercise page. Other tests include for example whether you can fill in the MapReduce code, press on run and get job output.

D

Using the system

D.1. Installing the website

In this quick step by step guide we will give a general guide how Trifle can be setup by a developer. Steps may be different depending on the type of Operating System that is used. These steps we provide are done on a CentOS machine.

- Install Apache
- Install MySQL
- Clone Trifle repository in /var/www/html folder
- Load database using dump.sql file
- Visit <url>/wp-admin to login to the admin panel using credentials: user: admin password: OeDvfACs2gm (change this after first login)
- Visit <url>/ to login to the student/lecture view (lecture view can only be used when you have an active admin panel session)

D.2. Using Wordpress

In this section we will discuss how to use Wordpress in combination with our custom theme 'Trifle'. Trifle uses the two plugins described in Subsection C.4.2 which do not require the user to configurate anything.

D.2.1. Page hierarchy

Before we go in depth on how to set up the pages we will discuss the hierarchy of the pages. This is actually very simple, in wordpress the homepage, overview page and lecturer page will be defined on the same level (root level). Then each exercisepage will have to be defined with the overview page as parent. To visualise the hierarchy we have included a small diagram below.

D.2.2. Setting up the first pages

Now that the global hierarchy is clear we will proceed to explain how to set up the pages in the website. The setup consists of the following steps:

- Creating the login page (and setting it as the homepage)
- Creating the overview page
- Creating the lecturerpage
- Creating an exercise page

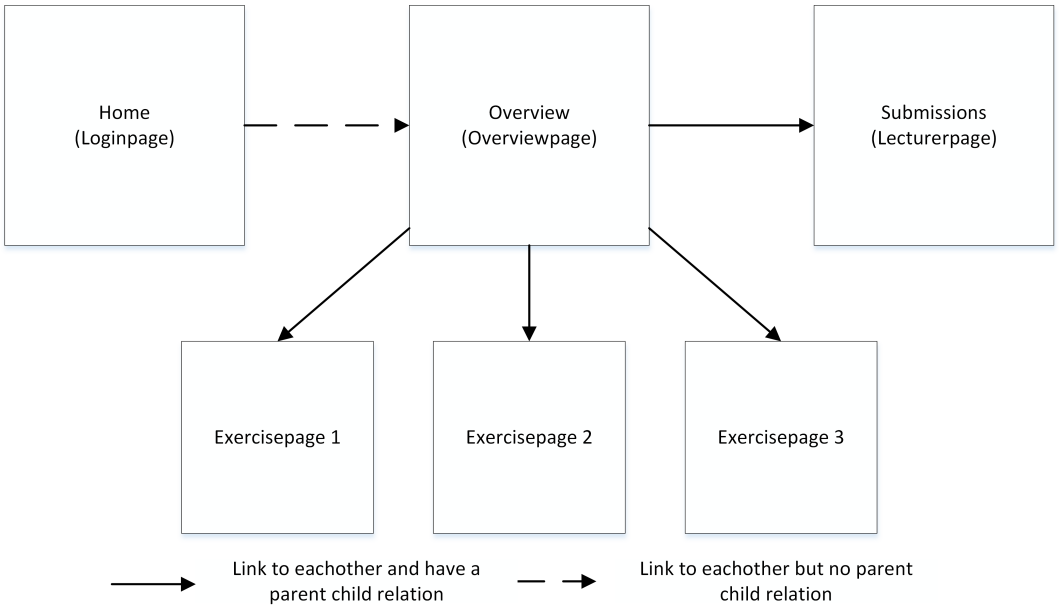


Figure D.1: A graphic representation of the hierarchy of page types within the website

Now we will continue to go in depth on each of these steps. Before starting, please go to the URL of your instance of the website (which you have defined during the installation) and append /wp-admin to this URL. Here you can log into the Wordpress administrator area with your admin account. This page will be the starting point for the rest of the guide.

Creating the login page

Once logged in the wp-admin area you can create a new page by hovering over the 'pages' menu item on the left and selecting 'new page'. There really is nothing to configure about the loginpage, all you have to do is set the template to 'LoginPage' on the bottom right of the page.

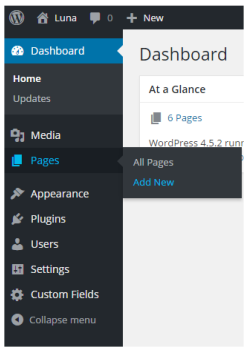


Figure D.2: Creating a new page in Wordpress

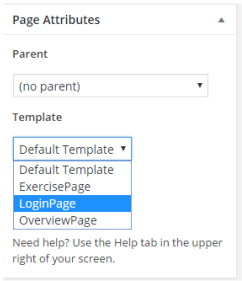


Figure D.3: Setting a template in Wordpress

Creating the overview page

In a similar fashion the overview page is created, the only difference being the template that needs to be selected is the OverviewPage template.

Creating the Lecturer page

Just like the overview and login page the lecturer page is created, needs no configuration except the template has to be set to 'LecturerPage'. The only difference between the other pages is that the parent should be the previously created overview page.

Creating an exercise page

Creating an exercise page requires some preparation before you can create it in Wordpress. Start off by collecting the input in separate .txt files and zipping these documents in a single zip archive. Make sure the file names are somewhat descriptive as these will be shown on the exercisepage and used as docIds in the MapReduce framework just like in Hadoop.
Once the zip is ready, create a new page, select the 'ExercisePage' template and make sure to also set the parent to the earlier created Overview page. Several new fields will now appear which need to

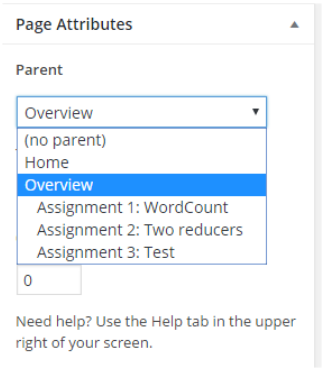


Figure D.4: Selecting the overview page as parent of the exercise page

be filled in. In the 'General' tab week number, amount of mappers and reducers can be filled in and input zip can be uploaded. A description of the exercise can be provided in the wysiwyg editor on the page. In the 'Editor settings' tab there are options to hide editors from view, pay close attention, if an editor is hidden, the default code set in the matching editors textarea will be executed when a student works on the assignment. The default code for each editor can be modified, these fields default to an empty function except for the combine and getPartition functions. The combine function defaults to the elementary combiner (which simply emits each key/value pair again) this function can be found in Appendix A. Finally the getPartition function defaults to the HashPartition as described in section C.1.2.

In the 'Student tests' tab tests can be added as a zip file. The zip should have the following properties: On root level in the zip there should be folders for each test. The name of this folder will be used as the name of the test. In each folder there should be 3 items, a .txt file named 'config', and two folders, one named 'input' and one named 'output'.

- test (Folder) - Mandatory
  - 'config.txt' (File) - Mandatory
    This text file should only contain two numbers, the amount of mappers on the first line, the amount of reducers on the second line. An example with 2 mappers and 3 reducers would mean the file only contains:
    2
    3
  - input (Folder) - Mandatory
    This folder contains all files that will be used as input for the test.
  - output (Folder) - Mandatory This folder contains the files that contain the output exactly seen as on the website. An important aspect is that each reducer outputs its own file. So for each reducer there should be a file named after the index of that reducer. So in the example with 3 reducers there would be 3 files in this folder: '0.txt', '1.txt' and '2.txt'.
    - '0.txt' - Mandatory
      This file (and the same file for higher index reducers as explained earlier) contain the output of the reducer with this index. the file contains key value pairs on each line, key and value separated by comma.
    - 'counters.txt' - Optional
      Finally there is one optional file with the same structure as the other files in this folder. It should only be present if counters have to be tested as well. Each line should contain

a counter name (ask the students to use this name in their exercises) and the value this counter should return. Like the key value pairs this should be comma separated. If for example a counter called `total` should have the value 420 then the file should contain total,420

When an assignment page is submitted, the zip gets parsed and the tests are added to the assignment data in the database. The student test data is downloaded once a student opens an assignment page.

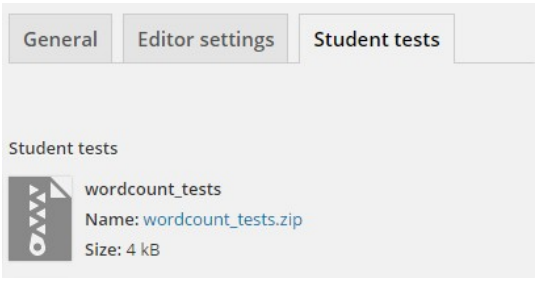


Figure D.5: Adding tests to an assignment

When students run or submit code for an assignment that has student tests, the student test framework runs these tests and presents the result to the student. When submit is pressed, the result for the tests are uploaded together with the student code to be viewed by the lecturer. The student code is tested for job output and counter output and students can see test results for both separately. Finally you

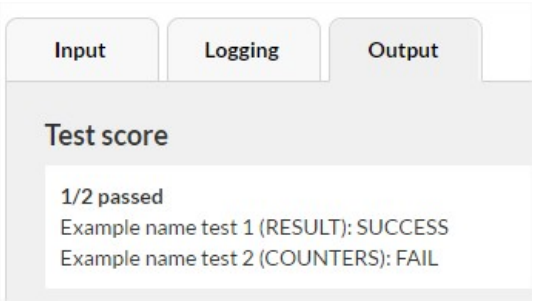


Figure D.6: Viewing test score, student view

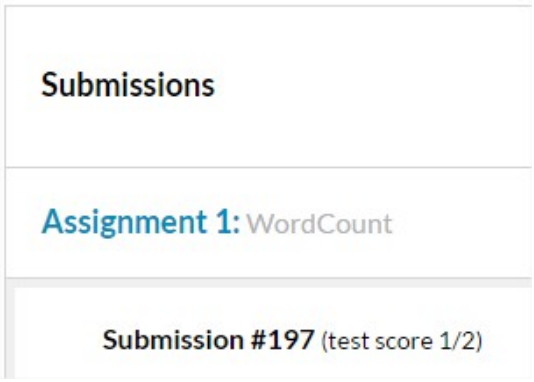


Figure D.7: Viewing test score, lecturer view

can also publish this page, it will also show up on the overview page under the selected week. From this point on more exercise pages can be added to ones liking.

D.3. Using the website

When a student visits Trifle for the first time, he or she will be presented with a login page on which only a captcha has to be filled in to sign in. Once that is done, a cookie will be set that will be valid for a maximum of 24 hours after inactivity. This way, students can use the site anonymously.

After signing in, students get redirected to the overview page on which two things can be done: sign out and selecting an assignment. Assignments are grouped by week number to keep the overview organised. When a student clicks on an assignment the student gets redirected to the exercise page of the assignment.

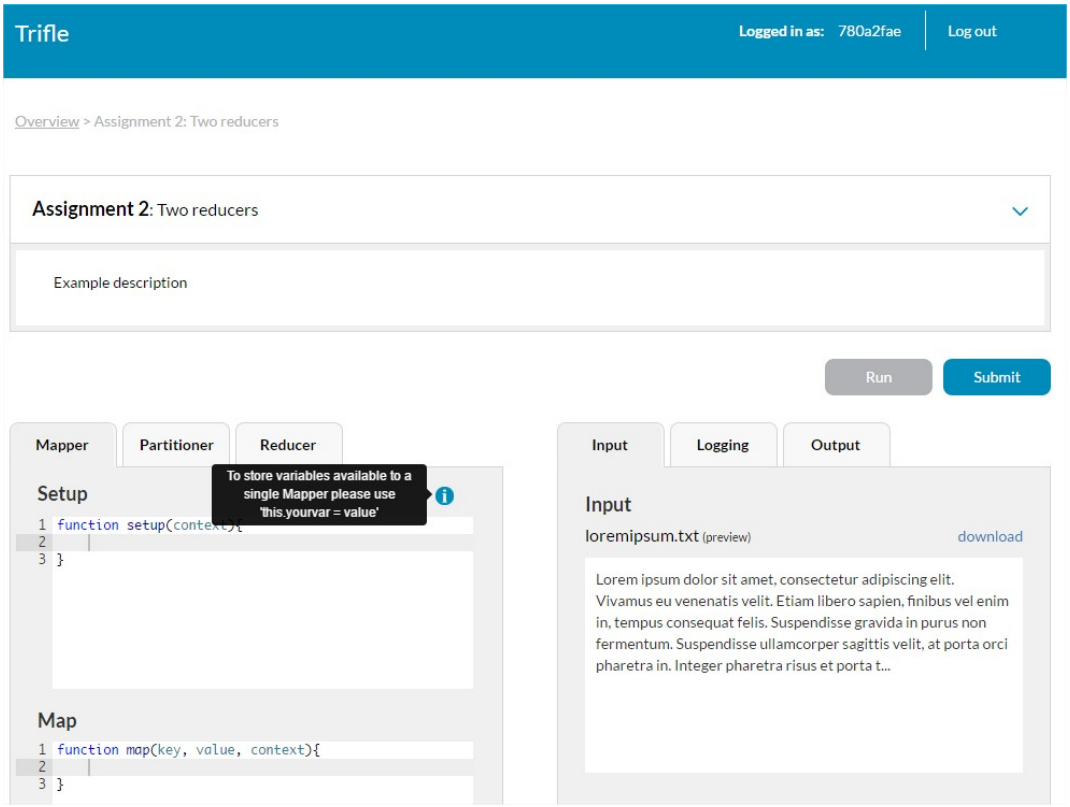


Figure D.8: Exercise page

Our user test has shown that the navigation is easy and Trifle is intuitive to use. Assignment description, test score and tooltips help students to understand what they are asked to do and complete an assignment even faster. Tooltips are shown when you hover over the '(i)' icon. Code for different parts of the system, input, logs and output can be viewed when you click on its designated tab. Jobs can be run when you click on the run button and code (together with the test score) can be submitted when clicked on submit. Input and output data can be downloaded by clicking the download button for students that want to see the data in a text editor of their preference.
Implementations must be written in JavaScript and there are only four non-standard JavaScript methods that have to be used.

- context.getCounter('test').increment(1); in order to increment the counter 'test' with one.
- context.write('abc', 5) in order to emit the key value pair 'abc',5.
- values.hasNext() in order to check if an iterator has any more values.
- values.next() to retrieve the next value of an iterator.

E

Project description

Hadoop is currently taught in the second year of the Computer Science Bachelor at TU Delft. Hadoop is one of the most popular frameworks for distributed processing on large clusters. Due to our setting (a university course) we have no access to a ”real” cluster and instead all lab assignments of the course are conducted through a virtual machine containing a single-machine Hadoop setup.

This setup has one main issue: some important aspects of Hadoop (e.g. no sharing of memory between Mappers and Reducers, or the non-default partitioning of keys) cannot be taught in the lab. While the students learn - in theory - how those issues influence programming in Hadoop, they often cannot apply these concepts as they do not experience these issues first-hand.

The project proposed here should SIMULATE the workings of Hadoop on a cluster. The goal is not to develop a fully-fledged Hadooop simulation, but to focus on those specific aspects of Hadoop that cannot be taught in the current VM setup. In additon, the solution should enable the lecturer to use the tool during in-class exercises for a more interactive teaching than is currently possible.

F

Raw benchmark results

	Thinkpad Yoga 1	8gb ram + i7 @ 2.4ghz				
	Case 1					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	587	198		293	230	
1	977	199		560	261	
10	8515	199		4688	553	
50	43168	655		24284	1.001	
100	87146	1414		-	-	partitioner thro
	Custom desktop	8gb ram + i5 @ 3.8ghz				
	Case 1					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	370	365		251	298	
1	669	404		388	299	
10	6368	513		3827	524	
50	32125	970		20972	956	
100	63621	1294		46008	1.613	
	Acer aspire v nit	8gb + i7 @ 2.6ghz				
	Case 1					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	463	104		284	224	
1	806	132		472	245	
10	7755	194		4787	484	
50	40463	890		25499	848	
100	80929	1300		58198	1623	

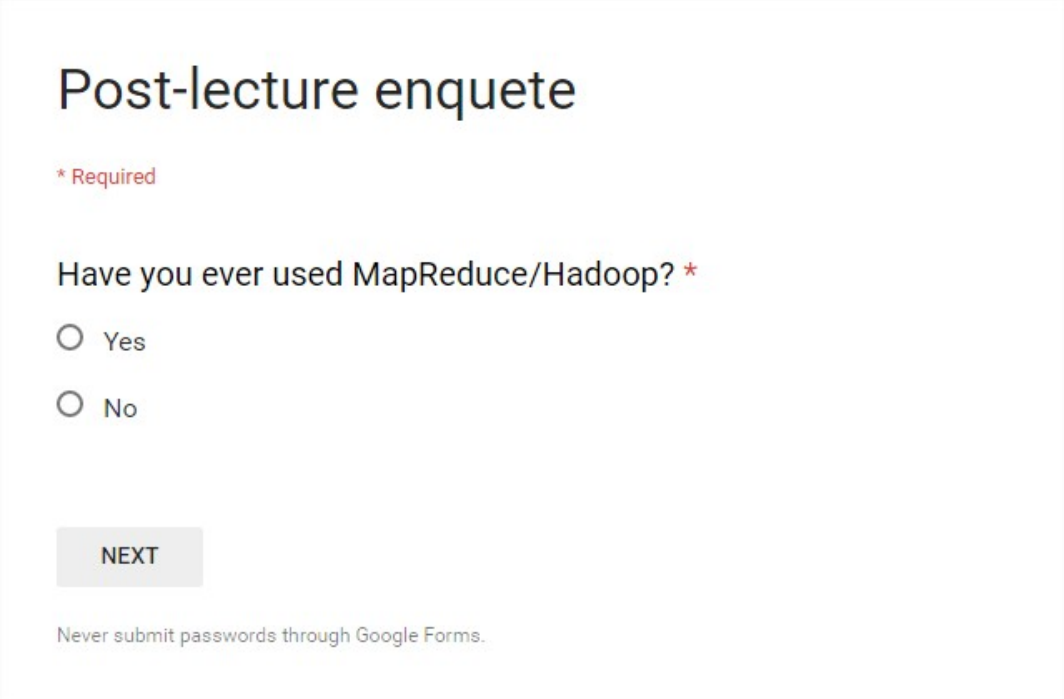
	Thinkpad Yoga 1	8gb ram + i7 @ 2.4ghz				
	Case 2					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	622	196	memory usage d	310	230	
1	996	224	memory usage d	501	456	
10	7379	248		4519	477	
50	41245	724		22885	967	
100	86480	1284	crashes with dev	47660	1302	
	Custom desktop	8gb ram + i5 @ 3.8ghz				
	Case 2					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	87	172		66	176	
1	129	155		103	195	
10	864	208		739	306	
50	4352	513		3715	507	
100	8406	821		7656	784	
	Acer aspire v nit	8gb + i7 @ 2.6ghz				
	Case 2					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage i	Note	Runtime in ms	Memory usage i	Note
0.5	168	104		112	166	
1	221	132		160	190	
10	1150	140		1023	275	
50	35576	580		4942	490	
100	10756	715		10403	1035	

	Thinkpad Yoga 1!	8gb ram + i7 @ 2.4ghz				
	Case 8					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	258	91		167	266	
1	304	88		221	253	
10	1486	122		1267	358	
50	6754	364		6347	687	
100	12757	637		12247	970	
	Custom desktop	8gb ram + i5 @ 3.8ghz				
	Case 8					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	91	141		88	193	
1	136	144		135	197	
10	974	235		1001	330	
50	4784	465		5045	637	
100	9349	833		10086	790	
	Acer aspire v nitr	8gb + i7 @ 2.6ghz				
	Case 8					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	157	108		109	132	
1	169	108		165	167	
10	1238	135		1331	330	
50	6281	380		6468	552	
100	10881	724		13242	1027	

	Thinkpad Yoga 1!	8gb ram + i7 @ 2.4ghz				
	Case 10					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	105	76		91	231	
1	175	88		126	211	
10	1389	158		946	358	
50	6723	360		4884	738	
100	13059	538		9030	720	
	Custom desktop	8gb ram + i5 @ 3.8ghz				
	Case 10					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	117	140		66	193	
1	163	145		93	199	
10	1001	212		627	320	
50	5175	427		3326	624	
100	9920	851		6867	965	
	Acer aspire v nitr	8gb + i7 @ 2.6ghz				
	Case 10					
	Chrome			Firefox		
Input size in MB	Runtime in ms	Memory usage ir	Note	Runtime in ms	Memory usage ir	Note
0.5	138	80		72	173	
1	207	95		130	180	
10	1176	164		721	290	
50	5738	360		3569	680	
100	10949	830		7381	950	

G

User test questionnaire



Post-lecture enquete

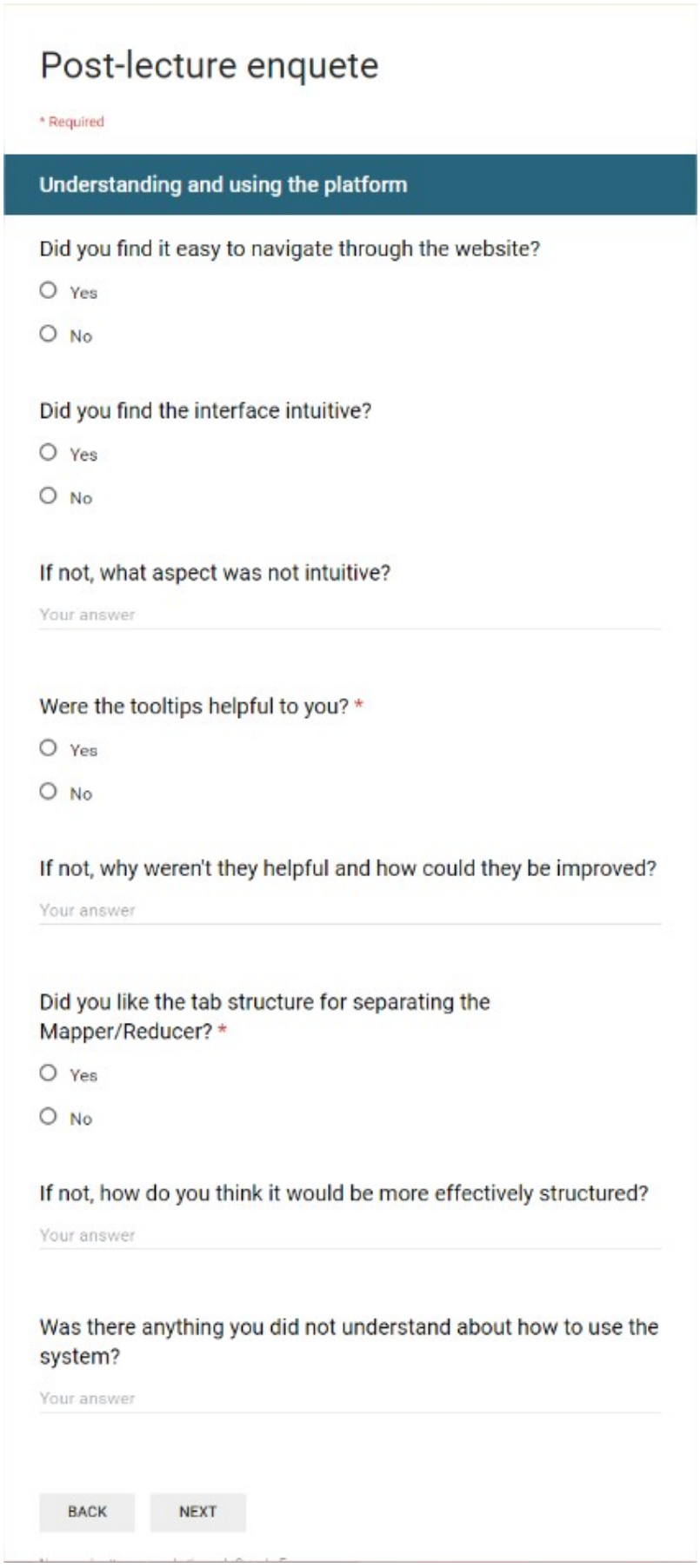
* Required

Have you ever used MapReduce/Hadoop? *

- Yes
- No

NEXT

Never submit passwords through Google Forms.



Post-lecture enquete

* Required

Understanding and using the platform

Did you find it easy to navigate through the website?

- Yes
- No

Did you find the interface intuitive?

- Yes
- No

If not, what aspect was not intuitive?

Your answer

Were the tooltips helpful to you? *

- Yes
- No

If not, why weren't they helpful and how could they be improved?

Your answer

Did you like the tab structure for separating the Mapper/Reducer? *

- Yes
- No

If not, how do you think it would be more effectively structured?

Your answer

Was there anything you did not understand about how to use the system?

Your answer

BACK NEXT

Post-lecture enquete

* Required

Running simulations

Were the error messages displayed clearly? *

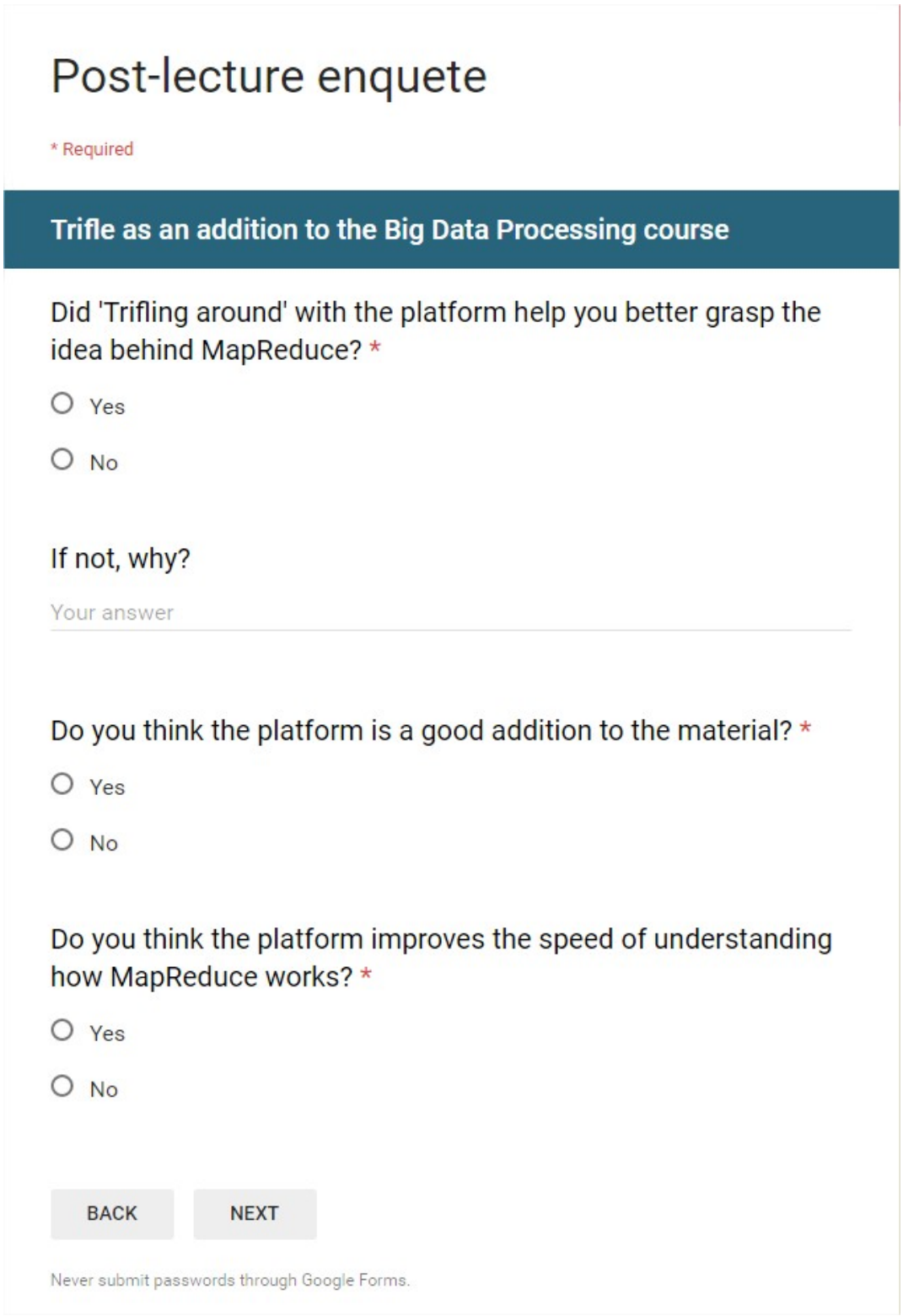
- ○ Yes
- ○ No
- ○ I did not have any errors

Was the performance of the simulations good on your machine? *

- ○ Yes
- ○ No

BACK NEXT

Never submit passwords through Google Forms.



Post-lecture enquete

* Required

Trifle as an addition to the Big Data Processing course

Did 'Trifling around' with the platform help you better grasp the idea behind MapReduce? *

- Yes
- No

If not, why?

Your answer

Do you think the platform is a good addition to the material? *

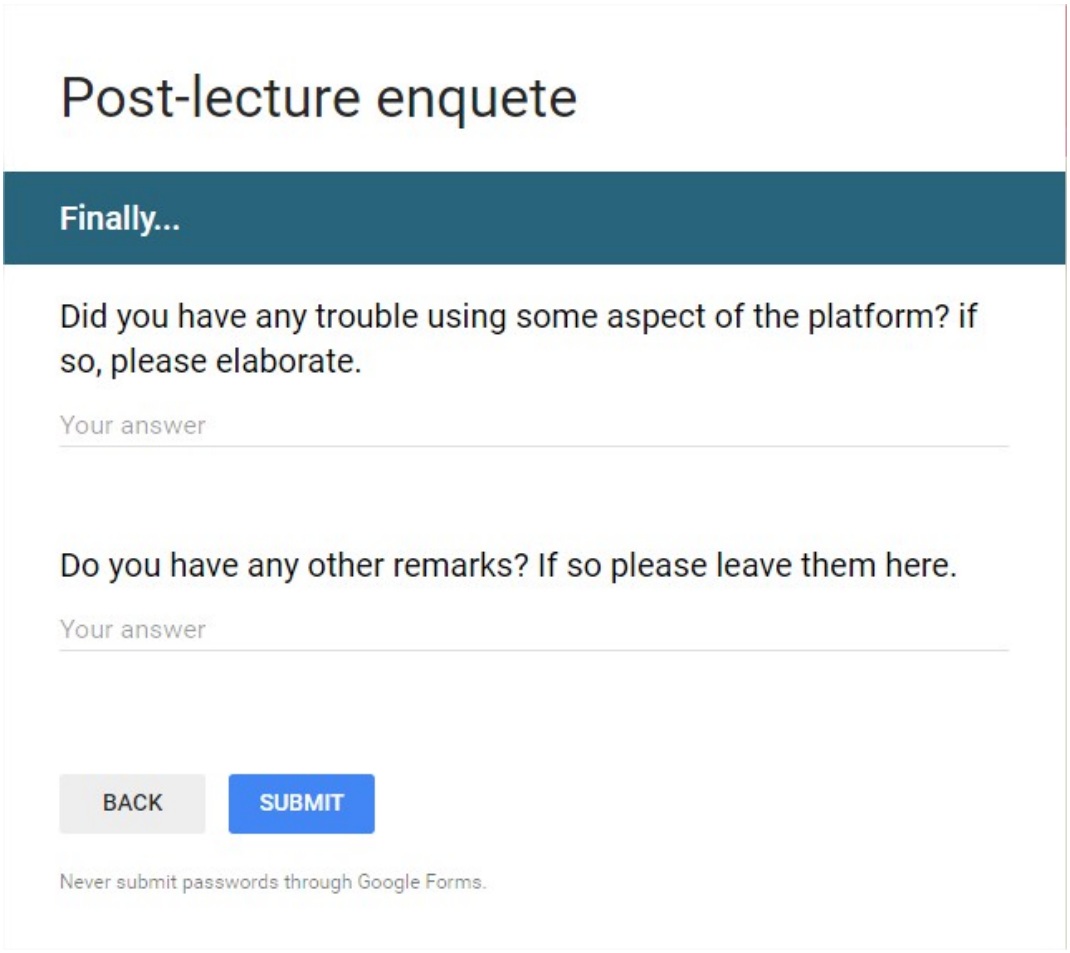
- Yes
- No

Do you think the platform improves the speed of understanding how MapReduce works? *

- Yes
- No

BACK NEXT

Never submit passwords through Google Forms.



Post-lecture enquete

Finally...

Did you have any trouble using some aspect of the platform? if so, please elaborate.

Your answer

Do you have any other remarks? If so please leave them here.

Your answer

BACK SUBMIT

Never submit passwords through Google Forms.

H

Infosheet

Title

Hadoop in the browser

Organisation

Web Information Systems

Date of presentation

24 June 2016 at 10:00

Description

The lecturer of the Big Data Processing (BDP) course approached us with the request to implement a tool that helps students understand the principles of MapReduce. At the moment some principles cannot be taught in currently used environments like the Cloudera Virtual Machine (VM). For practical assignments a Cloudera VM is used to start Hadoop jobs. This VM is slow, cannot be used interactive and it cannot be used to teach all MapReduce principles like true memory separation in a real cluster. In order to overcome these and other shortcomings we have looked at several options and decided to build a web-based simulation of MapReduce. This solution has several advantages over other options such as Cloudera. It runs on all popular Operating Systems without installation, enables the lecturer to give interactive lectures by getting real-time insights in students approaches to problems and allows teaching of some MapReduce principles that cannot be taught with currently available tools.

Members of the Project Team

Hasan Bilen

Hasan is a movie lover and a passionate Computer Science student building real-time back-end systems at a Dutch IT company. Hasan mainly focussed on the database, API and student testing.

Marc Zwart

Marc is a Computer Science bachelor student at the TU Delft. Working as a web-developer at Tam Tam, a Dutch digital agency alongside his studies. Marc was responsible for the user interface and setting up the Wordpress website.

Both team members have contributed to the MapReduce simulation framework, unit testing, preparing the report and final project presentation.

Coach:	Dr. Thomas Abeel	TU Delft
Client:	Dr. Claudia Hauff	TU Delft
Contact person:	Dr. Thomas Abeel	T.Abeel@tudelft.nl
Contact person:	Dr. Claudia Hauff	C.Hauff@tudelft.nl
Contact person:	Marc Zwart	marcdevin@hotmail.com
Contact person:	Hasan Bilen	hasanbilen38@gmail.com

The final report for this project can be found at: http://repository.tudelft.nl

I

SIG Feedback

In this appendix we shortly discuss the feedback received from SIG and what we did with this feedback.

I.1. SIG Feedback on code quality

This is the feedback we received from SIG, it is not written by us

De code van het systeem scoort bijna 3.4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size. Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de '$(document).ready'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '//make sure ace is loaded' en '//surpress warning in console' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen. Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden html bestanden, bijvoorbeeld tussen overviewpage.html, regels 1-19 en lecturerpage.html, regels 1-19. Dit soort duplicatie can opgespoord worden door het gebruik ven partials. In dit bepaalde voorbeeld kunnen jullie een header partial maken dat door de twee bestanden aangeroepen zou worden. Over het algemeen scoort de code gemiddeld, hopelijk lukt het om dit niveau te verbeteren tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

I.2. Taking advantage of the SIG Feedback

There was some constructive and some non-constructive feedback we received from SIG, as the feedback was written in Dutch we include a short list in English that summarizes the feedback:

- Unit size was too large (especially the document.ready() method)
- There is redundancy in the HTML files, like the header which is present on each page
- There was no unit testing found

So to start of, we had a critical look at out unit size, we drastically decreased this by splitting functionalities wherever possible. This did make our code a lot easier to read. The redundancy in the HTML files was a bit non-constructive as the HTML files are stored in the repository purely as a demo. This demo

is also a good backup in case something would happen to the wordpress templates. The redundancy SIG spoke of is not present in these templates. So we left the demo HTML as it is. The fact that no unit tests were found is quite flabbergasting since on the root folder of the repository we send to them there was a folder 'tests' which included two test suites, one QUnit suite which tested the framework and a Selenium suite which tested the interface's behaviour. So there was little to nothing to improve on this part, we did update the repositories README.md to instruct the reader to set up the wordpress website and go to the '[domainname]/tests' URL to run the QUnit test suite and instructions on how to run the Selenium test suite.

Bibliography

[1] Big Data Processing study guide, http://www.studiegids.tudelft.nl/a101_displayCourse.do?course_id=40230 (), accessed: 2016-05-01.

[2] TI2736-B: Big Data Processing, http://www.st.ewi.tudelft.nl/~hauff/TI2736-B.html (), accessed: 2016-05-01.

[3] J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Commun. ACM **51**, 107 (2008).

[4] Apache Hadoop, https://hadoop.apache.org/, accessed: 2016-05-25.

[5] Cloudera VM, https://www.cloudera.com/documentation/enterprise/5-3-x/topics/cloudera_quickstart_vm.html, accessed: 2016-05-25.

[6] S. Grissom and M. J. Van Gorp, A practical approach to integrating active and collaborative learning into the introductory computer science curriculum, in *Journal of Computing Sciences in Colleges*, Vol. 16 (Consortium for Computing Sciences in Colleges, 2000) pp. 95–100.

[7] J. J. McConnell, Active learning and its use in computer science, ACM SIGCSE Bulletin **28**, 52 (1996).

[8] T. Briggs, Techniques for active learning in CS courses, Journal of Computing Sciences in Colleges **21**, 156 (2005).

[9] L. Thomas, M. Ratcliffe, J. Woodbury, and E. Jarman, Learning styles and performance in the introductory programming sequence, ACM SIGCSE Bulletin **34**, 33 (2002).

[10] L. C. Kats, R. G. Vogelij, K. T. Kalleberg, and E. Visser, Software Development Environments on the Web: A Research Agenda, in *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2012 (ACM, New York, NY, USA, 2012) pp. 99–116.

[11] R. Reis, Pierre Volschenk-TP Msg.# 1146 Designing and Delivering Effective Lectures, (2012).

[12] N. McIntosh, Delivering effective lectures, JHPIEGO Strategy Papers (1996).

[13] WebLab: Learning Management System., https://weblab.tudelft.nl/, accessed: 2016-05-26.

[14] P. Garrity, T. Yates, R. Brown, and E. Shoop, WebMapReduce: An Accessible and Adaptable Tool for Teaching Map-reduce Computing, in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11 (ACM, New York, NY, USA, 2011) pp. 183–188.

[15] WebWorker API and browser compatibility, https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers, accessed: 2016-06-10.

[16] JavaScript browser compatibility, https://kangax.github.io/compat-table/es5/, accessed: 2016-05-21.

[17] WordPress CMS, http://wordpress.org/, accessed: 2016-06-16.

[18] Overview (Apache Hadoop Main 2.6.3 API), https://hadoop.apache.org/docs/r2.6.3/api/, accessed: 2016-05-30.

[19] Mapper (Hadoop 2.4.1 API), https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/Mapper.html, accessed: 2016-05-21.

[20] Partitioner (Hadoop 2.4.1 API), https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/Partitioner.html, accessed: 2016-05-21.

[21] Reducer (Hadoop 2.4.1 API), https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/Reducer.html, accessed: 2016-05-21.

[22] L. Li and W. Chou, Design and describe REST API without violating REST: A Petri net based approach, in *Web Services (ICWS), 2011 IEEE International Conference on* (IEEE, 2011) pp. 508–515.

[23] D. Crockford, The application/json media type for javascript object notation (json), (2006).

[24] J. Sutherland, N. Harrison, and J. Riddle, Teams That Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams, 2014 47th Hawaii International Conference on System Sciences **0**, 4722 (2014).

[25] HashPartitioner Hadoop, https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/mapreduce/lib/partition/HashPartitioner.html (), accessed: 2016-05-22.

[26] JavaScript hashCode implementation, http://werxltd.com/wp/2010/05/13/javascript-implementation-of-javas-string-hashcode-method/ (), accessed: 2016-05-22.

[27] reCAPTCHA: Easy on Humans, Hard on Bots, https://www.google.com/recaptcha/intro/index.html, accessed: 2016-05-15.

[28] Accordion (GUI), https://en.wikipedia.org/wiki/Accordion_(GUI), accessed: 2016-05-21.

[29] Ace - The high performance code editor for the web, https://ace.c9.io/, accessed: 2016-05-10.

[30] ACF | Advanced Custom Fields Plugin for WordPress, https://www.advancedcustomfields.com/, accessed: 2016-04-29.

[31] QUnit: A JavaScript Unit Testing framework., https://qunitjs.com/, accessed: 2016-06-16.

[32] Selenium - Web Browser Automation, http://www.seleniumhq.org/, accessed: 2016-06-16.