FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Using MapReduce in education and the development of tools for this purpose

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Informatik Didaktik

eingereicht von

Martin Dobiasch
Matrikelnummer 0828302

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: A. Prof. i.R. Dr. Erich Neuwirth

Wien, 30.09.2014 (Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Using MapReduce in education and the development of tools for this purpose

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Didactic for Informatics

by

Martin Dobiasch
Registration Number 0828302

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: A. Prof. i.R. Dr. Erich Neuwirth

Vienna, 30.09.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Martin Dobiasch
Ramperstorffergasse 46, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

I want to thank my supervisor A. Prof. i.R. Dr. Erich Neuwirth for the possibility to write this thesis by introducing me to this topic and helping me at critical points of my work.

Of course I want to thank my wonderful girlfriend Marion for all her help while writing this thesis. Thanks for all the countless hours of listening to my ideas and especially for supporting me with my sometimes poor language skills. Moreover, thanks for your humorous comments on my thesis.

Also I want to thank Dr. Brian Harvey from UC Berkeley for his interesting and helpful suggestions for my framework.

Moreover, I want to thank all the people reading and writing on the NetLogo mailing list and especially to Seth Tisue who invested his time to answer all my questions.

Last but not least I want to thank my parents for their continuous support of my studies.

Of course I also want to thank all the people I have not mentioned explicitly. I am thankful for all the help I have received.

Abstract

With growing amounts of data being processed efficient programs become more and more important. Over the last decade new frameworks have been developed to ease programming and avoid repetitive solutions to common problems. MapReduce is one of these frameworks which also received growing interest both in academic research as well as in commercial applications. Due to its structure MapReduce can also be used when teaching parallelism and abstraction in a computer science class.

One downside of MapReduce-frameworks is that they are currently mainly available for large clusters rather than schools with small IT-infrastructure or private end users. This results in missing possibilities for teaching the key concepts behind MapReduce. Another problem is the lack of courses publicly available for studying MapReduce. Most of the material on hand targets experienced programmers with deeper knowledge of a certain programming language such as Java.

This thesis first examines learning theories on their relevance for being used to teach MapReduce. The insights gained are then used in order to reason about criteria for a good framework in the context of teaching MapReduce. The thesis then presents a MapReduce-framework which is targeted to be used in schools for teaching programming novices. Additionally, some course material for a course on MapReduce is provided.

Kurzfassung

Durch immer größer werdende Mengen an Daten, die täglich verarbeitet werden, steigt der Bedarf an effizienten Programmen immer mehr. Über die letzten Jahre hinweg wurde eine Vielzahl an Frameworks entwickelt, um die Programmierung zu vereinfachen, aber auch um die ewige Neu-Erfindung des Rades bei verbreiteten Problemen zu verhindern. MapReduce ist eines dieser Frameworks, welches zunehmend Beachtung von sowohl der Wissenschaft als auch der Wirtschaft bekommt. Durch seine Funktionsweise kann MapReduce auch benutzt werden um, Parallelität und Abstraktion zu unterrichten.

Ein Nachteil der meisten MapReduce-Frameworks ist, dass sie nur für große Cluster und nicht für Schulen mit kleiner IT-Infrastruktur entwickelt worden sind. Als Konsequenz daraus sind die Möglichkeiten der Vermittlung der Konzepte hinter MapReduce stark eingeschränkt. Ein weiteres Problem ist, dass es nur wenige bis keine frei zugänglichen Kurse zum Thema MapReduce gibt. Das meiste an verfügbaren Materialien ist auf erfahrene Programmierer ausgerichtet, welche schon ein breites Verständnis einer Programmiersprache wie Java besitzen, und nicht, wie wünschenswert, auf Programmieranfänger zugeschnitten.

Die vorliegende Arbeit diskutiert in einem ersten Schritt Lerntheorien und deren Relevanz für die Vermittlung von MapReduce. Die gewonnen Erkenntnisse werden dann verwendet, um Kriterien für ein Framework, welches im Unterricht eingesetzt werden kann, aufzustellen. Im Anschluss wird ein MapReduce-Framework präsentiert, welches darauf ausgerichtet ist im Schulunterricht verwendet zu werden. Zusätzlich werden einige Unterlagen für einen Kurs zum Thema MapReduce vorgestellt.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

With growing amounts of data being processed efficient programs become more and more important. One way to write efficient programs nowadays is to use approaches which can work in parallel. However, it is not always easy to write parallel programs since concurrency issues arise once an algorithm or a program exceeds a certain basic level of complexity. Some programs are even too large for being processed on a single computer and thus need to be computed on a cluster. Moreover, in order to be even more efficient, distributed data or distributed file systems, respectively, are often required. As an additional problem some programs are also required to scale automatically or at least “easily” with growing amount of input data. All of these requirements add additional work and complexity to all stages of the development and maintenance of a program. What is more, it means that most of the tasks are similar between different programs. As a result of this, these tasks either have to be reimplemented over and over again or should be part of a framework.

MapReduce [11] - as will be described in the next chapter - is a framework or programming model designed to process large amounts of data, tackling a lot of problems mentioned above. It is designed to take care of almost all of the aforesaid tasks so that the programmers and analysts can focus on the main tasks of the program. Another aspect of programs that MapReduce takes into account is that most computations share a certain structure: First data is processed in order to generate intermediate values. These intermediate values are then sorted and grouped before another computation is performed on every of these groups in order to obtain the final result. As a result, programs using MapReduce just have to provide the computations on the data while all other tasks (spreading data, handling failures, ...) are handled by the framework.

One downside of the richness of MapReduce-frameworks is that they are currently mainly available for large clusters rather than schools with small IT-infrastructure or private end users. Moreover, some of the frameworks are merely available commercially. This results in a lack of possibilities for teaching the key concepts behind MapReduce. Another problem is the lack of courses publicly available for studying MapReduce. Most of the material available targets experienced programmers with deeper knowledge of a certain programming language such as Java. Additionally some courses also require access to a cluster and additional skills such as shell scripting. Thus, it is not easy for beginners to learn how to write MapReduce-programs. Moreover, although MapReduce could be used when teaching parallelism and abstraction in a computer science class, the barriers and mentioned problems make it almost impracticable to use MapReduce in a class room.

This thesis aims to overcome two problems: First the lack of resources for programming novices on how to use MapReduce and secondly the lack of MapReduce-frameworks tailored for educational purposes.

1.2 Aim of the Work

The goal of this thesis is to design a framework tailored to the needs of a course teaching MapReduce. Moreover, the framework will be demonstrated by providing sample material for its usage when teaching MapReduce.

Some key requirements for the framework include

- Easy setup - the installation framework should be very easy and only involve a small number of steps. Additionally, there should not be a need for editing configuration files and dealing with IP-addresses and the like.

- Lightweight - the framework should stick to basic functionality instead of overwhelming potential students with its numerous possibilities.

- Standalone - it should be possible for students to develop and test MapReduce-programs on their own PCs without the access to other PCs or a cluster.

All provided examples and course materials are designed for programming novices and do not require long experience with programming and parallelism in specific. However, in order not to exceed the scope of this thesis no programming tutorial will be given; nevertheless, the teaching material provided requires some basic programming skills and especially a basic knowledge of NetLogo [41]. To give an estimate for the knowledge required to understand the material provided with this thesis: a student should be able to deal with variables and write if-statements on their own.

1.3 Methodological Approach

1. **Literature Review:** In a first step I gathered background information about MapReduce and learning theories through a thorough review of existing literature. During this literature review, I identified different MapReduce-frameworks and studied their similarities and differences. By doing so, I identified central features of these frameworks. Later on, I decided which of these features have to be supported by the framework created in this thesis.

2. **Sample-Problems:** In a next step I defined a set of problems suitable for teaching MapReduce. These problems represent different problem classes like graphs, machine learning problems, etc.

3. **Implementation:** Based on the insights gained from the literature review I created a MapReduce-framework for NetLogo providing all relevant features relevant for teaching MapReduce.

4. **Course Material:** In the last step I facilitated the framework to solve the identified sample problems in order to create sample material for a course on MapReduce.

1.4 Structure of the Work

Chapter 2 gives a detailed introduction to MapReduce. Included in this chapter is a list of available MapReduce frameworks and distributions. *Chapter 3* describes cognitive learning theories and their relevance for teaching MapReduce. *Chapter 4* investigates the requirements for a MapReduce-framework tailored for teaching and compares existing frameworks with these requirements. *Chapter 5* presents the framework created as part of this thesis and describes its development process. *Chapter 6* presents the course featuring the created framework, demonstrating the functionality of the framework and providing code examples. Moreover, it shows approaches of how to include these examples in a course teaching MapReduce by providing tasks for students. *Chapter 7* presents a summary, some concluding remarks and an outlook to future work.

CHAPTER 2

About MapReduce

2.1 A Short Introduction to the History of MapReduce

Around 2004 two scientists at Google proposed a framework called MapReduce for processing "large amounts of raw data" [11] on a cluster. The reason for this framework was that over the years before the invention of MapReduce programmers at Google had written numerous programs for analysing large amounts of data (for example logfiles) or performing computation on these data sets. The problem the programmers at Google encountered most of the time - and all other programmers writing such programs - is that when data increases it is difficult to distribute the computation on several computers or a cluster in order to get a result in a reasonable amount of time. In other words, scaling was a serious issue for the engineers at Google. In their paper the authors described it in the following way: "The issues of how to parallelise the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues" [11]. As a result, the framework, then developed by the scientists at Google, takes away the work of writing code for distributing the computation on a cluster from the programmer. The programmer only has to write the so-called map and reduce routines, while the framework deals with the distribution of data and code, the parallel execution, the gathering of results and so on. MapReduce as proposed by Google is not available in public, but there exist various open-source implementations like - probably best known - Hadoop [17]. Figure 2.1 shows the growing interest in MapReduce and Hadoop, while other technologies like OpenMP start losing "impact" on the community. The graph in the figure highlights the search volume at Google for the terms "mapreduce", "hadoop" and "openmp" over the timespan from 2004 to 2012. After the introduction of MapReduce in 2004 Google is claimed to have rewritten over 80% of its data-processing tasks for using MapReduce [34].

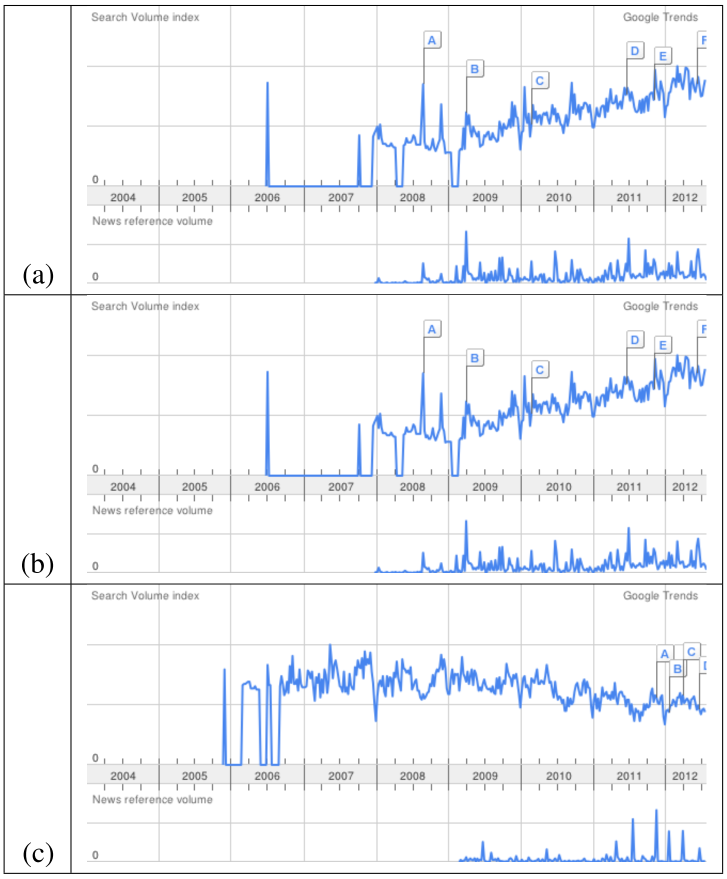

Figure 2.1: Google trends for (a) "mapreduce" (b) "hadoop" and (c) "openmp". Graphic generated with Google Trends `www.google.com/trends/`

2.2 Terminology

Before presenting more details about MapReduce, I will introduce some terms which will appear quite frequently in this thesis. All of them are used in descriptions of the framework or algorithms using the framework. It is important to distinguish and give precise definitions for terms like jobs and tasks in order to deliver a legible and understandable work. What is more, the following definitions are useful for future teachers of MapReduce, because they will give them the ability to be technically precise when teaching or preparing a course, or even when preparing such a course.

Master

Controls the execution of the program. The master is the part of the software package responsible for all scheduling-decisions, distribution of data and so on. Furthermore, it decides how to partition the input data. In most MapReduce distributions the master is a daemon-process to which the user can submit jobs directly or via another program like a jobtracker.

Worker

Executes tasks. A worker receives information from the master about what to execute with which parameters. It informs the master about its status and produced results. Besides setting up the worker once and maybe starting it later on, the user usually does not have to make interaction with workers. They run without the need of attention from the user, being only controlled by the master.

Task

A single atomic execution of a map or reduce function. A task is executed by a worker. It can be a run of a map-, reduce- or combine-function and its according input as $(key, value)$-pairs. The user does not have to bother with executing the tasks and passing parameters to them. The master schedules where and when to execute which task with which parameters or data.

Job

A MapReduce program. A job is a complete map-reduce 'run'. If a program is based on an iterative MapReduce-approach - i.e. it involving multiple runs of MapReduce - there is no commonly agreed definition whether the whole program as such is a job or only one single run of MapReduce. In most cases the reader will get the idea which case is used from the context. In this thesis I will try to be as consistent as possible and call one run of map and reduce a job. To be even more precise, one iteration of MapReduce will be denoted as a job.

Node

One computer. A node is usually one computer in the cluster. This computer can be physical or virtual[1]. Usually each node has its own hard-disk CPU and main memory available exclusively[2] for computation.

2.3 Programming Model

The main idea behind MapReduce is inspired by routines known from functional programming languages like Lisp [11]. This functional approach is due to the fact that most of the computations done at Google involved applying a function to all records of a data set (map function) and then 'combining' the results (reduce function) in order to get a final result.

[1]A virtual node could be a virtual machine.
[2]Exclusive here means that no two nodes share the same CPU, hard-disk or main memory.

Execution Process

"The computation takes a set of input key/value pairs, and produces a set of output key/value pairs" [11]. In a first step the user data is passed to the map function as $(key, value(s))$-pairs. Every execution of this map function will then produce zero or more intermediate $(key, value)$-pairs. "Then, the system automatically performs a group-by operation on the intermediate key" [9]. For every intermediate key I the reduce function is executed with "a set of values for that key" [11]. Every execution of the reduce function produces zero or more final $(key, value)$-pairs. These results (the $(key, value)$-pairs) are then collected and stored as result-file on the file system. The programmer usually provides only the map and reduce functions to the framework. Additionally, a configuration of a MapReduce job can be provided for the framework in order to accelerate the computation. Among others, the following parameters can be configured:

- Number of used machines
- Number of tasks
- Memory used for one task
- Partitioning-function

The possible acceleration is due to the fact that the framework has no a priori knowledge about the input-, intermediate- and result-data, and the map and reduce functions. In figure 2.2 the run of a MapReduce-program is illustrated:

1. The program is distributed to the various nodes
2. The master assigns the tasks to the workers
3. Every task is computed by a worker reading the partition of the task (denoted as 'split') and
4. produces some output written on a local disk on the node.
5. The - previously assigned - workers execute the reduce tasks reading the data written before by the map tasks and
6. write the output or the reduce tasks to the disk

Fault Tolerance

One key-feature of MapReduce is fault tolerance. Since the framework is designed to run on a cluster with "hundreds or thousands of machines" [11] which are not considered to be high end machines with a high uptime, it needs to keep track of failing workers. Basically, there are two failures that can happen:

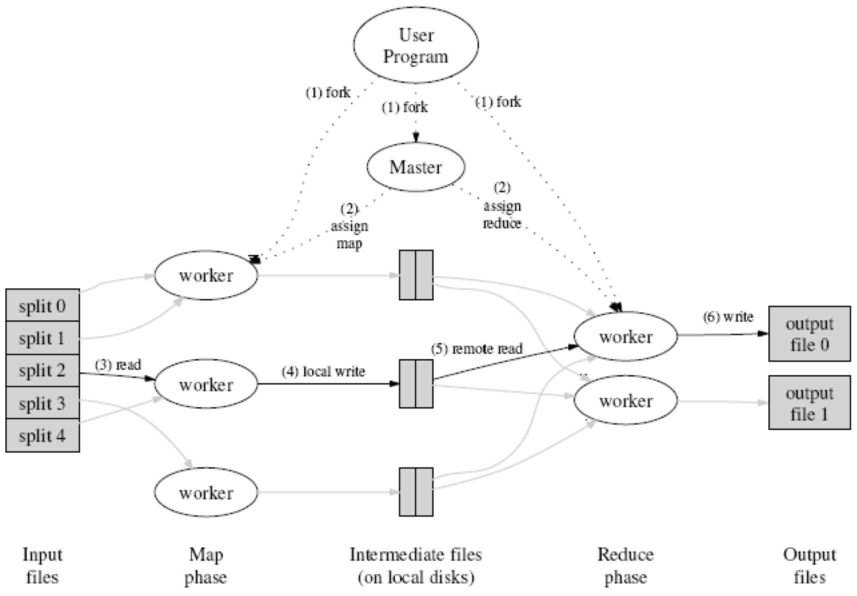

Figure 2.2: MapReduce, taken from [11]

A Worker fails: A machine or task executing a map or reduce function fails. In a normal program this would be very hard to handle in the code. In a MapReduce-program, however no extra code has to be written. The master keeps track of all running tasks and thus can restart a task when it fails. More sophisticated implementations can even skip bad records (see below).

The Master fails: When using some kind of book keeping where the master periodically writes 'checkpoints' to a log file. Thus the master can be restarted at the last 'checkpointed' state. It should be pointed out that, given the fact that there is always only one master, failure is unlikely [11]. In another approach a backup master could be used where, if the original master fails, the backup master takes its place and restarts the computation at the last 'checkpoint'. The problem with this approach is that the "masters" have to communicate with each other all the time in order to provide the backup functionality. This constant communication could decrease the overall performance of the cluster.

Bad Records

It is common for programs to have bugs. Sometimes a program crashes deterministically for given inputs. The usual approach to handle discovered bugs is to fix them, which is not always feasible. Big applications using - closed source - third party libraries are one example for this. A bug triggered by a third party library cannot always be fixed, since the source code might not be accessible. Another case where fixing bugs might not be feasible or necessary for certain records is statistical analysis for large data sets, which also might include incomplete records

[11]. MapReduce can skip such bad records: The workers can be equipped with a feature to detect if a task has failed. If a task has failed, the worker which has executed this task informs the master that this particular task has failed and reports the last record. The master would then restart the computation. If the master detects two or more failures for a particular record, it can skip the record from the computation.

Combiner Function

Every map function can produce several intermediate keys. For every distinctive key a worker node will be assigned, causing all the corresponding data to be sent from other nodes to this node. This will limit the execution time, since the amount of data to be sent across the network is increased. If a reduce function is associative and commutative a partial reduce function can be applied to the intermediate results stored on every node before shifting them to other nodes [9]. These intermediate reduce functions are called combiners. "Partial combining significantly speeds up certain classes of MapReduce operations" [11], since the amount of data to be sent across the network is decreased. Usually the combiner and reduce function do not differ.

Single-Node or Local Execution

Debugging a distributed application can be very tricky given the fact that it involves getting access to all included nodes. This could be very hard on a cluster since the decisions of the master which node to use are not always deterministic. Moreover, most programmers do not have access to a cluster in order to test programs during (early) development stages. Thus most MapReduce-libraries include a feature to run MapReduce on a single node in order to have all computation done on computer. Most implementations also allow running a computation in sequential mode so that all tasks are executed one after another.

Backup Tasks

One of the common causes that lengthens the total time taken for a MapReduce operation is a "straggler" [11]: One task takes longer than all the others, causing the whole computation to wait for this task to finish. There could be various causes for this behaviour:

- The machine could have a hardware failure, for example disk errors, causing the reading performance to slow done significantly. Another example are network errors like the sudden disconnection of a node.

- Work load on the machine: The scheduling could prioritise other computations and thus cause the task to decelerate

In order to avoid waiting times when a MapReduce program stage is close to its completion, the master will schedule redundant backup executions of those tasks still in stock to be completed. The first of these redundant executions 'wins', causing the other copies to be terminated.

Distributed File System

As stated before network bandwidth is a limiting factor for computation on clusters. Thus most of the MapReduce-distributions include a distributed file system. This 'addition' is not surprising since MapReduce is a 'standard' for a computation-model on top of a distributed data storage [11]. These file systems store files redundantly on the cluster and split them into smaller pieces, called chunks, which are replicated over the nodes in the cluster [18, 36]. One obvious reason for this replication process are reliability issues. This means that a node failure - which is considered to be likely - is not going to harm the whole system. A second, more important reason is a better performance when using distributed computing. These file systems are usually tailored to the needs of MapReduce: "large streaming reads and small random reads" are best optimised on these file systems. [18] The number of replicas can be fixed for a file system. If there are more nodes than replicas needed, then the distributed file system also increases the data storage space available for computation. On a real world cluster it is obviously the case that there are more nodes than replicas, since a default value for replication count is three replicas on a cluster [18]. The whole data-network is usually managed by one or more masters. Those masters[3] additionally store meta-data about the files [36]. For HDFS the terminology is master and data-node, for GFS it is master and chunkservers. On a cluster one node - or more in case of multiple master nodes for the file system - is the master node for the file system; all other nodes are data-nodes/chunkservers.

Programs do not access data-nodes directly. A program can query the master and obtain information about the locality of the chunks of a file. Thus the MapReduce-master can schedule the execution of tasks, using a file, in a way that a task using a particular chunk of a file is executed on the corresponding machine. In case this is not possible it will try to schedule the task in such a way that the 'transport-costs' are low [11, 36]. So normally "most input data is read locally and consumes no network bandwidth" [11].

2.4 Criticism About MapReduce

Critics from a Database Perspective

There have been several critics who said that MapReduce is neither useful nor novel at all. Some of these critics had 'database-background'. David DeWitt stated the following points of criticism concerning MapReduce in his article "Mapreduce: A major step backwards" (see [12]):

1. A giant step backward, in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all – it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS

[3]Do not confuse these masters with the master of MapReduce.

5. Incompatible with all of the tools on which DBMS users have come to depend on

[12]

It has to be noted that this critique is stated in a way assuming that MapReduce concerns "the database community" [12]. With regards to the first statement I have to agree with the author that when it comes to writing big application suites MapReduce does not offer as much possibilities as a DBMS does. On the other hand, it is not aiming assisting programmers with persisting data. At present (2013) my opinion is that MapReduce lacks the possibility of persisting data to a DBMS once a job is done. At the moment the programmer has to find his or her own way of how to store data into a cluster and load it back from a cluster once the computation is done. Still, most of the applications of the MapReduce-model are data applications where no direct access to a DBMS is needed.

The second statement refers to the fact that "all modern DBMSs use hash or B-tree indexes to accelerate access to data" [12], whereas MapReduce lacks such a feature. Still, it is the case that MapReduce does not need to have indexes and other mechanisms for accessing data, since the whole input is always processed or at least there is no searching in the data. This is an aspect not mentioned by DeWitt. Instead the data is partitioned into smaller pieces for the mapping stage. These partitions are then processed individually. This fact probably makes B-tree indexes an unnecessary overkill.

Relating to the critique that MapReduce is not novel: Yes, that is true. In [11] the authors state that it is not completely new, but rather an abstraction which "is inspired by the map and reduce primitives present in Lisp and many other functional languages" [11]. All common functional programming languages possess two build-in functions which are commonly called map and fold[4]. Map applies a function to all elements in a list. Fold successively applies a function to the elements of a list: It starts with an initial value and applies the (two-parameter) function to the first element of the list. The result of this is then taken as first argument for the application of the function together with the second element of the list. This continues until all elements of the list have been consumed. If a fold operation f is associative, i.e., computing $\texttt{fold}(f, \{a, b, c, d\})$ is the same as computing $\texttt{fold}(f, \{f(a, b), f(c, d)\})$, then the computation can be easily parallelised. "Typically, map and fold are used in combination" [25].

In his fourth statement the author claims that missing features in MapReduce are included in modern DBMSs such as "Bulk loader, Indexing, Updates, Transactions, Integrity constraints, Referential integrity, Views" [12]. As stated before some of these features are missing since there is no convenient way of handling the data besides performing computation on it. Still, this concerns more the underlying file system and not MapReduce as such. So this point of criticism is probably addressed to the wrong point since it does concern the concept of MapReduce as such but only the known implementations.

His statement that "MapReduce is incompatible with the DBMS tools" [12] is true. As a counter-argument it can be stated that MapReduce is not meant to be the 'keeper' of data and thus there is no intended way of using DBMS tools. The normal way of using the framework is to transfer data to a cluster, perform computation on it and transfer the result back from the

[4]Other names for fold are: reduce, accumulate, compress, or inject.

cluster. So there is no real way where analysis tools and similar devices could be plugged in. The normal way would be to transfer the resulting data (back) to a RDBMS and use DBMS tools.

Summing up the pros and cons about [12], the article points the reader to a wrong direction - MapReduce and RDMS are intended to do the same thing - whereas the arguments of the author are right. Besides the fact that the author tries to compare two different tools like apples and oranges the article makes some good points on missing features on the file system.

API

One major problem with MapReduce is that, since it is not a library as such but a "way how to program", there are many different frameworks and libraries. Thus most of the APIs of the different implementations are generally not compatible with each other. This means that a program written for Hadoop is not going to run without major modifications on a "Google-System". Therefore, the programmers are bound to one specific implementation of MapReduce. Another problem with those APIs is that most of them are rather complicated. This causes difficulties for novice programmers to learn programming with “the map-reduce paradigm” which requires an understanding of APIs. This (too) high complexity of the APIs should be tackled with this work! A reason for the diverging APIs lies in the “paradigm-part”. Every distribution/implementation is an interpretation of the MapReduce-paradigm as proposed in the original paper. This means that since there is no ’defined’ underlying distributed file system, there cannot be a defined way of accessing it and thus no defined API functions. This also leads to differences in other parts of the API.

2.5 Comparing MapReduce to Other Frameworks and Tools

MP

MP stands for Multiprocessing. An open source standard and implementation of MP is OpenMP. “OpenMP is a shared-memory application programming interface (API)” [8]. It works by ‘extending’ a programming language[5] with elements in order to make a sequential program run in parallel. Shared memory in this context means that multiple processors or processes use the same memory. When using OpenMP the programmer only has to add directives to the source code in order to give hints to the library on how the work can be spread among various threads and now it can synchronise those threads. OpenMP-programs tend to be highly readable in a ‘sequential way’.

On an interesting side note I want to point out that OpenMP has a directive “reduction” which (simplified) tells the library about a block that can be parallelised like an application of a “fold operation” known from functional programming languages.

OpenMP is targeted to run on shared-memory systems. Thus, it is impossible to compare OpenMP with MapReduce since they tend to achieve different things. While the former is useful when trying to ‘make a program run faster’, the latter is a paradigm for parallel computing.

[5]C,C++ and Fortran

Moreover, it has to be pointed out that OpenMP is not intended to be used for computation on clusters, since it cannot distribute the computation to several computers. ‘Classical MapReduce’[6], on the other hand, targets to perform the computation on several computers connected to form a cluster. Nevertheless, there are also MapReduce-implementations for shared-memory systems. One example for this is Phoenix, which was developed at Stanford[7] [33]. Additionally, there is an Intel-based extension "Cluster OpenMP", which targets the problem of using a cluster for computations when programming with OpenMP without the requirement to rewrite the entire program.

MPI

MPI stands for Message Passing Interface. Like MP it is a standard for parallel programming, but intended to run on distributed memory systems (for example a cluster). Like MapReduce it removes the need of implementing routines for communication between the computers. One big difference is that MapReduce also includes a distributed data storage, whereas MP does not deal with any data. MPI-programs tend to be very complex and hard to understand, and has to set up the communication between the processes in order to distribute data and collect results. One advantage of MPI over MapReduce is that it is already commonly used on clusters while still not all clusters are equipped with a MapReduce-library. The VSC[8], for example, has various MPI-versions installed but no MapReduce-library.

RDBMS - Relational Database Management System

“Though it may seem that MR and parallel databases target different audiences, it is in fact possible to write almost any parallel processing task as either a set of database queries (possibly using user defined functions and aggregates to filter and combine data) or a set of MapReduce jobs” [32]. This is due to the fact that the languages used for MapReduce are Turing complete and so are certain SQL languages like PL/SQL [7]. One key difference between the two systems is that databases require the data to be in a structured form, whereas for MapReduce data can be in arbitrary form. Another big difference between the two approaches is that at the moment there are no "usable" open source parallel RDBMS available. MapReduce was developed originally to help processing web data, which is usually not structured. This is probably why MapReduce-data is not structured, which can be a benefit over RDBMSs or a disadvantage depending on the field of application. For example when having to deal with complex data it is often necessary to write a custom parser in the MapReduce-program in order to be able to use it. One advantage of MapReduce over parallel databases is its way of dealing with failures. A failure normally does not produce any big harm to the overall computation since the master just reschedules the failed tasks. On the other hand, if a node of a parallel RDBMS - which splits up the computation into transactions - fails, the whole transaction has to be restarted. [32] Probably a disadvantage of being new to the market and thus being under heavy development like MapReduce is that in some frameworks API-functions get deprecated during release upgrades. This can make the

[6]as proposed by Google in [11]
[7]Google has its origins at Stanford!
[8]Vienna Scientific Cluster, `http://vsc.ac.at`

maintenance of a pure database application easier. On the other hand, from a maintenance perspective parallel RDBMS tends to be harder to install and configure than for example a Hadoop Cluster (as stated in [32]).

2.6 Some Implementations and Distributions

In this section some of the currently available implementations and distributions of MapReduce are listed and described. It has to be noted, however, that such a list can never be conclusive. Some of the frameworks are analysed in more detail in section 4.2.

Google

Since researchers from Google have invented MapReduce as described in this thesis, their framework is the first implementation. One downside is that Google does not give much insights to its implementation of MapReduce. Still it is considered to be the best implementation although neither the source code nor the API are available for the public. Furthermore, Google holds the patents for MapReduce (see 2.7 for more details).

Hadoop

Hadoop is probably the best known open source implementation of MapReduce. It is currently licensed under the "Apache License 2.0". The project commenced as a fork of a project called Lucence[9], which is similar to the ideas of MapReduce. Most of the source code of Hadoop is written by people associated with Yahoo!, which is one of the biggest users of Hadoop. It is also interesting to know that most of the MapReduce-distributions use Hadoop as their underlying technology.

Cloudera

Cloudera Inc. is not a MapReduce-implementation as such but a distributor of MapReduce and MapReduce-related software and services. Moreover, the company contributes back to the Hadoop-core. Additionally, Cloudera ships the source code of Hadoop with its distribution.

MapR

Like Cloudera, MapR does not distribute its own MapReduce-implementation but is developing and selling Hadoop-related software. One difference of the distribution to other distributions is that it is using NFS instead of HDFS. Amazon has added MapR as an option to its "Amazon Elastic MapReduce"-service.

[9]`http://lucene.apache.org/`

Aster Data

Aster Data is a company focusing on analytic database management systems. Their main product is a cluster-software called nCluster. This cluster also comes with a Hadoop installation. In order to use Structured Query Language (SQL) better together with MapReduce, they created their own framework called SQL-MapReduce. This framework allows developers to write functions which are fully integrated in the database of the Aster Data system. These functions can then be invoked like regular SQL-functions.

Twister

Twister is a framework invented and managed by Indiana University. This particular framework extends the MapReduce-Framework with the idea of performing multiple iterations of a MapReduce-computation. The aim of this framework is to be able to perform better on computations which involve an iterative computation like PakeRank, Multidimensional Scaling or k-Means Clustering [15].

Interfaces

As a difference to other MapReduce-distributions an interface acts as a gateway to a MapReduce-framework. These distributions provide the user with a different set of tools in order to develop and test MapReduce-programs. The user does not have to deal with the full API and set of tools of Hadoop, for example. Instead a smaller interface with less complexity is provided for the user. Most of these interfaces serve only educational purposes. Two examples for such interfaces are the Lisp-implementation at UC Berkeley and WebMapReduce developed at St Olaf College.

2.7 US-Patent

In 2010 Google obtained a patent for MapReduce[10]. This has of course caused a discussion whether this patent is correct or not. After all, the question is whether something like MapReduce can be subject of a patent, since it is a combination of well-known techniques. This could be a reason why it was not easy for Google to obtain a patent for MapReduce, which was pointed out in a post[11] on the DBMS2 blog The author of the blog article noted the long time it took Google to obtain a patent, which could be an indication for a necessity for Google to make several adjustments on the original patent text.

Still, at the moment there is no indication that Google will enforce its patent. An indicator that this behaviour is not subject to change in the near future is the fact that Google supports study programs which include MapReduce using Hadoop[12]. An enforcement of the patent could

[10]http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=/netahtml/PTO/srchnum.htm&r=1&f=G&l=50&s1=7,650,331.PN.&OS=PN/7,650,331&RS=PN/7,650,331 last accessed 2013-10

[11]http://www.dbms2.com/2010/02/11/google-mapreduce-patent/ last accessed 2013-10

[12]http://googlepress.blogspot.de/2007/10/google-and-ibm-announce-university_08.html last accessed 2013-10

mean that Hadoop would be considered as a patent violation. This would probably also include that users of Hadoop and Hadoop-related products would have to pay fees for using them.

In summary, it is still not clear what the purpose of Google concerning the patent is.

CHAPTER 3

Learning Theories

Since this thesis is written for the study program “Didactic for Informatics” it should not lack a study of theoretical backgrounds in the field of learning theories. However, this chapter is not intended to give a full analysis of the presented theories, since this would exceed the scope of this work, namely analysing how to present MapReduce to students.

In this chapter two cognitive learning theories will be presented and analysed: constructivism and constructionism. The focus of this analysis is mainly laid on their applicability for using the ideas of the theories when teaching programming.

3.1 Constructivism

Constructivism is not only used to refer to a learning theory, but is also a familiar concept in art, mathematics and architecture, for example. This chapter will mainly focus on constructivism as a learning theory. Jean Piaget is probably the most influential and well-known constructivist, followed by other leading figures such as John Dewey and Lev Vygotsky and Ernst von Glasersfeld. In contrast to other - preceding - learning theories constructivism is based on the assumption that meaning is imposed on the world and not vice versa. This imposed meaning is constructed by using what Piaget calls “schemata” [40].

Schemata

One can see the schemata of a person can be seen as an “index file in which each index card represents a schema” [40]. In other words, these schemata represent the structures of how we think and perceive the world. Each single schema represents a single concept or a single category of objects for a person. In general, categorisation is the task of mapping several stimuli to a class of objects. Thus a schema is needed in order to be able to tell which objects should belong to a certain category.

As an example for categorisation we could think of the stimulus of a cow. When an adult sees a cow he or she will probably know that it is a cow. A child, on the other hand, might think that it is looking at a dog, because it has maybe never consciously seen a cow and therefore does not yet know what a cow is since. Thus, the child does not possess a schema for “cow”. However, it is more likely that it will soon have a schema for “dog”. As a result of this, the child will maybe think “animal, 4 legs, big, brow that should be a dog”. Due to this need to think what this object could be, the response time of the child could be higher.

It is important to note that a schema will not always remain constant over time, as it is subject to change whenever it has to be adapted in order to be consistent with the environment [40]. The processes involved in the change of schemata are called **assimilation**, **accommodation** and **equilibration**.

Assimilation is the process by which a person tries to map a stimulus to a known class of objects. This means that when presented with a stimulus a person tries to map it to a class. It is important to note that a person does not process a single stimulus at once, but an increasing number of stimuli [40].

However, these mappings can, of course, be incorrect. In Austria, for example, there is the phenomenon that children, when asked to draw a cow, colour it purple. This can be ascribed to a famous advertisement used by the chocolate-company Milka, in which the so-called “Milka-Cow”, a purple cow, appears. For children living in cities with no or little contact to farming or rural areas, this reappearing purple cow on TV might become their concept of how cows looks like. As a consequence of this, they might think that all cows are in fact purple. Thus, the Milka-Cow is “assimilated” to the category of real cows instead of the category of fictional objects used in advertisement (or thought of as a cow painted purple).

Whenever it is not possible to assimilate a stimulus into an existing schema, accommodation comes into play. This impossibility can stem from the fact that there is no schema present or that existing schemata are not “fitting” for the stimulus. To overcome this there are two possibilities: construct a new schema or alter an existing one. This process of construction or alternation is called accommodation.

Now that assimilation and accommodation have been described, it still remains to think about their interaction. If a person always accommodated and never assimilated, he or she would soon have many very small schemata with almost no generality. On the other hand, if the opposite happened, then a person would never accommodate. As a consequence, he or she would have a very big schema and no opportunity to detect differences between stimuli (i.e. perceive objects as different). From this it is possible to conclude that “no behaviour is all assimilation or all accommodation” [40]. Thus the challenge is to find some kind of balance between the two processes. This balance is what Piaget calls equilibrium [40]. If assimilation and accommodation are not in balance than there is a disequilibrium. The term equilibration denotes the process of bringing a disequilibrium to an equilibrium. The bare existence of a disequilibrium motivates equilibration, since every organism strives to go back to equilibrium [40]. This also highlights the fact that schemata are constantly undergoing changes. Moreover, it is important to note that there are no wrong schemata - only “better and better placements as intellectual development proceeds” [40].

From this section about schemata it is possible to deduct one possible origin of the name constructivism: all knowledge is constructed - using schemata - from the beginning of our childhood through adulthood until we eventually die.

Kinds of Knowledge

Piaget divided knowledge into three kinds: physical knowledge, logical-mathematical knowledge and social knowledge. [40] In order to construct new knowledge of any of those kinds, a learner is required to perform certain actions, but for different reasons. An action, which can be mental or physical, is necessary in order to gain new knowledge, since without active experience development cannot take place. [40] When teaching informatics we are mostly concerned with logical-mathematical knowledge. Here learners "invent" knowledge. This knowledge is not bound to objects but constructed, based on the experience gained from the actions on the object. "The objects serve merely as a medium for permitting the construction to occur" [40]. In contrast, physical knowledge is bound to objects. Here knowledge is constructed while manipulating the object. For example a child could learn about sand while pouring it from a sand box into a basket. The third kind of knowledge stands in contrast to the other two: Social knowledge is constructed based on action on or interaction with people rather than objects. Here knowledge stands for social or cultural rules and morals, for example.

The development of language made it possible to represent actions in our mind. Nevertheless, active actions are still necessary in order to master the task of acquiring new knowledge.

Concrete and Formal Operations

Piaget argued that the "intellectual development" of a child occurs in stages. It is important to note that these stages are not meant to be discrete levels so that a child is moving from one stage directly to the next like on stairs; instead the change of levels is rather fluid. The age groups ascribed to these levels are not fixed as well. Piaget identified four stages:

1. sensorimotor intelligence (0-2 years)
2. preoperational thought (2-7 years)
3. concrete operations (7-11 years)
4. formal operations (11-15 years) [40].

In the stage of concrete operations a child develops the ability to apply logic to problems. These logic operations enable the child to understand the basics about how objects are transformed into a new object and how operations can be reverted. Furthermore, children in this stage develop the ability to bring objects into an order and group them using similarities. If a child is still in a concrete operational stage, it can only deal with tangible and concrete problems

In contrast to the concrete operational stage a child in formal operations does no longer have the need for problems to be tangible and concrete. Formal operations are necessary for a child to be able to "deal with all classes of problems" [40]. Characteristic of formal operations are scientific reasoning and the building and testing of hypotheses. This also means that a child is now aware "that logically derived conclusions have a validity independent of factual truth" [40]. Both concrete and formal thought employ logical operations.

During the stage of formal operations a child develops the following new structures:

Hypothetical-Deductive Reasoning: Using hypothetical-deductive reasoning one can deduce the following: If $A < B$ and $B < C$ then $A < C$. On first sight, for an adult this does not seem to be a big achievement. A child in the concrete operational stage, however, lacks this ability to reason deductively about hypothetical assumptions. Only from the formal operation stage onwards, a child is able to deduce knowledge from a hypothesis and therefore deal with things that it does not know directly. At the same time, from this stage on it is also possible to infer conclusions about hypotheses which are believed to be not true.

Scientific-Inductive Reasoning: This kind of reasoning enables children to reason "from specific facts to general conclusions" [40]. Using this process of reasoning one can build generalisations and scientific laws. In [23] Inhelder and Piaget concluded that there is no difference in the capability to reason between scientists and children with formal operations. One big difference to the concrete operational stage is that children are now able to reason about more than a single variable. They can reason in a coordinated manner over a number of variables. Furthermore, formal reasoning allows to think about the effect of a single, all or a combination of a subset of variables. This is what Piaget refers to as combinatorial reasoning [40]. Scientific reasoning helps when a conclusion cannot be derived from observation only.

Reflective Abstraction: Reflective abstraction goes beyond possible observations and results of mental reorganisation. It is always present when logical-mathematical knowledge is constructed. Technically speaking, it is the major mechanism being part of these constructions. Furthermore, reflective abstraction enables the use of analogies. An analogy consists of objects which have certain relations. It is not possible to deduce all these relationships from experience, but the relationships become clear through reflection (reflective abstraction to be precise) [40].

Another important understanding children gain during this time is the understanding of proportions. This enables them to understand concepts such as probability. Probability in this case is the ability to understand change and proportion. It is also important to note that the concept of probability will not be constructed before reaching the stage of formal operations. A simple experiment can show if a child has developed an understanding of probability: "A set of 96 one-inch wooden blocks of four different color is placed on a table where they can be seen by the child. The distribution of the blocks by color is 36, 36, 20, and 4. The blocks are separated into groups by colours, then each group is halved. Half the blocks of each color (18, 18, 10, 2) are set to one side as a reference set. The remaining blocks, which the child must acknowledge as identical to the reference set, are placed in a bag or box and hidden from view" [40]. After the blocks have been mixed up, the child is told to pick two of these hidden blocks without looking into the bag. Then the child is asked to predict which colour the chosen blocks will have and explain the reason for the prediction. Before the child has developed formal operations the predictions will not be based on probability, but a random strategy or the favourite colour of the child.

Faces of Constructivism

As previously mentioned constructivism is a theory furthered by several people from different backgrounds. Thus, there are several types or faces of constructivism. Two of these faces are social and radical constructivism, which are presented in the section below.

Social Constructivism

One of the main forces behind the theory of social constructivism is the Russian psychologist Lev Vygotsky. The theory emphasizes the culture and context of a learner. “We "are" because of others” and “It takes a whole village to raise a child” are two good quotes to give a first insight into social constructivism [1]. Another emphasis of social constructivism is that the construction of knowledge is not a passive process shaped by external forces but a social process. Learning occurs when students engage in social activities [24]. Social constructivism is not so different from what Piaget describes as the construction of knowledge through relating to people and things. However, in his theory Vygotsky puts his emphasis on the acceleration and enhancement of the learning of children when adults with a greater expertise are present. Another aspect studied by him are cultural artefacts and how they are used in order to mediate the process of learning [1]. It is important to point out that for social constructivism the context in which learning occurs matters as much as all social contexts that are brought to the learning environment by the learner [24]. Vygotsky states that the cognitive development of a child is a process going from the outside to the inside or from socio-centric to egocentric. Any function appearing in the development will first appear on the social level before it appears on the individual level of the child [1].

Radical Constructivism

One of the main forces behind radical constructivism is Ernst von Glasersfeld. His theory relies on two basic principles:

1. Knowledge is not something received passively through our senses or by communicating with others, but built up in an active manner by a cognising subject.

2. Our cognition has the goal of organising our experienced world. This is done in an adaptive way, which means that there is no discovery of an objective reality [4].

The term radical constructivism comes from the fact that it is the most extreme face of constructivism. The basic foundation of the theory is “the concept that while a reality external to the individual may exist, the true nature of this reality is unknowable” [14]. Von Glasersfeld states in his paper that “[d]oubts concerning the correspondence between knowledge and reality arose the moment a thinking individual became aware of his own thinking” [39]. This is a contrast to social constructivism where knowledge is gained through interacting with others. Radical constructivists view other learners as “additional environmental entities” [14]. Any social interaction with other learners can lead the particular learner to rethinking of his/her own ideas, which makes social interaction useful. However, it is the rethinking which is considered to be responsible for constructing knowledge and not social interaction [14]. The responsibility for which knowledge is constructed is in the hands of the rethinking learner.

In radical constructivism it is assumed that teacher is concerned with the question whether the learners' understanding of a topic is coherent and valid with materials presented and interacted with in class. Furthermore, that the goal of teaching is "an individual, viable model of understanding, not the acquisition of a predefined set of supposed reality-based concepts" [14].

3.2 Constructionism

Constructionism is a theory for science teacher education [20]. A good way of explaining the idea of constructionism is to reformulate it in a different way, for example as "learning by making" [31]. This word play comes from the author/inventor of constructionism Seymour Papert [31], who was a student of Piaget for five years [37] and also collaborated with him later. Seymour Papert rethought education in our digital age on the basis of Piaget's findings. His research work is mainly focused on the use of digital media and computer-based technologies for educational purposes.

As stated before the theory of constructionism shares the constructivists' view of learning as building blocks. Papert expresses the connection of constructivism to his "own theory" in the following way:

> "Constructionism – the N word as opposed to the V word – shares constructivism's connotation of learning as "building knowledge structures" irrespective of the circumstances of the learning. It then adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity" [31]

.

In the 1980s Papert gave a speech for educators in Japan, relating constructionism to instructionism. He stated that the latter tries to achieve better education through better instructions [30]. The same can be applied to a scenario in which a computer is used for education: When the computer gives better instruction to the student, the educational output will be better. Constructionism, on the other hand, is more about learning, as Papert stated: "Well, teaching is important, but learning is much more important." [30]. Constructionism is about how people form their ideas and express them through different media. The particular context in which the individual minds work out these ideas matters as well [1]. In the same talk Papert also illustrated ideas of constructionism when it comes to the task of teaching children the concept of fractions. Usually when fractions are taught in school, the teacher uses pies in order to illustrate how the concept of fractions works. The "problem" with this approach is that this usage of pies is someone else's knowledge [30]. The pies might represent how fractions work for the teacher but not for all of the children. The child in the presented example learned fractions by studying shapes while working on a software project. She constructed her own meaning of fractions and how to use them. This was "the transition when fractions stop[ped] being teachers' knowledge and become her knowledge" [30]. The presented result is that even by doing very simple things, the learning ability of the children improved.

Artefacts

In the speech Papert gave to educators in Japan mentioned earlier, he included an important message about constructionism: “Giving children good things to do so that they can learn by doing much better than they could before” [30]. This means that teaching is not as important as learning. What matters is that the students actively construct their own knowledge. Therefore, in order to make “learning experiences personal” [20] children should construct their own public artefact as a result/representation of their study process. In that way the students have to engage in deep learning. They will have to research for themselves in order to be able to design and construct something which then becomes their “artefact or model as a representation of their knowledge” [20]. This style of learning will help children to be more aware of their own methods and styles for solving problems. In addition, this self-engaged studying of problems will lead to a deeper understanding of nuances of the problem [21]. Constructionism proclaims a shift from a perspective where a student is treated as a universal object to being taught towards an individual with own “favorite representations, artifacts, or objects-to-think with” [2]. The process of active learning is focused on the mental process occurring while the artefact is being constructed. This also means that the quality of the final artefact is not the main focus [3]. Allowing the children to focus more on components of their solution rather than the final artefact allows them to imagine more and different endpoints or “out of the box” thinking. All this means that the student will have an increased connection with his or her artefact [21].

Constructionism is well known as a learning theory for science and mathematics, but especially for teaching programming. This comes from the idea that it is well-suited and tailored for programming since most of the examples given by Papert involve some kind of programming [31]. A task sometimes difficult to solve is to provide students with the ability of creating interactive computer-based artefacts. This is due to the fact that not all students are computer or programming experts. However, there are several computer programs tailored for the needs of programming novices. One approach to enable students to construct meaningful programs is the paradigm of using task-tailored programming languages. One example for such a language is Logo, as invented by Papert.

One thing to point out is that an artefact does not always have to be a computer program [3]. A different approach for artefacts is that students use spreadsheets to create their artefacts. In [3] Beynon and Roe invent the term ’construal’ to describe a computer-based artefact constructed during active learning. Such a construal is usually much simpler than a computer program. This means that learning under constructionism is not limited to writing a computer program. A child could, for example, learn about the “electromagnetic phenomenon associated with a wire coil” by performing an experiment and having the result displayed on a computer screen or printed on paper [3]. One has to keep in mind that this result has to be meaningful for the child. The child should be able to have a primitive interaction with the result, in a sense that he or she can read the “direction and strength of the electric current, and the disposition and density of the lines of the magnetic field” produced by the wire coil [3].

Sharing of Artefacts

Another aspect of constructionism is that knowledge can be shared. This is meant in a sense that the constructed artefact can be shared with others more easily than knowledge that only exists in the head of the child. The girl from the example presented by Papert in [30], for example, is able to share her understanding of fractions with her classmates by demonstrating the program and explaining the ideas behind it.

Sharing is not only limited to classrooms or school buildings. As Holbert et al. point out in their paper [21] platforms like YouTube can also be used for sharing. They explain that players of a video game, for example, could record their solution to an in-game problem and upload it to YouTube. Other players should then be able to watch these solutions and also comment on it. Moreover, it is possible to construct a video game in a way that such solutions can be watched while playing the game. For computer programs a child could create a small video showing a program run and explaining the code while browsing through it. This is not a very difficult task since most of the modern laptops come with a microphone built in. Still, in most scenarios it would require installing additional software in order to create a screen cast of the program.

At Northwestern University, the origin of NetLogo, a doctoral student in learning sciences has recently developed a platform called “Modelling commons”[1] for sharing NetLogo models on the internet. This platform is now fully integrated into NetLogo. This means that any user can simply create an account for the platform, upload a model to the platform and download a model to the platform from within NetLogo.

3.3 Relating Constructivism and Constructionism

The theories of constructivism and constructionism share many similarities and some important differences. The aim of this section is to explore these similarities and differences by comparing the two theories. When describing constructivism the focus will be on Piaget’s theory.

Both Piaget and Papert share the idea that knowledge is constructed rather than transmitted. Both researchers view the world as something which is constructed by the child and of which there is no absolute truth. For them knowledge is also an incremental matter. Children constantly reconstruct their knowledge whenever it is necessary. Such a reconstruction is necessary, for example, in a situation where their view of the world is in conflict with something newly experienced. Both researchers investigate the situations that make learners change their cognitive structures. They want to find out under which conditions a learner is able to change his or her perception or view of the world and thus gain a deeper understanding about themselves or a certain topic.

However, Piaget and Papert’s approaches to get to such a situation are different [2]. One basic difference between their theories, for example, is the type of external aids given to students. While Piaget does not limit his aids to a particular kind, Papert focuses his study mainly on computer-based technologies. However, this fact should not be overestimated since Piaget passed away in 1980 long before computing became ubiquitous. Furthermore, Papert never limits constructionism to computers.

[1] `www.modelingcommons.org/`

Related to this difference is a second difference, namely in the treatment of "objects to think with" [2]. While constructionism relies on artefacts constructed by the learner, Piaget's theory is based on the assumption/fact that children will become progressively detached from concrete objects and thus gain the ability to mentally manipulate objects and test hypotheses. Papert studies how children construct and express their knowledge through different media. His idea is that being "in the situation" is most powerful for understanding [2]. Piaget's theory draws the attention to the things needed "to maintain the internal structure and organization of the cognitive system" [2]. Here, constructivism and constructionism seem to be a bit contradictory. When viewing the previously stated ideas as "goals for how teaching should be" then it is not possible to make children, on the one hand, create real objects and, on the other hand, think about objects as abstract concepts.

Papert's artefacts are also a possible difference to Piaget's theory. While Papert encourages the sharing of artefacts and postulates that this sharing helps learning, Piaget does not mention something like this explicitly in his theory. Nonetheless, one can argue that this sharing can be seen as a form of social interaction. This social interaction, however, is a part of constructivism. Altogether there are still some differences in the aspects of the theory since a physical artefact is easier to share with others.

Another difference between Papert and Piaget is the way how they think about children and the behaviour of their "ideal learning child". Both constructivism and constructionism allow children to construct their own idea of the world. Still they differ in their focus of the way to think about children and how to treat them. In constructivism all children are treated equally, while in constructionism there is a shift in the emphasis from universal to individuals with their own artefacts [2]. Papert sees a child as someone who wants to learn on its own, someone who wants to experience new things personally.

3.4 Conclusions for Creating a Course on MapReduce

One first conclusion that can be drawn from the statements above is that one should put an emphasis on artefacts rather than teaching processes. Throughout the course the creation of artefacts, i.e. programs, should be in the focus instead of the presentation of the topic. This means that rather than spending much time on preparing perfect presentation slides/material, time should be spent on inventing or defining tasks for students in which they can create meaningful artefacts. Moreover, when grading these artefacts it is important to keep a focus on the creation process of the artefact rather than the end result as handed in by the student. However, it is important to communicate this since this might be not a common grading process to some students. Another aspect that should be kept in mind is that there should be space for students to come up with their own projects (i.e. artefacts they create).

Another finding for designing a course with constructionistic ideas in mind is to include the sharing of artefacts, which can be done in different ways. A simple idea is to encourage teamwork. However, this is by no means a trivial task since in most scenarios teams in programming classes end up to be a "one is working, others are watching" scenario. Thus, it is important to monitor the constitution process of the teams and division of work within the teams. During the creation of the artefacts it might not be necessary that every student involved in the project writes code as long as he or she constructs an artefact which represents his or her understanding of the topic. Another way for sharing the artefacts is that all students are encouraged or required to present their solutions. This presentation of solutions could be done in various ways like a presentation in front of the whole class or a subgroup of the class. Additionally, the students could be asked to just create a small presentation of the artefact. This presentation should be written and intended to give an overview of the artefact so that an interested colleague can understand the ideas behind the artefact. Yet another way to share the artefact is to use online platforms like Modelling Commons.

CHAPTER 4

Requirements For A Framework For Use In Education

In this section some aspects for a possible MapReduce-framework are discussed. In particular, important factors for the success of a framework over others are investigated. Moreover, some existing frameworks are analysed for their usability when teaching a course for inexperienced programmers. The last part of this section describes all decisions that have been made during the development of the presented framework.

Throughout this section the term "productive environment" occurs a couple of times. It can be defined as a setting where multiple users can submit jobs to a jobtracker. In those scenarios clusters are not considered to consist of only a few nodes (e.g. less than ten nodes), but more than 100 nodes. This is despite the fact that the aim of this thesis is to work with small, i.e. non-productive, environments.

4.1 Considerations Before Implementing the Framework

Platform

A platform in this context is similar to a development environment; in fact, it could be described as an integrated development environment (IDE). This means that a platform in this context should contain everything that is needed in order to create, maintain, run and debug programs.

A framework for MapReduce should come with a platform or be integrated in an existing one. The reason for targeting this is the goal to keep the focus of the students on the content, i.e. on MapReduce and not the framework, so that the students can focus more on developing algorithms and then on how to set up the environment for the framework or the tools needed. This leads to the point that the framework should be easy to use in a sense that only common technologies should be employed. A benefit of this is that the extraneous cognitive load will be minimal; in other words, the technology does not distract the student from learning how to program with MapReduce. What is more, the development platform should not be tailored to a specific operating system (OS). A dependency on a platform could be a limiting factor for acceptance and popularity. Additionally, an OS-independent system is subjected to be developed by more developers. Nowadays, platform independence can be achieved easier by using platform independent programming languages like Java. A disadvantage of this is a possible performance drawback. On the other hand, the framework is not intended to compete against MapReduce-implementations tailored for productive use.

Another requirement that comes into mind is that one should be able to run and build MapReduce-programs without the need of invoking commands from a console/terminal. Although there are many books and tutorials about commandlines like bash, dos, PowerShell and Ash, for example, many users prefer a graphical user interface (GUI) over a commandline-script. An argument for having a GUI over a commandline interface is that the mental processes involved in writing and running a MapReduce-program should be kept at a minimum. The target group of the framework are programming novices and students with a low computer science background. Thus requiring knowledge about shell-scripting and commandlines would not be feasible.

In conclusion, the framework should keep the focus of the user on the programming aspect. One could argue that programming always involves some knowledge about computer basics like basic commandline operations (move, copy and compiling amongst others). Still, a shift away from this paradigm has taken place. Furthermore, the framework should not be restricted to "experienced" programmers who use commandline-based technologies.

File System

All productive MapReduce-implementations use a distributed file system (DFS) like GFS or HDFS. This is one key point of MapReduce since it removes the need of copying data to nodes before starting a computation. The downside of a distributed file system is its maintenance. Normally a server, or even more than one, is involved in managing meta-information about the files. So far, there seems to be no problem to achieve this behaviour in schools, since the file system server could run on the PC of the teacher. The problem is that in most cases this server needs a static IP-address. Such a static IP is not always guaranteed in schools. Additionally, for managing the required information for the distributed file system properly all the data nodes would be required to have a static IP. A third reason against a classic DFS is that it is designed for productive use. This means that it is intended for environments where the data nodes are not meant to be switched off on a regular basis. If a student, for example, switches off his or her PC because the lesson is over or the PC just needs to be rebooted, the server would issue several copy-tasks because some of the data is not replicated often enough. Another reason against implementing or including a DFS in the framework is that its setup usually involves changing firewall-settings. This is a possible security risk, especially when the targeted PCs belong to non-experts in internet security. A faulty firewall configuration would lead to a vulnerable PC, which is very dangerous in environments like schools where numerous PCs are connected over a network and probably operated by different users.

As a side note it should be mentioned that HDFS might still not be ready to use in such a scenario. During my experiments with Hadoop in the analysis phase I ran into trouble with HDFS quite often. The replication process frequently failed, leaving me with the only possibility of getting the system back to work by deleting all data as well as reformatting the namenode and all datanodes. This switching on and off of the virtual machines seems to be the reason why HDFS failed or had problems during my tests. Of course this resetting task could be done automatically by a small bash script in a normal environment where all IP-addresses are known, but when the data nodes to be formatted are not clear, this script would be more complex and prone to errors.

In conclusion, all these mentioned down-sides of a classic DFS should be avoided in the framework. This means that the framework cannot rely its performance on the existence of a DFS. Of course, a sginificant performance gap could be the result of this fact when benchmarking against MapReduce-implementations like Hadoop. To get a bonus from this loss the non-existent distribution of data should be used for a simpler code base of the framework. HDFS's rack awareness, for example, does not have to be implemented. Rack awareness means that the framework knows which node is in which rack of the cluster. This knowledge is used during the assignment of tasks. A good task assignment is used for minimising the network load during the computation.

One reason why a distributed file system might not be necessary is that in most of the application cases a distributed computation might not take place. Models will be developed and tested on a single machine by the students. In this scenario no DFS is needed. Of course when the model is tested and run on a cluster, a DFS would speed up the computation and thus be a great example to show students the acceleration of performance gain of MapReduce in comparison to other programming paradigms. However, the need of a DFS could be circumvented by just distributing the data to all involved PCs/nodes. In practise this could be done by just providing the data on a course website like moodle. A participating student would have to download the dataset to his or her computer. As a bonus these datasets could also include smaller datasets in order to be able to test the program locally. A critical reader might say that this data distribution would be a huge effort. Still it is necessary to keep in mind the effort it could take to maintain the DFS-clientsoftware on all student PCs when working in a school.

4.2 Analysis of Other Frameworks

In this section some of the existing MapReduce-implementations are analysed. Its aim is to give some insights into these implementations and point out some crucial parts of the techniques behind MapReduce. What is more, this chapter can be seen as a "state of the art" or "best case" analysis.

Aspects to Be Analysed

Any analysis should be carried out in a systematic way. In order to be able to do so some criteria for analysing existing MapReduce-implementations have to be identified and described. For a viewpoint of using a framework in an educational setting four categories of aspects can be identified:

Installation: *What is needed to install the framework?* This involves the whole process of installing, from obtaining the software to a running cluster. The differences between an installation for a cluster and an installation for a single node configuration should also be analysed. Moreover, the question should be answered whether a single node configuration can be connected to a cluster and whether a node of a cluster can be turned into a single node. Additionally, all dependencies on other software packages should be identified.

Writing programs: *How are programs written?* One important aspect is the documentation of the framework. A long and technical documentation for a framework could hinder potential users to start programming with it. What is more, the question is how the source code is compiled to an executable. Of particular interest are also the possibilities available for testing MapReduce-jobs.

Running programs: *What steps are needed to run a program?* This usually involves a communication of some kind with the cluster. The question that should be investigated is how this communication takes place. Moreover, it is interesting to know whether users need to communicate with the cluster directly or whether there is an interface provided for them. Another aspect is how the program is invoked: are there special parameters necessary to run a MapReduce-program?

Maintenance: *How to maintain a cluster?* Once a cluster is configured it usually needs some maintenance. Possible tasks include backups, for example. Another important task for maintaining a cluster is the replacement of single nodes. Moreover, the ability of adding new nodes to the cluster should be investigated.

Google's MapReduce

There is not much knowledge about Google's framework. Nevertheless, some known aspects are described in the section below.

Installation

Since the framework is closed source and binaries are not available, is impossible for private users to install this particular MapReduce on a private cluster. The paper [11] suggests that there are different implementations of MapReduce for different cluster configurations. Possible configurations are, for example, small shared-memory machines, large NUMA multi-processors or a collection of networked machines [11]. This could also mean that there are different installation procedures for each of the different configurations.

Writing Programs

For the framework presented in the original paper [11] MapReduce-programs are implemented using C++. The framework is provided as a library linked to the user programs. Mapper and reducer functions are implemented as classes. These classes are then presented to the framework using C++-macros. What is not mentioned in the paper is the task of testing programs. Due to the nature of the framework and Google's eagerness to keep the framework private, it can be assumed that programs can only be tested using a cluster. A behaviour like this would certainly slow down the development process of a program.

Running Programs

Looking at the example code provided it seems that programs are just executed on the cluster without any further configuration parameters. From what is known it can be assumed that users do not have direct access to the cluster but to a scheduling system. To this system a user submits a job, while the system allocates the nodes which will be used for the computation of the job [11].

Maintenance

Google owns several clusters, often only consisting of so-called commodity hardware. Commodity hardware is a category of "cheap hardware". This means that the computers are easy to obtain but are not high-end computers, which is a contrast to the usually high-end computers in data centres. Moreover, these clusters at Google usually consist of "hundreds or thousands of machines" [11]. This makes hardware failures within a cluster very likely. Unfortunately, no information about how failures are handled is available for the public.

Hadoop

The framework analysed in depth is Apache Hadoop version 1.0.2. The reasons for this decision include among others:

- The fact that it is open source.
- Hadoop is part of many other MapReduce-distributions. (Refer to 2.6 for some examples.)
- It is used by many major companies. (For a full list see `http://wiki.apache.org/hadoop/PoweredBy/`)

Installation

One thing about the installation of Hadoop that makes it not practicable for the use in education is the strong dependency on configuration files. In order to set a cluster up one has to edit a variety of configuration files. The official guide for a cluster setup includes about 100 variables which are subjected to be configured. Another aspect of the setup is that every node needs a password-less SSH access. Of course this is a potential security risk, especially when it comes to schools. What is also of interest is the fact that for Windows it is necessary to additionally install Cygwin[1]. It has to be noted that when installing Cygwin also SSH can be installed during the installation process. Hadoop is shipped as a compressed archive. The archive just needs to be unpacked into a directory. For a proper installation some environment variables have to be set. When the configuration is not altered, it will run in a non-distributed mode. As a result, an (experienced) programmer can develop and test applications without access to a cluster. It also has to be mentioned that Hadoop relies on static IP-addresses. All nodes are declared in a single file by their IP-address or hostname.

[1]A tool collection to provide Windows with tools available for Unix

In summary, Hadoop is straightforward to install for experienced (Linux-)users. However, the requirement of altering environment variables could trouble novices. It might be useful to note that the whole process of installing is outlined in detail on the official homepage.

Writing Programs

Hadoop is based on Java and provides a Java-API. Nevertheless, programs can be written in any language when the Hadoop Streaming utility is used. In this streaming utility a mapper or reducer is a single executable script which is then executed by the Hadoop framework. Another possibility is writing MapReduce-programs in C++ using Hadoop Pipes. Both mentioned approaches (streaming and pipes) are not desirable for educational programs. One reason for this is that they often lack a good program structure, especially in the case of streaming. In the case of pipes the compilation is much more complicated than for usual programs.

Mapper and reducer functions are written as classes which need to implement the Mapper-interface of the API. The basic schema is outlined below:

```
public static class Map extends MapReduceBase implements Mapper
    <LongWritable, Text, Text, IntWritable>
```

As can be seen in the latter part of the definition, the API uses genericity to be able to deal with different data types for keys and values. Of course this means that a programmer also needs to have good knowledge of this mechanism in order to be able to write a MapReduce-program.

For compiling the source code only the Hadoop library is necessary. Due to the fact that a single node installation of Hadoop can be obtained with just a few steps, it is also easy to test MapReduce-programs locally.

Running Programs

Programs can only be executed on the cluster when they are transferred to the cluster. This means that a program needs to be copied to the cluster, where it can then be executed by the user. A program is usually provided as a jar-archive. This archive is then passed to Hadoop along with a parameter specifying the name of the main-class to run. Additional parameters can be passed alongside to specify input and output location, for example. Of importance is that both input and output need to be on the distributed file system. As an additional feature Hadoop provides a web-interface where running jobs can be monitored.

Maintenance

Since nodes are specified only to the master, an individual node can be replaced by guaranteeing that the new node will have the same IP-address as the old one. Additionally, if all worker-nodes share the same configuration, the configuration files can be copied from any other worker-node. Likewise, a new node can be added to the cluster by the declaration of its IP-address on the master.

Lisp-MapReduce at Berkeley

For their CS61A “Structure and Interpretation of Computer Programs” course[2] the team at UC Berkeley has developed a MapReduce-framework which uses Scheme, a Lisp dialect, as programming language. The framework in this case merely serves as an interface to a Hadoop installation. The Hadoop installation has some local modifications featuring a Scheme interpreter.

What makes this framework different from others is its functional-style MapReduce design. While the ‘classic’ MapReduce uses some kind of iterator for values passed to the reducer, the Lisp-framework uses a different approach: A true functional MapReduce will not pass all values to a reducer at once; instead the reducer takes two values: an accumulator-value and a value. The result of one application of this reducer-function is the next accumulator-value. This also means that all reducer functions have to be associative which also means that the computation of the reduce-phase can be parallelised more naturally than for traditional MapReduce frameworks where the reducer function does not need to be associative.

Moreover, the framework keeps its functionality to a minimum. Programmers cannot access any of the configuration parameters offered by Hadoop. However, as a result of this the programmer does not have to provide a job configuration as normally required from Hadoop. Another difference is the restriction for possible inputs for the mappers. The Berkeley-framework will only accept the name of a directory on the distributed file system or the result of a previous MapReduce-computation.

Installation

Installing the framework requires of course setting up a Hadoop cluster. Additionally, the Lisp-Hadoop interface needs to be installed and configured on the cluster. Another task during the setup is that all students need an account in order to access the server. All students are provided with login credentials to the cluster using ssh.

Writing Programs

The framework is reduced to a basic functionality. Mapper and reducers are simple functions, taking one (mapper) or two parameters (reducer). Mappers are limited to take their input from a predefined dataset or from the output of another MapReduce-computation. It is also important to note that the framework requires clean input-data in order to function properly. This can be seen as probably the only big disadvantage of the framework. One can imagine the amount of manual work necessary for providing students with clean data when data-sets are usually the size of a gigabyte or even bigger. Moreover, it needs to be pointed out that since the framework needs access to Hadoop, students cannot test their programs at home.

Running Programs

Programs are executed like normal scheme programs. However, in order to run a program it needs to be copied to the cluster where the framework is installed. Running programs can be

[2] `https://inst.eecs.berkeley.edu/~cs61a/sp11/` last accessed 2013-10

tracked using the tools provided by Hadoop, since the framework integrates into the infrastructure of Hadoop.

Maintenance

There are not many resources available about the details of the framework, thus not much can be said about its maintenance. Since the framework runs on an on-site cluster of the university, students are not involved in maintaining the cluster. However, it can be assumed that the maintenance is similar to Hadoop-clusters. Another aspect about maintaining the cluster is the need of maintaining the user access. This means that there must exist an email-hotline or an office for students to request password-resets.

WebMapReduce

WebMapReduce is a frontend for Hadoop developed at St Olaf College. The framework is a web application with a web frontend for the user to submit jobs and a backend that serves as gateway to the Hadoop-cluster. Every user that can access the web interface can also submit jobs to the cluster. This means of course that no accounts on the cluster need to be created for students. However, every user needs an account for the web interface. If the accounts for users are not required to follow a naming convention, it is also possible to allow self-registration for potential users. Naming conventions come handy when assessments should be graded automatically or when it is required to know the real identity of the job owner.

Installation

Besides the installation of Hadoop the web interface needs to be installed. This is done by installing a webserver like Apache and a series of other tools. The whole task of setting up WebMapReduce is outlined on the official homepage in a detailed manner. However, like for Hadoop it is also necessary for this framework to modify a series of configuration files.

Writing Programs

Programs for the framework can be written in Python, C++, Lisp (Scheme) and C#. As a big difference to the other frameworks described a program is submitted directly to the web-interface by either copy-and-pasting or uploading the source code as a file. Moreover, a MapReduce-program is restricted to just a mapper and a reducer. This means that no additional computation can be done. As a result of this a program cannot be tested locally without access to the cluster.

Running Programs

A program can be run by just uploading the source code to the web interface and starting the job. Every job is configured using this interface. As input the user can either input data by hand, use files stored in the distributed file system of the cluster or use a pre-defined data-set. Once configured a job can be stored to be run again with the same settings.

Maintenance

One possible problem when maintaining WebMapReduce is the (not yet) very elaborated administration console. For example, a user cannot reset his or her password without assistance. An administrator needs to look up the user in the database and set a new password. Another difficult maintenance task involves the security of the cluster. Jobs are submitted to the web interface which submits it using the backend to the Hadoop cluster. This means that no matter which user is logged in on the web interface the same user will be used for submitting the job to the Hadoop-framework. Due to this fact it is not possible to introduce security concepts featuring different levels of user rights. As a result of this it is necessary to secure the web interface from abuse. Of course the maintenance for the underlying Hadoop framework is not different.

Summary & Conclusion

When looking at the implications made by all of the presented frameworks it is striking that all frameworks assume that a cluster is at hand. This means that in order to use the framework with all its possibilities, an access to a cluster is needed. Most schools will not be able to fulfil this condition due to financial conditions and sometimes also space constraints. As a result of this none of the presented frameworks can be used in schools. However, one could of course argue that existing computing infrastructure could be used to build up a cluster. The computers in a school's computer-room, for example, could be used for installing a cluster. A cluster like this would connect the PCs in the room to build a cluster. There are various problems for bringing this idea to practice:

- Most schools - at least in Austria - use hardware for their computer-rooms that is behind current standards by a few years. This is most probably due to two reasons: Firstly, computer education is not seen as a primary goal by most schools. Secondly, today almost all families own a PC or even every child possesses a laptop. As a result of this less money is spent on PC infrastructure in schools, even in those which have a technical/computer background. All this results in problems when trying to use existing infrastructure for setting up a cluster.

- Another problem for a school to run a cluster using the normal PCs of a computer-room is the power bill. Normal PCs are not meant to serve as nodes of a cluster and thus the power consumption of such a cluster would be much higher than that of a "real" cluster.

- Additionally, there is the problem that most software-packages used to run on/build a cluster are not able to handle the possible shutdown of a PC by a student. Such shutdowns cannot be avoided when students are working on the PCs.

Moving the mentioned problem about the cluster aside, the frameworks are still different in their suitability for the use in education. Both Hadoop's and Google's framework are less suitable to be used as frameworks for a course with educational purposes. Their need of writing code in order to configure the framework and thus the requirement to understand the API and working principle of the framework is not a desirable behaviour for a framework in education.

The two frameworks that are both positioned in an educational field are the Berkeley-framework and WebMapReduce. Both of them require no or almost no configuration for a job. This allows the students to focus on the programming part of MapReduce. Thus, assuming a course has access to a cluster and uses Scheme as programming language, then the Berkeley-framework is a good choice. The limitation of WebMapReduce to only single jobs makes it a problematic choice when a bigger course on MapReduce with more advanced assignments should be taught.

In summary, of all the arguments in favour of or against the individual framework WebMapReduce sticks out of the crowd. By using it, it is very easy to explore MapReduce. However, the problem remains that it is not possible to create more complex algorithms using MapReduce in an iterative manner.

4.3 Goals for a Framework

The result of the discussion of frameworks is that a new framework has to be developed in order to meet the requirements of education. The framework to be created should achieve the following goals:

Simple The problem with existing frameworks is that most of them overwhelm the novice programmer with a lot of possibilities and parameters. Most of the offered functions are not strictly necessary for an introductory course. Thus a framework designed explicitly for educational purposes should basically offer only a single command: MapReduce and the parameters “mapper”, “reducer” and “input”.

True functional design The original MapReduce rests upon an iterator-based design where the reducer iterates over the values for a key and outputs the result as a key-value pair. However, a true functional design would rather suggest that a reducer takes an accumulator and the next value. A design like this is more intuitive for the concept of functions applied to data.

Fault tolerance The problem with, for example, the Berkeley-MapReduce-framework is that it is prone to errors in the input data. Because of this circumstance all input data needs to be cleaned before it can be handed to the students. Thus, it is not suitable for courses featuring real life cases or long-running courses where new problems should be issued every year.

Single node The framework should offer the ability to run MapReduce-computation on a single computer. Additionally, it should make no difference whether the program is executed locally or in a distributed matter.

Multiple jobs Unlike WebMapReduce it should be easy to use the result of a MapReduce-job as an input for the next MapReduce-job.

Job-Progress For many years users have been accustomed to see a progress bar displaying when they start a task like copying several files or run a long running task in a program.

It would be an advantage to add this behaviour to every program. Thus the framework should offer the functionality to check the progress of the current job. Most frameworks offer the possibility to check the status on a webpage. However, referring the user of a program to a webpage where the status of the program can be monitored is not practical, since it would require the user to switch between web browser and the actual program.

Documentation The framework has to be well documented. The documentation should not only include a description of all provided functions (i.e. an API documentation), but also a description of how the implementation works. Moreover, an in-depth description of bottlenecks and possible enhancements could be beneficial for more advanced students. They could think of ways to extend the framework and thus create a deeper understanding of MapReduce.

4.4 Decisions Made and Their Justifications

Logo

In order to develop a framework for MapReduce a programming-language has to be picked first. A simple framework for programming novices cannot be developed for multiple programming languages, although there are examples that seem to contradict this statement: the Hadoop Streaming API, for example. In this framework a MapReduce-program outputs key-value pairs that are submitted to a running Hadoop-Instance. The output created by mapper and reducer running on the Hadoop-Instance is fed back to the program where it can be processed further. Thus it is not fixed to any specific programming language. However, such programs often lack a good structure and are thus no desirable examples for teaching. Therefore, a programming language for which the framework should be designed must be chosen for the target group of the framework. The decision about the language was made in favour of Logo. It is based on various arguments, for example:

Simple Logo is a very simple and basic language. This is mostly due to the fact that many aspects of it are very intuitive which also means that writing programs can be done intuitively. The command and control structures, for example, are closely related to spoken instructions. Taking a language which is easy to understand should prevent potential problems when teaching MapReduce to programming novices, who might have difficulties understanding a high-level programming language. The language as such is very simple as it is built on few (reserved) keywords. Those keywords sometimes also have abbreviations in order to decrease writing and reading time of programs. One example for this is **forward** x which can also be written as **fd** x. What is more, Logo as a language does not differ a lot from spoken language while still maintaining a compact notation. All these conditions make Logo easy to learn. Students not familiar with Logo can gain knowledge about how to program with it very fast.

Widespread Logo is widespread in computer science education because of its simple terms. This means that many students and kids know this language from other courses or modules. Due to this fact it can be assumed that the students already have some knowledge about Logo or can acquire it within a short time. Another benefit from being widely spread is that the chance that some Logo-software is already installed on the PCs in a school or on the student PCs is higher than the chance that, for example, a Java or C++ compiler is installed. Still, a drawback of this also has to be pointed out: It could happen that an old version of the required software is installed and thus requires changes to the setup of a PC, which are in some cases not easy to accomplish. Removing old software and installing new versions can sometimes cause dependency problems. Especially for programming languages this can sometimes mean that old source code needs to be rewritten. However, in the case of Logo there are rarely changes to the language. Furthermore, it is usually not difficult to keep two different versions of a Logo dialect on the same PC.

(Almost) no limitations Logo is not tailored to a specific domain of problems or topics. Some programming languages were created to tackle a certain domain of problems. A tailored language is good when only this certain domain of problems is considered. When such a language is then used for different problems, it will lose some of its attractiveness in most cases. This tailoring can also result in a language containing a lot of features which are not trivial to understand. This means that the language could distract the students with its features from analysing problems and writing code. As said previously Logo does not have much tailoring towards a specific domain. Still, the language itself does not lack features required for writing programs for any kind of problems, although one might miss a certain feature like writing a Graphical User Interface (GUI), for example. However, most missing features can be implemented by a third party library or are already implemented and distributed as a library or extension.

Constructionistic origins As mentioned earlier, Logo was invented by Seymour Papert. Thus, when also using constructionistic learning theories, choosing Logo is obviously a conclusive if not required decision. What is more, Logo is considered to appeal to children due to its turtle graphic [28].

Another noteworthy aspect of Logo is its dialects. There are several dialects for Logo, each with its own environment for programming. As of June 2013 there are 267 known dialects of Logo. However, most of them are no longer active or unused [5]. This can of course produce a problem: A program written for a specific dialect cannot be used in a programming environment for a different dialect. Instead, is has to be rewritten in most of the cases when a different environment (i.e. dialect) is used. This means that the dialect(s) for which the framework is developed has to be chosen carefully, especially when the framework is not written in pure Logo.

NetLogo

As stated previously, Logo is not a usual programming language where it does not matter which environment (i.e. dialect) is used. This means that it is important to fix the dialect for which the MapReduce-framework should be implemented. With this choice of dialect the programming environment for the students is also chosen or at least limited to a few options. In Austria there are three commonly used Logo-versions: MSWLogo, FMSLogo and NetLogo [41]. The first two options are closely related since FMSLogo is a fork of the former. While the development of MSWLogo seems to be on a halt since there have been no new releases since 2002, FMSLogo is under active development.

First I considered implementing the framework for FMSLogo. A reason for this was that the package is easy to use and has some networking functionality implemented. Thus it could have been possible to implement the framework using only native Logo-Code. However, after some primary research I decided to implement it for NetLogo. This decision was mostly based on five reasons:

1. The source code of FMSLogo is based on a Borland-C++ dialect. Thus, implementing any features would require either porting the code to a different dialect or trying to acquire a license, which could, according to the FMSLogo webpage sometimes prove to be difficult. Another reason against FMSLogo was that there are only builds available for Windows, while NetLogo is available for Windows, Linux and Mac OS.

2. NetLogo is still under active development, while this could not be positively said about FMSLogo in early stages of searching the platforms for Logo for this thesis. Since the development of FMSLogo seemed to be on hold. There have been no changes to the code available on `sourceforge.net` for about six years. Furthermore, it seems that there is only one developer making significant contributions to the codebase. On the other hand, there are a few programmers working actively on NetLogo at the moment. As a result of this a new major release (5.0) of NetLogo was introduced in 2012 and minor releases are shipped about every six months at the moment (2013). Another point which should keep the NetLogo-project active over the next years is its close affiliation with the Northwestern University, Illinois.

3. The third reason for choosing NetLogo as platform is its extensible structure. One can easily write extensions to add features to NetLogo. What is more, installing these extensions is very easy. Extensions can either be placed in the installation directory or in a directory next to the model which is using the extension[3]. Since it should be easy for the user to experiment the extension, this was one major reason for choosing NetLogo over FMSLogo.

4. NetLogo comes with HubNet (see [38] for more details), a feature for connecting multiple NetLogo instances together. This seemed to be a very useful feature for implementing MapReduce using NetLogo, since the protocol stack could be built on top of HubNet and thus remove complexity from the implementation. More on the issue of using HubNet is summarised in sections 5.2, 4.4 and 5.1.

5. The last strong argument for choosing NetLogo over FMSLogo was NetLogo's user community, acceptance by users and spread (especially in the scientific community). As illustrated in figure 4.4, according to Google the interest-rate for FMSLogo is very low, while the search volume for NetLogo is significantly higher. However, the total search amount could not be established and is thus not revealed in the figure. Keeping the aim

[3]See 5.3 for more details.

in mind that the MapReduce-framework should be useful, choosing NetLogo seems to be reasonable. Useful in this context means that the students can use their experience gained from learning MapReduce directly for other purposes, such as additional classes using NetLogo.

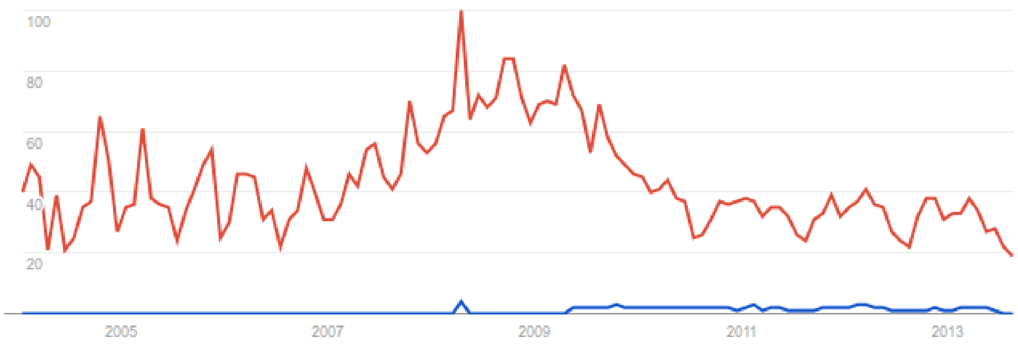

Figure 4.1: Google trends for (red) "NetLogo", (blue) "fmslogo". Graphic generated with Google Trends `www.google.com/trends/`

Lastly it should be pointed out that in case a transition from FMSLogo to NetLogo as programming environment is needed for a student, this transition is not a very difficult one. The probably biggest difference between the two programming styles are variables. Whenever the value of a variable should be accessed in FMSLogo, one has to precede the variable name with a colon (: variable), while for NetLogo there is no such requirement. Moreover, in FMSLogo variables have to be declared as local variables explicitly using "**local** " variable " and "**make** "variable value" for global variables. NetLogo, on the other hand, requires a statement like " let variable value" to define a variable implicitly. Additionally, in NetLogo every variable occurring within the context of a function (reporter or procedure) is a local variable by default. Global variables in NetLogo have to be declared explicitly using a "global [variable]" directive.

HubNet

A next decision was to use HubNet as protocol stack for all communication related code. This means that the protocol for every communication between master and nodes, and possible node-node communication will rely on HubNet. The advantage of this is that the code of the framework does not have to handle certain parts of any protocol at a low level. Parts of protocol involve issues like setting up the connection between master and nodes, failure of nodes, or any message that needs to be sent from the master to the nodes. If, for example, a node fails, i.e. the connection to the node breaks down, the master-node will receive a HubNet-message informing it about the closed connection. Another benefit of choosing HubNet is that it is fully integrated into NetLogo. As a result of this students which already have a solid experience basis with NetLogo will be familiar with HubNet. Thus, they will not be confused about how to set up a connection using HubNet between two or more computers. However, it has to be pointed out that using HubNet for a MapReduce-framework means that central concepts of HubNet are broken. See sections 5.2 and 5.1 for more details.

DFS

As explained previously, a distributed file system (DFS) can be particularly useful for MapReduce. One could even argue that it is a required feature when a MapReduce-implementation should be (performance-wise) competitive to others. However, for the framework in this thesis a DFS is not implemented or used. The main reason for this is the complexity of such an implementation. While frameworks featuring a DFS are developed by a team of programmers, this thesis is written by just one person. Another point against adding a DFS to the framework is its maintenance, as described previously. Nevertheless, a DFS can be used in a way that the files are stored in such a file system. This means that the advantages of a DFS are used on the file system side, but the MapReduce-framework will not make any scheduling decisions based on attributes (for example distribution of file chunks) of the DFS.

Java

Once the language for which the framework should be implemented and the Logo dialect had been chosen, the language for implementing the framework had to be selected. In this case Java was chosen as language for implementing the framework as an extension of NetLogo.

NetLogo as a programming environment is written in Java and Scala. Although currently most of the Java-Code is ported to Scala or subject to be rewritten, extensions can be written in both Java and Scala. While most of the code samples and manuals are still written for Java, programmers are encouraged to write new extensions using Scala as programming language. However, the decision was made to stick to Java. This decision is based on two reasons: First I am more experienced with Java than with Scala, thus implementing it in Java will take less time. The second reason is that at the Vienna University of Technology and University of Vienna most computer science students study only Java in computer science courses. Thus in order to create the possibility of a follow-up student extending my work Java had to be selected. Moreover, Scala is a relatively new language, thus there are not as many programmers familiar with it as with Java. As a result of this it can be expected that it is more likely that other programmers will decide to extend the framework once this thesis is finished when the code of the framework is written in Java.

Summary of Decisions

Logo was chosen as programming language for which the framework will be implemented.

NetLogo as Logo-dialect and as development environment for the students who will use the framework.

Java as programming language for the implementation of the framework.

HubNet as the protocol stack for the communication protocol between master and nodes.

no DFS will be provided/implemented for the framework.

CHAPTER 5

A Framework For Use In Education

This chapter describes the implementation process of a MapReduce-framework which is designed to be used for educational purposes. Moreover, the chapter gives insights into the presented framework. However, this chapter does not serve as a technical documentation. The purpose is to present facts about used technologies and reasons for their selection. Additionally, it lists some problems or difficulties encountered during the implementation.

The main goal of the framework is to serve the needs of teaching the concepts of MapReduce rather than to serve as a framework for productive use. This means that a single "user" serves as the dedicated master which will schedule a particular job (i.e. the current NetLogo-model) across a virtual cluster of NetLogo-instances which all have opened the same model as the master.

5.1 Used Technologies

NetLogo

NetLogo, which was developed in 1999 by Uri Wilensky, is "a multi-agent programmable modeling environment" [41]. Moreover, it is a special dialect of the Logo programming language. Instead of a single turtle the programmer can control various turtles modelled as individual agents, hence the multi-agent attribution. Another difference to "classic" Logo is that instead of creating a program the user creates a *model*. In contrast to other Logo distributions the NetLogo user interface consists of three tabs: *Interface*, *Info* and *Code*. On the interface-tab the user interface for the model can be created. It has to be pointed out that there are no differences between the user interface of the developer and that of a potential user. An example of the user interface can be seen in figure 5.1. On this interface tab every model can be equipped with interactive elements such as sliders, toggle switches, buttons and likewise. These GUI-items can be utilised by placing them on the interface and configuring them it to display the desired data or run the desired command. The intention is that by adding the possibility of controlling the model in an interactive way different parametrisation of a program can be explored in an intuitive way.

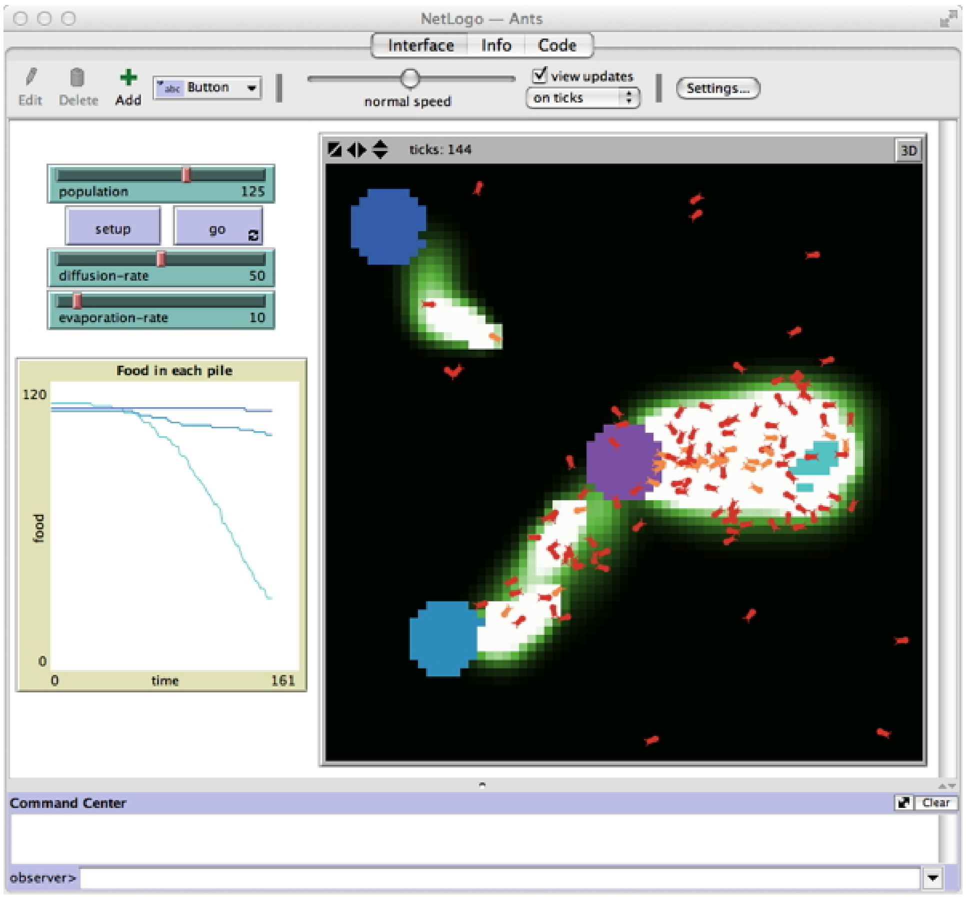

Figure 5.1: NetLogo User Interface on Mac Os

NetLogo aims to be a language both for children and novice programmers and advanced programmers. While the former most probably just want to learn how to program and explore the programming concept, the latter will aim to create models that can be used for research. As a result of this certain tradeoffs are made. For example in favour of an easy program creation the source code of a model is not distributed as a binary. This, however, reduces the execution speed of a model by at least a constant factor.

There are three types of agents: turtles, breeds and links. All of these types can own a set of variables. A link is a connection between two turtles or two breeds. A connection can be one- or bidirectional. This idea makes NetLogo powerful for modelling real-life aspects. Due to the simplicity it is even possible for people with no computer science background to create sophisticated models.

A key benefit of using NetLogo is that it runs on the Java Virtual Machine (JVM). This means that NetLogo can be used on Windows, Mac OSX and Linux. Additionally, the differences in "look and feel" between the different operating systems are marginal. Thus problems when students are asked to use NetLogo on their private computers are reduced.

Another benefit of using NetLogo is the tight link to constructionism. For example, models can be shared with others using the Modelling Commons platform. The info tab can be used in order to describe the model. Thus, a student can share his or her ideas on the model in written form.

HubNet

HubNet enhances NetLogo with a technology to "run participatory simulations in the classroom" [38]. This means that instead of a classic scenario where a teacher shows a model to students who are required to watch, students can take part in the computation. Every student controls a part of the simulation such as an agent, for example.

HubNet works using a client/server architecture. A model can be started as a HubNet-activity and thus serves as the server. Other users can then connect to this activity serving as clients. Currently there are only two clients available: a Java-Client for computers (either as standalone or as applet) and a client for Texas Instrument graphing calculators. However, there is no restriction on which type of clients can connect to the HubNet-activity.

Extending NetLogo

One particular useful aspect about using NetLogo as a framework is its design with regard to extensibility. The program offers two APIs in order to extend the way it can be used.

Extension-API This API offers the ability to extend the NetLogo-language by new commands. Thus, required features for a model or a series of models can be added to the NetLogo-programming language. Doing so increases the readability of programs and decreases the programming effort for modellers.

Controlling-API Additionally, a Controlling-API offering control over NetLogo is provided. Using this API NetLogo-instances (or workspaces to be precise) and commands executed in them can be created. Thus, a model can be loaded and an experiment performed without a required interaction by the user.

Moreover, it is possible to use the two APIs together. An extension can interface the controlling API and using the controlling API commands, using the extension API, can be invoked.

NetLogo is using a concept of workspaces. Every model runs in its own workspace. Additionally, a workspace can be run in a "headless-mode", which means that the GUI is not shown. Using this headless-mode models can be tested automatically without the need of interaction by the user. The automated testing can be done using the controlling API, for example.

5.2 About Developing the Framework

The framework was written entirely in Java. Additionally, a small library written in Scala was used to interface HubNet.

Problems with NetLogo

NetLogo is described as “a multi-agent programmable modeling environment” [41]. However, this statement is a bit misleading since the code of NetLogo-models is not suited for parallel execution. NetLogo works in two stages:manner First the code is compiled and then it is executed by the interpreter. Both compiler and interpreter are parts of the software developed by the NetLogo development team. The Logo dialect as such has no limitation to a single threaded execution mode. However, neither the compiler nor the internal language interpreter yet support a parallel execution of commands. Furthermore, any attempt of an extension to execute NetLogo-code in parallel will result in a deadlock.

As a result of this an attempt to model MapReduce requires a workaround: using extension and controlling API in parallel. This means that from within the extension code the controlling API is interfaced. To execute tasks in parallel each task will run in its own workspace. Using the controlling API workspaces are created before the mapping or reducing process is started. These workspaces are then used to run a map- or reduce-task.

Using workspaces leads to the next problem: inserting data into a workspace. Every task needs a set of parameters like the input file and start and end position in this file, for example. Using the parameters for a task the concrete values for a mapper or reducer can be obtained. The problem behind the parametrisation of tasks is that the NetLogo-design currently only offers the possibility to execute pre-compiled commands without parameters in a workspace or execute a command with parameters on the fly. This on the fly execution, however, requires of course the compilation of the command. Still, it would be possible to execute each task in an on the fly manner. However, this would result in a tremendous performance loss. So far there is no optimal solution for this problem. However, it can be solved by the following trick: using a factory-class all workspaces have access to a central manager. This manager stores information about the workspaces used for executing tasks. The information contains the parameters for the task subject to run in the workspace. Thus, a workspace can use itself to query for the task parameters. Once the parameters are obtained the key-value data of the mapper or reducer can be read.

From a performance perspective running a mapper or reducer with the key-value parameters is not trivial. This is also due to the fact that there is no direct access from the controlling API to variables defined in a workspace. The workaround here is simpler than the previous workaround. Instead of invoking the mapper or reducer directly with all parameters a wrapper command without parameters is created. This command just consists of a function-invocation, namely the mapper or reducer. Instead of values for the key-value data, custom reporter-commands are used as parameters. The wrapper can then be compiled and thus a lot of computation time is saved. The mentioned custom reporter-commands just report the data loaded during the task-setup as described in the previous paragraph.

One problem not yet solved fully satisfactory is loading the framework. The source of the problem is that it is vital for the framework to enable communication between workspaces using the previously mentioned factory-class. This is of course possible when all workspaces can share a communication interface. However, to make this sharing between workspaces possible, the framework needs to be loaded by NetLogo instead of the extension manager. An extension is loaded by the extension manager when the “extensions [extension]”-directive is used in the source code. If a workspace loads a model featuring an extension the extension manager will thus load it using an own *classloader*. This behaviour causes the problem since the framework is then loaded multiple times and no communication between the workspaces is possible. The workaround to this is that the framework needs to be on the *classpath* before NetLogo is started. Thus, the framework is loaded only once and the inter-workspace communication is possible.

Problems with HubNet

HubNet as a networking protocol is not well-suited for implementing MapReduce. Looking at the way HubNet works it could be assumed that every instance in the HubNet-network has the full computing power of NetLogo. However, this is not the case, since the HubNet-client does not include the interpreter for the NetLogo-language. As a result of this the HubNet-client cannot be used for serving as a node in a MapReduce-computation. The official HubNet-client is only capable of displaying data from the "server" and sending commands to it.

However, HubNet is used for the framework for the following reasons:

Client-library Due to the fact that the current capabilities of the HubNet-client are considered to be a limitation, the NetLogo-team created libraries to interface HubNet. Using these libraries it is possible to build a different client which seamlessly integrates into the HubNet structure. An example for such a client is the TI-Navigator$^{\text{TM}}$-client[1].

Network-stack HubNet offers extension programmers the ability to build an extension on top of HubNet. Thus, the framework would not be required to implement an own network-stack. Using the existing network-stack would prevent the framework from the need of dealing with tasks like keeping track of new connections and sudden disconnections, for example. Thus, the network-stack for the framework can be separated into a network level handled by HubNet and an application level handled only by the framework. The application level handling involves agendas like scheduling decisions, for example.

Existing interfaces NetLogo respective HubNet provides user-interfaces for connecting computers. These interfaces can be used by the extension to obtain data like the IP-address of the master-node. Using these common user-interfaces is a good software engineering practice. Moreover, it is easier for users to use the application when well-known concepts are reused.

Additionally, it has to be pointed out that HubNet like NetLogo is not suitable for multi-threading. As a result of this a managing interface to HubNet had to be created. The entire communication with other nodes is done using an additional workspace. In this workspace the communication can be handled in an infinite loop. All occurring HubNet-events are passed to the corresponding infrastructure of the framework. Commands and messages from the framework can be passed using this interface via HubNet to the nodes.

[1] `http://www.inquirelearning.com/calc-hubnet.html`

5.3 Description of the Framework

This section briefly describes how to use the implemented framework and its working principle. Moreover, it summarises previously stated aspects about the implementation and puts them in context. A full documentation can be found at `https://github.com/mado89/netlogo-mapreduce/wiki`.

Installing

Extensions for NetLogo can be installed by just placing them either in the directory of NetLogo or in the directory where the model using it is located. All files provided with the extension are placed in a folder having the same name as the extension used in the model. More information on this topic can be retrieved from the official NetLogo-site: `https://github.com/NetLogo/NetLogo/wiki/Extensions`.

As mentioned in "Problems with NetLogo" the framework needs to be loaded prior to loading a model. In order to load the framework it has to be added to the classpath for NetLogo. The classpath contains all locations Java will be using for searching for objects used in programs. Thus, having the framework on the classpath is a necessary requirement. Therefore, the start-up of NetLogo needs to be modified slightly. For Unix-Operating systems this can be done by creating a modified version of the startscript "netlogo.sh". The command starting NetLogo can be extended with the classpath argument "-cp". One problem with this approach is that when different versions of the framework should be tested the start script has to be modified accordingly. However, one can use the following command to dynamically load the framework which is located in the same folder as the model is located:

```
`readlink -f $@ | xargs dirname`/mapreduce/framework.jar
```

$@ in this case is the model which should be started[2]. The drawback of this approach, of course, is that it is necessary to always load a model when using this start script.

The installation on Windows is almost similar and differs only in small details. It is assumed that users have installed the NetLogo-bundle containing a Java environment. As a result of this, one has to be careful when modifying the Java classpath. The usual way to modify the classpath (i.e. modifying an environment variable) is wrong. In order to use the framework one has to create a launcher script like in the Linux case. The best way is again to create a run script starting NetLogo with the correct parameters. A possible run script is as follows:

```
jre\bin\java.exe -classpath "extensions\mapreduce\mapreduce.jar
    ;NetLogo.jar" org.nlogo.app.App %1
PAUSE
```

The installation on Mac is not much different from the installation on Linux. The only difference are the default paths where NetLogo can be found.

Sample run scripts are of course provided along with the framework. More details and uptodate information on how to install the extension can be found at `https://github.com/mado89/netlogo-mapreduce/wiki/Installation`.

[2]To be correct $@ stands for all parameters passed to the script.

Writing a MapReduce-Program

One key design-goal of the framework was to keep the API as simple as possible. Thus a MapReduce-job can be started by a command using the following syntax:

```
mapreduce:mapreduce "mapper""reducer"value "input"
```

In this example "mapper" is the name of a procedure, "reducer" is the name of a reporter, value is the starting-value for the reducers and input is the name of a directory where the input files are stored.

A mapper takes two arguments: a *key* and a *value*. In the default configuration the key is the name of the document and the value is a line from the document. A reducer takes three arguments: a *key*, an *accumulator* and a *value*. The key for the reducer is a key emitted by a mapper. The reducer must report the next value for the reducing process.

In order not to tingle with the NetLogo execution process the `mapreduce:mapreduce` procedure returns immediately after a job is started. The termination of the job can be checked using the `mapreduce:running?` reporter which reports true while a job is running and false otherwise. However, it is important to await the termination of a job before a new job is started.

Source-Code

The delivered software is divided into two parts: the extension code and the framework itself.

The extension-code handles all manners related to extending NetLogo. The commands added are divided into two packages: the main commands and commands allowing to configure the framework. The configuration commands are not required for most use cases of MapReduce. However, one can provide a more sophisticated configuration for a job using these commands. In addition to the *mapreduce*-command the framework provides the emit command for the mapper and other commands for running MapReduce-programs.

5.4 Possible Enhancements

Visualisation

In order to understand the process of MapReduce it could be of interest to visualise the computation process. At the moment there are no visualisations implemented. Smaller datasets could be visualised through a logfile printing which task was executed with which data on which node. However, this will not be sufficient for bigger datasets. Even a small dataset resulting in only about 20 map-tasks creates a logfile exceeding 100 lines of text. A logfile of such length would not be easy to comprehend in a small amount of time. Thus, a decent simulation of the data-flow would exceed the scope of this thesis.

Display on Nodes

At the moment a node can only display the progress of its assigned tasks but not the overall progress. Both possibilities are desired. Seeing for how long the node will be occupied with tasks can be useful as well as seeing how long it will take until the overall result is available.

At the moment the difference between running a worker-node in *headless*-mode or in normal mode is small. Improving the capabilities of the framework for this matter would be beneficial for a potential class since students can be involved more in the computation process.

Scheduling

A key benefit of MapReduce comes from the fact that the programmer does not have to deal with scheduling individual tasks. At the moment the framework has no sophisticated scheduling policy. The framework creates the task-schedule in a round robin using all available nodes. The workload is spread equally across all those nodes. Productive frameworks take various factors into consideration before assigning tasks. Some of these factors are the hardware configuration of a node, structure of the network connecting the nodes, locality of the data used for a job and likewise. However, the framework is not yet able to reassign tasks when a node is behind schedule and thus slows down the overall computation of a job. One problem occurring in rescheduling of assigned tasks is that all results produced by the previous assigned task have to be removed from the intermediate result sets.

Pipelining

In [10] Condie et al. describe a modified Hadoop version featuring a pipelining mechanism. They identified a performance gain when it is possible to start follow-up tasks in advance before all input data is ready for them. In the paper two different types of pipelining are described: pipelining within a job and pipelining between jobs. The former modifies Hadoop in such a way that reducers no longer pull for their data; instead mappers push data to the reducers as soon as it is produced. Another benefit of pipelining within a job is that intermediate results can be displayed earlier. Pipelining between jobs, on the other hand, means that another job depending on the current job can be started without the - normally required - extensive intermediate computation of persistence related tasks.

One problem a possible implementation of pipelining has to deal with is the failure of tasks: Like for scheduling it is crucial to keep track what needs to be rolled back in case of a failure.

Job Dependencies

MapReduce is usually presented as a program running a single job. However, some problems require an iterative-approach where a problem is solved using consecutive runs of MapReduce-jobs. This is currently possible using the framework. Still, it is not always a trivial task since the output of the first job needs to be transformed into the input of the first job. A possible extension of the framework could decrease the amount of code required to develop iterative MapReduce-programs.

Moreover, some problems require even various different MapReduce-algorithms. This means not only that a job depends on the output of another job, but also that it uses a different configuration. Of course it is possible to develop such programs using the current program. However, one could extend the framework in order to decrease the code required to model job dependencies.

An example: a program consisting of four jobs (A,B,C,D). The output of job A is used as input for B and C, while their output is then used as input for the last job D. There are two - or more - ways to extend the framework to model this problem on a high level: as a *batch job* or as a *directed graph*. A possible extension for batch-processing could add commands to the framework providing the user with commands

- to define a procedure as a job
- to define how the output of a job is transformed to the input for another job
- commands for running a job (including waiting for a job to finish)

A very nice way of modelling a problem as a directed-graph of jobs could involve the currently experimental NetLogo-feature of importing other models to a model. By using this the user could be enabled to construct a graph using the NetLogo-GUI. Every node would represent a model including a MapReduce-program and every edge a dependency.

CHAPTER 6
A Course For MapReduce

This chapter is not intended to serve as a course book, but it presents ideas for a course on MapReduce using the framework presented in this thesis. Nevertheless, the presented ideas can be used to create a course book or design a classroom curriculum. The chapter is divided into several sections, each presenting a basic example and additional tasks for students. These tasks will in most cases be very similar to the presented example, but will require the students to work on their own. Moreover, in some cases they might have to perform further research to solve the task. Additionally, example questions are provided for some of the presented programs. These questions can also be used to create a deeper understanding of MapReduce and the presented framework. In this way the students can create their own MapReduce-program but are not "let alone in the dark" with the complete task of writing a MapReduce of a certain problem class.

6.1 General Remarks

Test Sets

When outlining a course on MapReduce, test or input-data for the students has to be provided or at least a source where it can be obtained. It is good to provide three different groups of test-sets: a group of small test-sets for testing, a group of large test-sets and an additional middle sized one, if suitable for the problem. A small test-sets will turn out to be useful for students in order to test their algorithms without spending too much time waiting for the program to finish. This is due to the fact that a real world test case can take hours to complete. If the students were given only the possibility to test with large data they would probably become rather frustrated. However, a large amount of data should be used to demonstrate the power of MapReduce and should thus also be provided for the students.

Obtaining Test-Data for Text-Processing

Project Gutenberg [19] offers a variety of free books for download. Most of the offered books are available as a simple text file. Thus a lot of books can be downloaded and used without major modifications for a course. For example a course could use MapReduce to count the words in the complete works of Shakespeare.

6.2 Word-Counting

The example program of counting word occurrences has been used as introductory example to MapReduce in nearly every presentation of MapReduce. This is not surprising since it demonstrates the capabilities of MapReduce in a very comprehensible way. It shows how the framework can be used in order to perform a task in a distributed way with only a few lines of code. Moreover, it already demonstrates probably everything that is needed in order to write more complex MapReduce-programs.

Presenting

In case students are not familiar with multi-threading or parallel programming the task can be presented in a way that the students first have to create a program that counts the words in an already familiar manner like a plain iterative program. After they have finished writing the iterative program the teacher can present the example using MapReduce. Listing A.1 shows an example of how word count can be solved without MapReduce. In order to complete the task one can use the "pathdir" extension to list files in a directory. Later in the course it can be shown that with MapReduce the task of limiting the files to ".txt" files is done with a single command using the job configuration. Still, this simple filtering of files to having the ".txt" extension can be done in *plain*-NetLogo by using the *substring* reporter and a simple *if*. However, in case wildcards are used (for example only files starting with the letter 'a' and ending with ".txt", i.e. "a*.txt") it is not trivial for NetLogo while it remains a single command for the MapReduce-framework. A sample solution for counting words can be found in listing A.2. It has to be pointed out that the main part of the source code is much smaller than the given example (see listing A.3) from [11].

What the students should take from the presentation is that with using MapReduce a simple task like counting all words in a set of files can be done in a very easy way with just a few lines of code.

Student-Tasks and Other Similar Problems

Summing

MapReduce can be used for simple things like computing the sum of a large amount of numbers. Of course MapReduce might be an overkill for this problem. Nevertheless, this task can be used to get students accustomed to writing programs using MapReduce. Moreover, in this case the mapper and the reducer have the same idea: both ignore the key passed to it and output the sum of the numbers in the value parameter. An example program could look like this:

```
to map-sum [key value]
   let sum 0
   foreach value [
      set sum (sum + ?)
   ]

   mapreduce:emit 1 sum
end

to-report reduce-sum [key acum value]
   report acum + value
end

to sum
   ...
   mapreduce:mapreduce "map-sum" "reduce-sum" 0 values
   ...
end
```

Filtering

The basic algorithm in A.2 treats “Gutenberg,” and “Gutenberg” as different words. For presenting MapReduce this is not a problem. However, the students can be encouraged to remove characters like “,” “.” and “;” from the words in the mapper. The task could be presented in a way that the students have to run the program, inspect the produced output, find out which words are not counted correctly and modify the program. This way the students will get a deeper insight into how the program works. Moreover, since they will have to modify the program on their own they will be able to perceive it as their own program. Thus, this enhancement can be used to enthuse students for MapReduce.

Anagrams

An anagram of a word is obtained by rearranging the letters in a word. For example some anagrams of the word orchestra are: carthorse, horse cart and orchestra. One can generate the list of all words being “anagram to each other” using a simple technique: a signature of each word, i.e. the sorted list of all letters in this word, is created. This means that when the signatures of two words are identical then those words are anagrams.

The task for the students could be to generate the list of all anagrams from a list of texts. In the output every line should contain a list of words which are anagrams. This can of course be done by just a few lines when using MapReduce. The mapper is similar to the word-counting example. Instead of $< \texttt{word}, 1 >$ it outputs the signature of the word as key and the word itself as value. The reducer then takes a signature and all words having this signature. Of course duplicates should/could be removed before generating the final output.

Using the Value-Separator

When using the normal input-reading procedure a map-task always receives a full line as value. For the word-count example the line is then split up within NetLogo-code into individual words. However, the framework also offers the possibility of configuring a value-separator. This feature can be used to split up a line into individual words before passing them to a map-task. Instead of a string the mapper now receives a list of strings.

This assignment can be given to students in such a way that they have to read the documentation of the framework and find out which commands are necessary and how the code has to be changed. In fact, to solve this task only two modifications are necessary:

- Before the job is started the configuration has to be changed telling the framework how the values should be separated. This can be done by invoking
  `mapreduce:config.valueseparator "␣"`
  before the call to `mapreduce:mapreduce`.

- The mapper has to be changed: The call to the self-written split reporter is removed. Instead the foreach-loop will just loop over the values passed to the mapper.

On a side remark, it has to be pointed out that there will be no or only little performance gain in facilitating the value-separator. However, students can be asked to explain the reason for this. The reason for this is probably mostly due to the fact that the NetLogo-code responsible for splitting a line is compiled and optimised before its execution in the mapper. However, the students' attention should be drawn to the fact that using the value-separator will reduce the amount of code to be written and thus the likeliness of errors.

6.3 Pi

Π is the mathematical constant describing the ratio of a circle's circumference to its diameter. It can be estimated by a very simple experiment. However, this yet simple experiment is another example where using parallelism accelerates the computation significantly. NetLogo usually does not perform code in parallel. Thus, when performing a significantly large experiment the MapReduce-code will run faster than *normal* NetLogo-code. This fact, however, is the main intention behind this task: Students should understand and feel that parallel code scales better for larger data sets. As a result of this the NetLogo-model solving the task of calculating Π should be created in a way that different experiment sizes can be chosen easily.

Mathematical Facts

The following mathematical knowledge has to be taught to the students so that they are able to understand the experiment. A circle with radius R can be inscribed in a square with side length $2R$. Naturally, the following areas can be measured: the square has an area of $(2R)^2$ and the circle has an area of $\pi * R^2$. Moreover, it can be established that the ratio of the area of the square to that of the circle is $\frac{\pi}{4}$.

Π can now be estimated by first picking N points inside this square at random and then checking how many of these are also inside the circle. This number is approximately $\frac{N*\pi}{4}$. In order to compute the number a number N is fixed. Then N times two numbers x and y in the range $[-1, 1]$ are generated at random. Every generated point is then checked whether it is within the circle by just checking whether $x^2 + y^2 \leq 1$. Pi can then be computed by the following formula: $\pi = \frac{4*M}{N}$ where M is the number of points within the circle.

Modelling the Problem Using MapReduce

One interesting aspect behind this task is that it demonstrates how MapReduce can be used to perform a task repeatedly independent from the computer where it is executed.

Clearly the point generation should be done within the mapper. Doing so the point generation could be performed on a cluster. Before creating the mapper one has to think about how the mapper can be used in order to create the points. In order to create a small model all mappers should always create the same amount of points. One way to do so is that the mapper creates the point independently from its input. The input to the mapper can be used for setting a random seed to be used for the point generation.

To be able to run the experiment the input to the MapReduce-program has to be created. This can be done on-the-fly within the model and then be passed to MapReduce as a list. The students need to understand that a list of key-value pairs has to be generated. Each of these pairs will result in an invocation of a mapper. One way to set the random seed is that the value in these pairs will be different while the key remains the same.

Sample Solution

Listing A.4 gives a sample solution for computing Pi. It uses two sliders “mapper” and “repeats”. Those can be used to experiment with different configurations.

Comparing MapReduce to Imperative Programming

Since the idea behind the program is simple it can be modelled as a linear program as well. As a result of this students can be asked to compare both approaches. While the amount of code to be written remains the same the running time will scale differently. For a small amount of points to be produced an imperative program will be faster since the setup of a MapReduce-job is time-consuming. However, when increasing the amount of points to be tested MapReduce will become faster since it scales better. In case the students are familiar with other programming languages allowing parallelism they can be presented with an example solution in this language. This solution will most likely be much more complex since it requires the spread and collection of data.

Thus, this yet simple program can be used to demonstrate the fact that MapReduce can be used to perform large scale computations.

6.4 Inverted Index

This task demonstrates the usefulness of MapReduce for search engines. For the inverted index the task is to determine which word appears in which documents. It appears as a subtask in various other problems like Page-Rank (see 6.8), for example. Thus it is beneficial to present this problem to the students before tackling bigger applications of MapReduce. The skills required for this task can be trained using the word count example. The working principle remains the same while the algorithm is changed slightly.

A problem statement like this can be given to the students: *"Given a set of documents output the words occurring in them. Moreover, for every word output the documents in which they appear."* The hint that it is similar to counting words can also be given to the students, especially to younger or inexperienced programmers.

It is important that the students are familiar with using lists. In case the students are not experienced with lists the following code-snippet can be presented or created together. The snippet can be facilitated in the reducer in order to check whether a value is in the list of files or not.

```
to-report in-list? [value #list]
 ifelse length filter [? = value] #list > 0 [report
    true][report false]
end
```

A solution to the problem can be achieved as follows. The mapper emits for every word the name of the word as key and the name of the file as value, while the reducer just removes all duplicates from the list of values. When a solution is created together as a team it might be a good approach to first skip the step of removing duplicates. That way students get more sense of the ideas behind MapReduce since they first have to construct the identity-reducer which just builds up a list from all values that are passed to it. After having built this identity-reducer they will realise that the framework has no knowledge about the fact that the values for a key might contain duplicates. A sample solution for the problem is given in listing A.5

Enhancements

Word-Filter

When using a large data-set the index soon becomes very large. However, some words included in the index like "a", "and" and "the" might not be interesting for further investigation. Thus, the students can be asked to remove words shorter than a certain length or remove words from an input list. Clearly, this has to be done in the mapper. A simple if-statement checking for the length of the word before emitting the key-value pair will solve this task. When aiming for a more sophisticated version one might use the `in-list?` reporter to check if a word appears in a list of words to be excluded from the index.

Filter for Frequent Words

After MapReduce featuring multiple jobs has been introduced, students can be asked to extend the program so that it will only build the index for the most frequent words among a set of files. A naïve program solving this task can use the word-counting as a pre-processing step. The output of this step can then be sorted by word-frequency. Thus at the top of the list the most frequent words in the input data are listed. After defining a certain threshold the creation of the index is straightforward. The mapper only needs to check whether a word is within the threshold before emitting the key-value pair.

6.5 Sorting

In fact, sorting as a task is used as a benchmark for MapReduce frameworks [22]. This comes from the fact that any MapReduce-framework needs to follow certain standards. The algorithm behind sorting is trivial once the principle of MapReduce is understood. It has to be pointed out that the presented framework most probably will not achieve good results for a benchmark due to its design. The overhead needed for the workspace creation is too high, for example. Moreover, the aim of the framework is not competitive to other frameworks like Hadoop.

One idea behind using the sorting example in a course is that its mastery most probably requires a deeper insight into MapReduce. MapReduce can be used to sort data easily because of the following fact:

“The keys are sorted before the reducing stage”

This means that in order to sort the input data the program has to somehow transform/pass all items in the input as keys to the reducing stage. In this stage all items can be persisted using an identity-reducer. Such an identity-reducer performs no further transformations on its input. Instead it just outputs its input. Although this framework uses functional MapReduce-style for reducers creating an identity-reducer is quite similar. Instead of emitting a key-value pair here the reducer just returns an empty string. As a result the framework will output just the key as a line in the output.

Questions for Students

What Will the Output of the Sorting Procedure Look Like?

What the students should be able understand is the fact that the output will only contain distinct items. Moreover, they should be able to argue why this is the case. The answer to this question is also closely related to the next question.

Is It Possible to Keep the Frequency?

The answer to this question is a bit tricky. In general yes. One could construct a reducer building up a list using the key. Thus, the result for a particular key would be a list having the length of the number of times the item occurs in the input and each item in this list is just the key itself.

However, in particular it is not really possible. When using a reducer like the one described the output would contain on every line a distinct item and a list of how many times this item occurs in the input. Still, when a feature to select different writer-modules for the output is added to the framework it would be possible to create a meaningful output. However, this would require the students to write their own output-module.

Another workaround would be using the `mapreduce:emit` command to produce multiple outputs from an reducer. However, this can be tricky since the reducer once finished will also produce a key-value and thus result in a duplicate in the output.

In summary, when presenting the question to the students they should also be asked to justify their answer. Thus it can be assured that they understood how the framework works. Possible answers are:

- Yes, the frequency can be kept in case the framework is modified.
- Yes, it can be kept, however, the output is not clear.
- No, it cannot be kept using this particular framework.

Difference to Word-Counting

After the introduction of how to create a sorting-program using MapReduce the students can be asked to think about the difference to the word-counting program. It should be clear to all students that the only significant difference between the two programs is the data emitted as values both in the mapper and reducer.

6.6 Graph Problems

Graphs are ubiquitous today. Thus, it is important to present the topic of graphs to the students in a way that they get the feeling that graphs are more than just a mathematical construct. For this course the students might not yet be familiar with graphs. Material on graphs and their presentation to students can be found on the internet at various occasions. However, some basic considerations for graphs and their use in MapReduce problems are outlined in this section.

One way of presenting graphs to students can utilise social networks as an example. Twitter can be used as a representative for graphs with directed edges while Facebook for graphs with undirected edges. Moreover, after the introduction of the "following" button Facebook also includes directed edges. This fact can be used to test the students' understanding of graphs.

Graph Representations

Before dealing with graph-algorithms one has to think about how to store and process a graph. There are two commonly used types of storing a graph:

- an adjacency matrix or
- an adjacency list.

Both of them have their strengths and weaknesses.

Adjacency Matrix

An adjacency matrix is a matrix with NxN entries, where N denotes the number of vertices in the graph. Every entry in this matrix denotes whether a vertex is connected to the respective other vertex. Additionally, it is also possible to store information like the distance between two connected vertices within these entries.

Adjacency matrices offer a good performance since the costs for operations like checking the connectivity of two vertices and connecting two vertices are low. This is due to the fact that array operations are computationally cheap.

One disadvantage of adjacency matrices is that they need to have a fixed size. This means that the size of a graph is not intended to be dynamic. However, workarounds using dynamic arrays or lists are also possible but harder to maintain. Another disadvantage is the space required for the matrix, especially for sparse graphs where most entries in the matrix remain empty. One example for such sparse graphs are social networks [16]. In a social network typically a huge amount of vertices exist, but compared to the amount of vertices every vertex has only a few connected vertices. Moreover, NetLogo does not support arrays directly but only via the *table*-extension. Thus it is rather complicated to use an adjacency matrix within NetLogo, but not impossible.

Adjacency List

In contrast to the matrix the adjacency list is a - unordered - list of lists. An entry in this list is a vertex in combination with all its adjacent vertices. Similarly to the adjacency matrix it is also possible to store additional information for edges in an adjacency list.

Adjacency lists are very space-efficient since only the edges of a graph are stored in the list. This space-efficiency is most valuable when sparse graphs are stored. Instead of a huge amount of empty cells as in the case of an adjacency matrix only the required information is stored.

However, the downside of this space-efficiency of the list are the computation costs required for most operations. While the connectivity of two vertices can be tested in constant time using an adjacency matrix, the time is linear to the number of vertices in the case of an adjacency list.

Graphs in MapReduce

At first sight it might look as if the only way of processing a graph with MapReduce is using an adjacency list. A classic algorithm would most likely use an adjacency list. Every mapper would perform an operation for a certain node (key) and its adjacent nodes (values). However, it would be possible to use a global adjacency matrix and perform the mapper using the vertex indices from the matrix as key-value pairs. A mapper or reducer could access this matrix via these indices. The problem with this approach is that updates within a particular task would affect all other tasks. This means that changes to the matrix need to be transferred to all workers. Thus, when updates to the representation of the graph need to be done, using a kind of adjacency list is the easiest approach.

Building a Graph with NetLogo

Although students are required to be familiar with NetLogo they are not required to be able to create a model for constructing a graph using NetLogo. Doing so requires a deeper, more than just basic understanding of all concepts of NetLogo. However, once these concepts are understood it is fairly easy to create a simple model in which nodes can be added and deleted and then further be connected and disconnected. On the other hand, NetLogo does not seem to be suited for graphs. One evidence for this is the fact that as of September 2013 among 1200 models uploaded to Modelling Commons only about 30 models are related to graphs or networks. Nevertheless, using its full potential NetLogo can be used for creating or displaying a graph. A key benefit of NetLogo when it comes to complex algorithms is that once an algorithm is created it is also easy to visualise the data. For other programming environments in most cases a third party library has to be used when a graph should be displayed. A sample solution of how to create a graph can be found in listing A.6.

As one goal for this course is to provide students with models which are fun to play with, it is important to have the ability to construct graphs with mouse clicks rather than writing or editing configuration files. Thus it is important to teach students how the mouse can be interfaced with NetLogo. One key problem for creating a graph "on the fly" is that NetLogo does not offer mouse-handlers or similar concepts offered in other programming languages like Java. This results in a rather complicated approach for handling mouse clicks. The sample solution uses an algorithm very similar to the algorithm proposed by Jim Lyons in a post on the NetLogo-mailinglist[1]. What needs to be understood is that instead of a handler a mouse-handling function has to be invoked from a *forever*-loop. Other than registering handler-functions these functions need to be invoked accordingly within this *handler*-loop. The code example demonstrates only the use of a "*mouse-clicked event*". However, a task for students could be to extend this example with further events like a double click to remove a vertex from the graph.

NetLogo offers the concept of breeds. This means that one can create a breed of turtles. All members of a certain breed share some common attributes or behaviours. However, they still can be controlled in similar ways as turtles are controlled. As a benefit of this students do not have to learn a lot of additional commands. Once breeds can be used by the students their models will become more readable since different kinds of turtles, which previously could only be distinguished by understanding the context, can now be read directly from the code.

After having constructed a graph it has to be converted into a key-value format for the MapReduce-computation. When an adjacency list is used this is straightforward since one can iterate over all nodes and create the list of adjacent nodes. To elaborate a bit more: In NetLogo this iteration process can be done using the *ask* directive. This directive executes a command-block for all members of a specified agent set. At this point one can see the benefit of using breeds. A simple `ask nodes [...]` will execute a command block for all nodes in a model. The next helpful aspect of NetLogo for this matter is the *out-<breed>-neighbors* reporter. It reports the connected nodes for a specific node. Summing up all these features of NetLogo, the only task which might not be solved as elegantly are the list operations required for the cre-

[1] `http://groups.yahoo.com/neo/groups/netlogo-users/conversations/topics/7374`

ation of the adjacency list. This is due to the fact that in this approach the creation of the list requires a repeated alternation of the list which involves a lot of set operations in combination with appending and the creation of new list elements.

However, it has to be noted that the sample program lacks the possibility to save and load a constructed graph. Still, there are several code-examples available on Modelling Commons; for example, demonstrating how such a feature can be implemented[2].

Shortest Path

One of the most common graph problems is the shortest path problem, i.e. the problem of how to get from a vertex A to a vertex B with the lowest costs [35]. There are two basic types of shortest path algorithms:

- **single source shortest path**: All shortest paths starting in a given vertex are determined. This problem can be solved by Dijkstra's algorithm or the Bellman–Ford algorithm.
- **all pair shortest path**: All shortest paths for all possible pairs of vertices are determined. One algorithm solving this problem is the Floyd-Warshall algorithm [35].

Single source shortest path algorithms are studied in every course on graphs and are moreover not particularly difficult to understand. Additionally, most of these algorithms work iteratively like the famous Dijkstra algorithm, for example. As a result of this a shortest path algorithm can be used to demonstrate that it is possible to use MapReduce together with iterative algorithms.

Solving the Problem

One way of developing a model for solving the single source shortest path problem together with students is to first implement a plain Dijkstra algorithm [13]. The algorithm is outlined in figure 6.6, while a sample implementation can be found in listing A.7. The benefit of doing so is that the students can then be asked to transform the algorithm to a MapReduce algorithm on their own. The idea behind this is that they only need to transform the body of the iteration into a MapReduce-algorithm. Moreover, the challenges of parallel algorithms need to be faced. One of these challenges is how to determine the termination.

The classic Dijkstra performs a search similar to a breath-first-search (BFS) and is done once all vertices are processed. However, this behaviour cannot be easily transformed to a MapReduce-program. A naïve parallel implementation could thus iterate as often as there are vertices in the graph. However, this would be a dramatic overhead. One argument for this can be found - among others - in the so-called small world phenomenon [27]. In his small world experiments Stanley Milgram tried to determine the average path length in social networks. For this purpose he sent letters across the United States of America containing instructions of how to pass on the letter to a certain person. The persons were only allowed to forward the letter to a person they knew on a first name basis. The study yielded an average path length of 5.5

[2]For the interested reader one needs to use the `import-world` and `export-world` primitives.

```
    Data: Digraph G with $w_{ij} > 0$ for all $(i, j) \in A(G)$
    Data: source node $s$
1  $\forall v \in V$: $\pi_v \leftarrow NIL, \delta_v \leftarrow \infty$;
2  $\delta_s \leftarrow 0; F \leftarrow \varnothing; Q \leftarrow V$;
3  while $Q \neq \varnothing$ do
4      $u \leftarrow Q.getMin()$;
5      $F \leftarrow F \bigcup u$;
6      $Q \leftarrow Q\ u$;
7      forall the $v$ adjacent to $u, v \notin F$ do
8          if $\delta_v > \delta_u + w_{uv}$ then
9              $\delta_v \leftarrow \delta_u + w_{uv}$;
10             $\pi_v \leftarrow u$;
11         end
12     end
13 end
```
In this algorithm w denotes the distance between two vertices. V is the set of vertices in the graph. δ is a vector containing the length of the shortest path to the vertices. π is a vector storing the predecessor for the vertices on the shortest path.

Figure 6.1: Dijkstra Algorithm. Taken from [26]

before reaching the target person. Since the algorithm performs only a search starting from a pre-determined vertex, a single source shortest path algorithm can terminate after the longest path is examined.

Considering the behaviour of the algorithm one solution to determine the termination of the algorithm is to find out whether no entries in the distance vector have changed after an iteration. However, it has to be noted that a simple checking routine would have a runtime linear to the number of vertices and thus should also be optimised.

Moreover, this algorithm can be used for teaching the analysis of algorithms to the students. The total number of iterations is $|V|$. Each iteration analyses the edges of the current node and thus runs $|E|$ times. It has to be noted that no edge is processed twice. Thus, the basic runtime for the algorithm is $\mathcal{O}(|V|+|E|)$. For the Dijkstra-algorithm the runtime depends heavily on the data structures used in the implementation. Q keeps track of the vertices still to be processed. In each iteration the vertex with the lowest distance needs to be found (Q.getMin()). If Q is implemented as a static array it takes at most $|V|$ steps to determine the current minimum. Thus, the runtime will be $\mathcal{O}(|V|^2 + |E|)$. However, if Q is implemented as a binary heap, finding the minimum would decrease to $\mathcal{O}((|V| + |E|) * \lg |V|)$ since finding the minimum will only take $\lg |V|$ time.

The MapReduce-algorithm as presented has a runtime of $\mathcal{O}(L * |V| * |E|)$. In this term L denotes the length - number of edges - of the longest path in the network. L is thus the number of iterations performed. In each iteration there will be a total amount of $\mathcal{O}(|V| * |E|)$ mappers, since for every vertex and its adjacent vertices a mapper is created. Every mapper has a constant runtime. Moreover, there are $\mathcal{O}(|V|)$ reducers which have a runtime of $\mathcal{O}(|E|)$. Of course when the framework is tuned differently and the mapper uses the edges as values, the runtime does not change. Not accounted in the runtime is the checking routine for the termination of the algorithm. As mentioned previously this can easily reach a runtime of $\mathcal{O}(|V|)$, which would cause the runtime to increase to $\mathcal{O}(L * |V|^2 * |E|)$. When comparing the two approaches one can see that the runtime is similar when normal data structures are used. Only the introduction of clever data structures will provide an implementation of the Dijkstra algorithm with a quicker execution.

A Remark on the Framework

As mentioned previously unlike most MapReduce-frameworks the presented framework uses a functional approach for the mapper. As a result of this some algorithms have to be implemented differently or have to be rethought before they can be implemented. Lin and Dyer, for example, present in their book [25] an algorithm (see listing A.8 for details) for solving the shortest path problem which is not only lexically different from the algorithm presented in this thesis but also in its working principle. While the Lin-Dyer-algorithm computes the distance using a data-structure for the graph which is passed along in order to “keep” the graph from one iteration to the next, the algorithm in this thesis uses the adjacency list in every iteration as input for the next. Additionally, in the book the reducer needs to check whether a value passed to the reducer is a “*graph-structure value*” or a *distance-value* in order to preserve the structure of the graph between the iterations. What is more, the algorithm presented in this thesis outputs a distance vector, while the Lin-Dyer-algorithm outputs a graph annotated with distances.

Extensions

Values The presented sample solution only uses a uniform distance of 1 between all connected vertices. A more sophisticated model would allow the user to additionally set distances for the edges when creating the graph. The shortest path computation is then computed with respect to these distances rather than the sum of edges.

Path Currently the model will only compute the length of the shortest path between connected vertices. However, in most cases also the path is of interest. A solution to this would need to pass more data to the reducers.

Persistence The sample solution is not equipped with persistence methods. Thus, as mentioned previously, graphs can only be stored and loaded using the export/import world functionality of NetLogo. As a result of this, it is hard to examine problem sets available from domains different from this course. However, there are already some models available on Modelling Commons tackling this problem.

6.7 K-Means

K-Means clustering or just k-means is a method using vector quantization for cluster analysis. In machine learning or data mining a cluster is a group of objects sharing certain properties. The

```
1 Mapper Compute nearest centre
  Key: Node-ID
  Value: Node-Coordinates
2 nearestCentre= 0
3 forall the centers $c$ do
4     if $c$ closer to Node-Coordinates than nearestCentre then
5         nearestCentre= $c$
6     end
7 end
8 Emit (nearestCentre, Node-Coordinates)
```

```
1 Reducer Update Centres
  Key: centreId
  Accumulator: Centre-Coordinates
  Value: Coordinates (of a single point)
2 update coordinates using Centre-Coordinates and Coordinates of the current point
3 return new coordinates
```

Figure 6.2: K-Means using MapReduce

algorithm is used to partition a data set of size n into k clusters where every data point belongs to the cluster with the nearest mean. This mean or centre point does not have to be in the input data set. K-Means clustering can be applied to all sorts of data as long as there exists a function for measuring the distance between two vectors. In a simple case k-means can be applied to two dimensional vectors, i.e. points on a plane. The algorithm then computes k groups of points belonging to the same region. Of course this problem cannot be solved easily (computationally-wise) since the algorithm involves a repetitive computation of distance to all the centre points for all data points.

However, as shown below k-means can be easily formulated as a MapReduce-program. Another aspect of the problem which makes it of particular interest for using it as a task in an introductory MapReduce-course is that it can be implemented for its two dimensional cases. Thus, the problem of computing the distance between two points is not difficult since it is only the euclidean distance. Moreover, any two dimensional data can be visualised using NetLogo in an easy way since a point can easily be represented by an agent.

Algorithm

Using MapReduce k-means can be computed in a straightforward way. In the mapping stage for every vertex its closest centre is determined. The mapper then emits the ID of this centre and the coordinates of the vertex which is being processed. Due to this each reducer gets a cluster, i.e. they get the ID of a centre point and a list of coordinates which belong to this cluster. Thus, the reducer can compute the new coordinates for the centre. Figure 6.2 gives a high level description of the mapper and reducer for K-Means.

Sample Implementation

The listing A.9 shows a sample implementation of the algorithm presented above. However, the implementation is not complete since it does not compute convergence of the algorithm but only runs for a fixed number of iterations.

6.8 PageRank

Introduction

PageRank [29] is the idea of how to bring order to web pages of a search engine. The name is also a world play on one of its inventors: Larry Page. When searching the web for a certain piece of information using a search engine one does not simply want to have a list of all web pages containing relevant information, but also a ranking of these pages. The problem is how to define this order. A search engine like Google is a very complex system of several partially independent components [6]. PageRank is the component required for ordering the search results. A potential teacher is advised to read both papers [6] and [29] since they contain information about the building blocks of search engines and are thus of particular interest not only for this course.

PageRank measures the "relative importance of web pages" [29]. In general one might say that a web page with a high number of backlinks (in-edges/links) is more important than these with just a few backlinks. However, such a behaviour would be prone to destructive behaviour since someone could create a so-called link farm, a collection of web pages linking to other pages just for the sake of creating backlinks to these pages. Other problems are "*rank sinks*": pages that form a link circle and have some backlinks, for example two pages that only link to each other and some web pages link to one of these two. Such a sink could cause the rank of the sink accumulating with every (re-)computation when the rank is computed, since it is not distributing any rank [29]. Another idea behind PageRank is that it should model a random surfer: a surfer who randomly follows the links on the web pages but will not follow a loop of pages.

PageRank as a Course-Project

One aspect of PageRank making it an interesting subject to study in a course is that it involves a set of different MapReduce-algorithms. Thus, for solving one problem (i.e. PageRank) several sub-programs have to be created, which helps practising MapReduce programming skills. Moreover, it is an iterative algorithm. The PageRank algorithm works in three stages:

1. *Create Graph.* The data for the algorithm can come in different forms. One form could be as plain HTML-files. Each of these files needs to be parsed in such a way that the result is a graph where websites are the vertices and the edges are links between them.

2. *Invert Graph.* After the first stage a vertex has a number of out-edges representing the links on the page. However, for the computation the in-edges for a vertex are of interest. Of course, a clever algorithm design could omit this step. Still this would slow down the

algorithm since the main-computation is performed several times. The result of this step is a graph which instead of out-edges consists of in-edges.

3. *Compute PageRank.* In the last step the PageRank of all web pages in the data set is computed by an iterative MapReduce-program. It has to be pointed out that this step takes place in the reducing step of the graph inversion.

What is more, the structure of the problem gives the possibility to divide it into parts which can be solved separately and is thus interesting for teaching programming in a team.

Remark: One problem not solved in this thesis is the task of generating or obtaining test data for the algorithm. One way to create a data set is to download a set of web pages linked from a news site. This set of pages should then be extended with other web pages appearing in the search results of terms included in the news articles.

Developing a MapReduce-Algorithm

Due to the complexity of the program and the problems described above and below, this thesis does not give a complete or sophisticated solution for computing PageRank.

Creating the Graph

One thing to be taken into account is that the URLs of the web pages should be transferred to unique IDs. This could be done by using a hashing algorithm like MD5, for example. Moreover, before doing so a URL should be brought to a canonical form. For example the URLs `www.host.com/folder/site.html` and `Host.com/folder/site.html` should be treated as equal. A vertex in the graph represents a document or page in the world wide web. Whenever a page links to another page an edge between the vertices representing those documents is inserted into the graph.

In order to keep the NetLogo-model simple one can assume that the URLs are already converted into numbers.

Inverting the Graph

This problem is in this case very similar to the Inverted Index problem described in section 6.4. However, the algorithm for the plain inverted index is slightly different from the algorithm for inverting the graph as part of the PageRank computation. Instead of indexing documents the inverted index lists for very document the documents linking to it. In the algorithm for plain inversion the mapper emits for every out-link in a document the key of the document to which the link points and as value it emits the key of the document which is currently processed. The reducer, on the other hand, can then easily *collect* all the in-links for a document. However, for PageRank the mapper will distribute the rank of a vertex to all its out-links. This means that for every adjacent vertex in the graph a key-value pair is emitted where the key is the adjacent vertex and the value is the rank to be distributed. The reducer will then be used for calculating the new rank of a vertex.

Rank Computation

The paper [29] gives one formula for computing the rank of a page:

$$R(u) = c * \sum_{v \in B(u)} \frac{R(v)}{|F_u|}$$

Whereas $R(u)$ describes the rank of a page u, $B(u)$ are the backlinks for the page u, i.e. the pages containing a link to u. $|F_u|$ is the amount of links going out of the page u and c is a constant used to normalise the values [29].

As one can see the ranks are highly dependent on each other. Thus, it is not possible to compute the ranks directly. Instead they are computed iteratively until the difference between the ranking vectors is smaller than some ϵ.

Due to the inversion of the graph it is rather straightforward to compute the rank, since it is only necessary to sum up all *gained* ranks. Figure 6.3 contains an algorithm outline for computing PageRank. The process of generating the inverted graph and computing the page rank has to be repeated until convergence has been reached. One simple criterion for convergence is that the sum of changes of the ranks is below a certain threshold. In [29] it is mentioned that for 322 million pages the algorithm converges within roughly 52 iterations, while for half of the data it takes about 45 iterations.

Sample Implementation

Listing A.10 gives a sample solution for computing PageRank. It simplifies the problem by fixing the format to the input in the following way: The input to the model is a simple text file where every line describes one web page. Each of these lines starts with the ID (a numerical value) of the web page, its initial rank (also a numerical value) followed by the ideas of the pages it links to.

In order to efficiently keep track of the current ranks of all pages the model uses the table extension provided with the standard installation of NetLogo. Before the PageRank-computation starts two pre-processing MapReduce-jobs are done in order to determine the number of out-links and initial rank for every web page. Then the actual PageRank computation starts. The mapper (`distributerank`) reads the first value of the line (i.e. the ID of the website) and then emits for all out-links the current rank of the website. The reducer, on the other hand, is very simple as it just sums up all ranks for a website. After the job has finished the new rank of every page is divided by its number of out-links according to the formula shown above (however, no factor c is used).

Extending the Model

Algorithm

Figure 6.3 shows a high level algorithm for computing the PageRank. When carefully examining the sample solution in listing A.10 students should see that the pre-processing step does unnecessary work, since it computes the number of out-links. However, this number can also be determined by every mapper during the actual computation. Changing the code in this case should be a small task, since it involves the removal of the first pre-processing MapReduce-job and the change of mapper and reducer of the main MapReduce-job.

Convergence

Currently, the model does not compute the convergence of the algorithm but rather runs for a fixed number of iterations. One task for students can be to change the algorithm to determine the convergence of the algorithm or to stop the computation after a fixed number of iterations. As described in [29], the computation converges once the change between two iterations is below a certain threshold ε .

Input Data

At the moment the model is not ready for real world applications. This is due to the fact that it simplifies the input to the algorithm. One task for students could be to build the input to the algorithm from a real set of web pages. These web pages can be stored in a folder which serves as input directory for a MapReduce-job. The files could then be processed and every link could be extracted. However, a solution for this task also needs to solve the task of how to transfer URLs to a unique ID.

Dangling Links

One problem of the algorithm as presented above and PageRank in general are so-called dangling links. Dangling links are pages with only in-links but no out-links. Such a page can be the result of not yet completed download of the page. Thus, it is known which pages link to that page but not to which pages this page links to.

The problem with dangling links is that it is not clear how the rank of the page should be distributed. A solution to this problem is to simply remove dangling links from the data before computing the PageRank as a pre-processing step and add them again to the data once the computation is done [29]. This removal can also be done using MapReduce since it involves computing the in- and out-degree of every page in the data.

```
1 Mapper
2 $p$ = pagerank($key$) / length $values$;
3 forall the $v \in values$ do
4     Emit($v$, $p$);
5 end
```

```
1 Reducer
  Key: Site
  Values: Rank of other sites linking to that site
2 $sum = 0$;
3 forall the $p \in values$ do
4     $sum = sum + p$;
5 end
6 Emit($key$, $sum$);
```

Note: The reducer is written in classic MapReduce-style rather than a functional in order to keep the syntax to a minimum.

Figure 6.3: Computation of PageRank

CHAPTER 7

Summary And Future Work

Summary This thesis presented a MapReduce-framework for the use in class rooms.

First, MapReduce was introduced and analysed in chapter 2. The next chapter, chapter 3 presented relevant learning theories for this thesis and outlined conclusions for a course on MapReduce. Chapter 4 presented existing MapReduce-frameworks and discussed criteria for a framework to be used in education. The results from this chapter were then used for the implementation of MapReduce for NetLogo which was described in chapter 5. This chapter presented the framework, its implementation, problems during the implementation and how the framework can be installed. Chapter 6 provided information on how the framework can be used.

MapReduce for NetLogo was designed according to the criteria examined in chapter 4. One of the most important criteria for a framework to be suitable for teaching is an easy installation process. MapReduce for NetLogo provides a simple installation which mainly consists of copying files to a directory. Another important aspect is that the framework can be used to solve a broad variety of different tasks. This thesis showed how the framework can be used by presenting and analysing sample tasks. The presented tasks are all well suited for using MapReduce to solve the problem. All of these tasks were solved with at least a partial implementation using the MapReduce for NetLogo.

Comparing the framework with other frameworks One aspect making the framework different from others is that it works almost "out-of-the-box". There is no need for a complicated installation process or modification of configuration files as is the case for other frameworks like, for example, Hadoop. In the unlikely event of a non-standard installation of NetLogo or system components like Java, a potential user just has to adapt a single line in the run script pointing to the Java interpreter or the installed MapReduce-framework.

One of the frameworks presented in section 4.2 sticking out from the others with respect to its usability when it comes to teaching MapReduce was WebMapReduce. A detailed analysis of WebMapReduce is laid out in subsection 4.2. In contrast to WebMapReduce the framework provided in this thesis offers the full functionality of a programming language (i.e. NetLogo) and the ability to test on own data sets. Moreover, by using the developed framework it is possible to create iterative algorithms which is not possible using WebMapReduce. Additionally, MapReduce for NetLogo can run on any data set, while large data sets on WebMapReduce need installation by an administrator.

The presented framework is more suited for teaching than other frameworks in some cases, especially for introductory tasks since the number of lines of code needed is lower than for other frameworks. This comes from the fact that object oriented Java requires more code for the main method. Additionally, the setup of a MapReduce-job is also lengthy in Hadoop. Having less lines to write is a desired goal for teaching since it is less prone to errors and saves time for more important aspects of the subject.

One aspect in which the presented framework differs from other frameworks is the functional design. One problem of MapReduce- as Hadoop and similar frameworks implement it - is that MapReduce suggests a functional design while MapReduce is not purely functional. While map and reduce suggest a functional programming paradigm, i.e. function that takes parameters and return values. Neither the mapper nor the reducer actually return a value. Both use the emit function to deliver values to the framework for further processing. The framework presented in this thesis partially uses a functional style. While the reducer is implemented in a functional style, the mapper still uses an emit function. This is due to the simple reason that a mapper can have more than one key-value pair as a result, forcing the user to manipulate lists which then would have to be checked for their correctness by the framework. In order to avoid problems by design, the mapper uses an emit function instead of a returning values.

One downside of the framework in its current version is that it is not able to compete performance-wise with commercial frameworks or frameworks for real world use like Hadoop. However, it has to be pointed out that most of the MapReduce-frameworks lack this possibility. For example, the Lisp framework from UC Berkeley was claimed to have had a start up time of about 30 seconds in its early versions. The lack of performance has (sadly) a couple of reasons. One reason of course is the lack of parallelism in NetLogo as mentioned in section 5.2. Due to this there is always a significant overhead of computations for opening workspaces and importing the current world. This creates - depending on the available memory on the computer - high start-up costs. However, it is hard to compare start up costs with other frameworks since not all frameworks offer this information to the user. Another reason for low performances in some cases is that the framework has no access to a distributed file system.

Future work The framework has yet to be *stress tested* in a real classroom experience. One reason for not testing the framework in an actual classroom setting in a school is the lack of schools teaching NetLogo in Austria. Thus, testing the framework in a school is difficult since it would also involve teaching programming and NetLogo prior to teaching MapReduce. Testing in real-world classroom would have provided feedback on the necessity of possible enhancements for the framework, especially the enhancements mentioned in section 5.4.

As mentioned in the summary, MapReduce for NetLogo lacks the possibility to integrate in an existing DFS. Thus, one possible future work is to add support for HDFS which should be feasible within a reasonable amount of time since Hadoop offers an API for its file system. Adding support for a DFS would also enhance the use of better schedules for individual tasks.

The feature to be implemented in the future is most likely a better scheduling algorithm for the tasks. Thus, the framework could be able to close the performance gap to other frameworks. Another feature which is subject to be implemented is of course the pipelining.

The current thesis presented some sample tasks which can be solved using MapReduce. In order to promote the framework further, more tasks have to be described and solved using MapReduce for NetLogo. For a good promotion of the framework the NetLogo-models should be offered to the public using the Modelling Commons platform. All of these models then need to be tagged with a MapReduce tag for better availability through the search function on Modelling Commons.

APPENDIX A

Code Examples

A.1 Counting Words

This program requires the installation of the “pathdir” extension which can be found at `http://sophia.smith.edu/~cstaelin/NetLogo/pathdir.html`. The program counts the words for every file found in the directory “input” using an imperative iterative algorithm.

Listing A.1: Counting words iterative

```
extensions [pathdir]

; Split a string into individual words
; As suggested in
    http://groups.yahoo.com/group/netlogo-users/message/6490
to-report split [ #string ]
  let result [] ; return value
  loop ; exit when done
  [
    let next-pos position "␣" #string
    if not is-number? next-pos
    [ report reverse (fput #string result) ]
    set result fput (substring #string 0 next-pos) result
    set #string substring #string (next-pos + 1) (length
        #string)
  ]
end

; Read a file and report all words found in it
to-report read-file [file-name]
  let words []
  file-open file-name
  while[not file-at-end?][
    foreach split file-read-line
    [
      set words fput ? words
    ]
  ]
  file-close
  report words
end

; Count the words in a file and write the result to result-#fn
to count-words [words #fn]
  show #fn
  let resultfile (word "result-" #fn)
  let result []
  while [length words > 0]
  [
    let curword first words
    let #words length words
    set words filter [? != curword] words
    let #new length words
    let #count (#words - #new)
    set result fput (list curword #count) result
  ]
  if file-exists? resultfile [file-delete resultfile]
  file-open resultfile
  foreach result [file-print ?]
  file-close-all
end

; Count the words in files
to word-count
  let words []

  foreach pathdir:list "input" [
    let file-name word "input" word pathdir:get-separator ?
    show word "Scaning␣" file-name
    set words sentence (read-file file-name) words
    count-words words ?
  ]
end
```

Listing A.2: Counting words MapReduce

```
extensions [mapreduce]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; Mapper
;;;; Read the line of a file, split it into words, emit word
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to read-file [file-name words]
  ;;; Loop over all words
  foreach words
  [
    ;;; emit: <word, 1>
    mapreduce:emit ? "1" ; <word, 1>
  ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; Reducer
;;;; Sum up the occurrences of a word
;;;; key the word
;;;; accum current count
;;;; value next value
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to-report word-count [key acum value]
  report acum + read-from-string value
end

to server
  mapreduce:acceptworkers
end

to node
  mapreduce:node.connect "127.0.0.1" 9173
end

to count-words
  let words []
  let runit true

  reset-ticks

  ;;; Tell MapReduce that a line has words, separated by spaces
  mapreduce:config.valueseparator "␣"
  ;;; Start the MapReduce computation
  let res mapreduce:mapreduce "read-file" "word-count" 0
     data-set
  ;;; Wait or the computation to finish and display the progress
  while [mapreduce:running?] [
   every 0.5 [
      print mapreduce:map-progress
      print mapreduce:reduce-progress
      ; plot 1
      tick
    ]
  ]
  tick
  print "done"

  ;;; Print the result
  show mapreduce:result res
  tick
end
```

The following code is taken from [11]. It shows how the Google-framework can be used for counting words.

Listing A.3: Word counting using the Google-MapReduce-framework (Taken from [11])

```
#include "mapreduce/mapreduce.h"

// User's map function
class WordCounter : public Mapper {
   public:
      virtual void Map(const MapInput& input) {
         const string& text = input.value();
         const int n = text.size();
         for (int i = 0; i < n; ) {
            // Skip past leading whitespace
            while ((i < n) && isspace(text[i]))
               i++;
            // Find word end
            int start = i;
            while ((i < n) && !isspace(text[i]))
               i++;
            if (start < i)
               Emit(text.substr(start,i-start),"1");
         }
      }
```

```
};

REGISTER_MAPPER(WordCounter);

// User's reduce function
class Adder : public Reducer {
   virtual void Reduce(ReduceInput* input) {
      // Iterate over all entries with the
      // same key and add the values
      int64 value = 0;
      while (!input->done()) {
         value += StringToInt(input->value());
         input->NextValue();
      }
      // Emit sum for input->key()
      Emit(IntToString(value));
   }
};

REGISTER_REDUCER(Adder);

int main(int argc, char** argv) {
   ParseCommandLineFlags(argc, argv);

   MapReduceSpecification spec;
   // Store list of input files into "spec"
   for (int i = 1; i < argc; i++) {
      MapReduceInput* input = spec.add_input();
      input->set_format("text");
      input->set_filepattern(argv[i]);
      input->set_mapper_class("WordCounter");
   }

   // Specify the output files:
   // /gfs/test/freq-00000-of-00100
   // /gfs/test/freq-00001-of-00100
   // ...
   MapReduceOutput* out = spec.output();
   out->set_filebase("/gfs/test/freq");
   out->set_num_tasks(100);
   out->set_format("text");
   out->set_reducer_class("Adder");

```

```
  // Optional: do partial sums within map
  // tasks to save network bandwidth
  out->set_combiner_class("Adder");

  // Tuning parameters: use at most 2000
  // machines and 100 MB of memory per task
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);

  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  // Done: 'result' structure contains info
  // about counters, time taken, number of
  // machines used, etc.
  return 0;
}
```

A.2 Pi

Listing A.4: Pi estimation

```
extensions [mapreduce]

to pointgenerator [key value]
 ; show (word "Running job " key " " value)
 let x 0
 let y 0
 let hitCounter 0

 random-seed read-from-string value
 repeat repeats [
   set x ((random-float 2) - 1)
   set y ((random-float 2) - 1)
   if x * x + y * y <= 1 [
     set hitCounter (hitCounter + 1)
   ]
 ]

 ; show hitCounter
 mapreduce:emit "1" hitCounter
end
```

```

to-report summer [key accum value]
  show "sumit"
  report accum + read-from-string value
end

to-report calcpi [#res]
  let hits first butfirst first mapreduce:result #res
  report (4 * hits) / (mapper * repeats)
end

to makepi
  let h 1
  let x []
  repeat mapper [set x lput h x
    set h h + 1
  ]
  let inp fput ( fput 1 fput x [] ) []

  let res mapreduce:mapreduce "pointgenerator" "summer" 0 inp

  reset-ticks
  while [mapreduce:running?] [
    wait 1
    tick
  ]

  let mypi calcpi res
  show mypi
end
```

A.3 Inverted-Index

Listing A.5: Inverted Word Index

```
extensions [mapreduce]

to mapper [file-name words]
  ;;; Loop over all words
  foreach words
  [
    mapreduce:emit ? file-name
  ]
```

```
end

to-report in-list? [value #list]
  ifelse length filter [? = value] #list > 0 [report
     true][report false]
end

to-report reducer [key #list value]
  ifelse not in-list? value #list
  [
    report fput value #list
  ]
  [
    report #list
  ]
end

to server
  mapreduce:acceptworkers
end

to node
  mapreduce:node.connect "127.0.0.1" 9173
end

to index
  reset-ticks

  ;;; Tell MapReduce that a line has words, separated by spaces
  mapreduce:config.valueseparator "␣"
  ;;; Start the MapReduce computation
  let res mapreduce:mapreduce "mapper" "reducer" [] (word
     "WordCount/" data-set)
  ;;; Wait or the computation to finish and display the progress
  while [mapreduce:running?] [
   every 1 [ tick ]
  ]
  tick
  print "done"
  show mapreduce:result res
  tick
end
```

A.4 Constructing a Graph Using the Mouse

Listing A.6: Constructing a Graph

```
breed [nodes node]
directed-link-breed [edges edge]

globals [
  ; mouse-globals
  mouse-clicked? %mouse-down? %mouse-time

  ; edge-creation-globals
  first-node]

to setup
  set %mouse-down? false
end

; Mouse-Utility
; Set mouse-clicked to true when the mouse is released
; Use it within a "forever"-command
to go-mouse
  ifelse mouse-down? [
    set %mouse-down? true
  ] [
    ifelse %mouse-down? = true [
      set mouse-clicked? true
    ] [
      set mouse-clicked? false
    ]
    set %mouse-down? false
  ]
end

; Create nodes
to node-creation
  go-mouse
  if mouse-clicked? = true [
    let nh false
    ask patch (round mouse-xcor) (round mouse-ycor) [
      ifelse any? nodes-here
        [ set nh true ]
        [ set nh false]
      ifelse nh [ask nodes-on patch (round mouse-xcor) (round
          mouse-ycor) [die]
      ] [sprout-nodes 1 [setxy mouse-xcor mouse-ycor
                         set shape "dot"
                         show who]
        ]
    ]
  ]
end

; Create Edges
to edge-creation
  go-mouse
  if mouse-clicked? = true [
    ifelse not is-agent? first-node [
      ; mark first node
      let nh false
      ask patch (round mouse-xcor) (round mouse-ycor) [
        ifelse any? nodes-here [ set nh true ] [ set nh false]
        if nh [set first-node one-of nodes-here]
      ]
    ] [
      ; mark second node
      let nh false
      ask patch (round mouse-xcor) (round mouse-ycor) [
        ifelse any? nodes-here [ set nh true ] [ set nh false]
        if nh [ ask one-of nodes-here [ let edge-exists false
            ask first-node [set edge-exists out-link-neighbor?
            myself ]
                                        ifelse edge-exists [ask
                                            in-edge-from first-node [die]
                                            ]
                                         [create-edge-from first-node]]
                                             set first-node false ]
      ]
    ]
  ]
end

to make-input
  let input []
  let kv-pair []
  let neighs []
  ask nodes [
    set kv-pair []
    set neighs []
    set kv-pair fput who kv-pair
    ask out-edge-neighbors [set neighs fput who neighs]
    set kv-pair lput neighs kv-pair
    set input fput kv-pair input
  ]
  show input
end
```

A.5 Single Source Shortest Path

Listing A.7: Single Source Shortest Path

```
breed [nodes node]
directed-link-breed [edges edge]

extensions [mapreduce]

globals [
  ; mouse-globals
  mouse-clicked? %mouse-down? %mouse-time

  ; edge-creation-globals
  first-node

  ; Shortest-Path
  dist
  infy
  notfinished?]

to setup
  set %mouse-down? false
  set infy (2 ^ 31)
  set notfinished? true
end

; Mouse-Utility
; Set mouse-clicked to true when the mouse is released
; Use it within a "forever"-command
to go-mouse
  ifelse mouse-down? [
    set %mouse-down? true
  ] [
    ifelse %mouse-down? = true [
      set mouse-clicked? true
    ] [
      set mouse-clicked? false
    ]
    set %mouse-down? false
  ]
end

; Create nodes
to node-creation
  go-mouse
  if mouse-clicked? = true [
    let nh false
    ask patch (round mouse-xcor) (round mouse-ycor) [
      ifelse any? nodes-here
        [ set nh true ]
        [ set nh false]
      ifelse nh [ask nodes-on patch (round mouse-xcor) (round
          mouse-ycor) [die]
      ] [sprout-nodes 1 [setxy mouse-xcor mouse-ycor
                         set shape "dot"
                         set label who]
        ]
    ]
  ]
end

; Create Edges
to edge-creation
  go-mouse
  if mouse-clicked? = true [
    ifelse not is-agent? first-node [
      ; mark first node
      let nh false
      ask patch (round mouse-xcor) (round mouse-ycor) [
        ifelse any? nodes-here [ set nh true ] [ set nh false]
        if nh [set first-node one-of nodes-here]
      ]
    ] [
      ; mark second node
      let nh false
      ask patch (round mouse-xcor) (round mouse-ycor) [
        ifelse any? nodes-here [ set nh true ] [ set nh false]
        if nh [ ask one-of nodes-here [ let edge-exists false
            ask first-node [set edge-exists out-link-neighbor?
            myself ]
                                      ifelse edge-exists [ask
                                          in-edge-from first-node [die]
                                          ]
                                       [create-edge-from first-node]]
                                           set first-node false ]
      ]
    ]
  ]
end

to-report make-input
  let input []
  let kv-pair []
  let neighs []
  ask nodes [
    set kv-pair []
    set neighs []
    set kv-pair fput who kv-pair
    ask out-edge-neighbors [set neighs fput who neighs]
    set kv-pair lput neighs kv-pair
    set input fput kv-pair input
  ]
  report input
end

to init-dists
  set dist []

  ask nodes [
    let d []
    set d fput infy d
    set d fput who d
    set dist fput d dist
  ]
end

; Report the distance for a Vertex
; #nid ID of the vertex
to-report get-dist [#nid]
  report first butfirst (first (filter [first ? = #nid] dist))
end

; Merge distance into the distance vector
to merge-dists [#dists]
  set notfinished? false
  foreach #dists [
    let #vertex first ?
    let entry (first (filter [first ? = #vertex] dist))
    if entry != ? [
      set notfinished? true
      let idx position entry dist
      set dist (replace-item idx dist ?)
    ]
  ]
end

; Mapper: Report a KV-Pair for all vertices reachable from
   this vertex
; Key: #vertex a vertex from the input
; Value: #neighbor one of the neighbors of #vertex
to reachable-vertices [#vertex #neighbor]
  ; show #vertex
  ; show #neighbor

  show word "mapper:␣" dist

  let #vdist get-dist read-from-string #vertex

  ; show word #vertex word " " word #neighbor word " " (#vdist
     + 1)
  mapreduce:emit #neighbor (#vdist + 1)
end

; Reducer: Report the minimum distance from the source for
   this vertex
; Key: #vertex the vertex
; Acumulator: #min minimum so far
; Value: #dist current distance to be tested for minimum
to-report min-dist [#vertex #min #dist]
  ifelse (read-from-string #dist) < #min
  [ report (read-from-string #dist) ]
  [ report #min ]
end

to load
  import-world word data-set ".csv"
end

to shortest-path
  setup

  reset-ticks

  ; Set all distances to infinity
  init-dists

  ; Node 0 is the source node. Thus, it has distance 0
  ; set distance of node 0 to 0
  let idx position (first (filter [first ? = 0] dist)) dist
  set dist (replace-item idx dist [0 0])

  while [notfinished?] [
    let #res mapreduce:mapreduce "reachable-vertices"
       "min-dist" infy make-input
    while [mapreduce:running?] [every 0.25 [tick]]
    merge-dists mapreduce:result #res
    show word "after:␣" dist
  ]
end
```

Listing A.8: Shortest Path algorithm. Taken from [25]

```
class Mapper
  method Map(nid n, node N )
    d = N.Distance
    Emit(nid n, N ) // Pass along graph structure
    for all nodeid m in N.AdjacencyList do
      Emit(nid m, d + 1) // Emit distances to reachable nodes

class Reducer
  method Reduce(nid m, [d1 , d2 , . . .])
    dmin = MAX
    M = EMPTY
    for all d in counts [d1 , d2 , . . .] do
```

```
      if IsNode(d) then
        M = d
      else if d < dmin then
        dmin = d
    M.Distance = dmin
    Emit(nid m, node M )
```

A.6 K-Means

Listing A.9: K-Means

```
extensions [mapreduce]

breed [points point]
breed [centroids centroid]

globals [centers ]

to test
  ask points [die]
  ask centroids [die]
  load-points "input/points4.txt"
  init
  kmeans
end

to load
  load-points word "input/" input
end

to load-points [#filename]
  file-open #filename
  while[not file-at-end?][
    let line file-read-line
    ; show word "line " line
    if (length line) > 0 [
      let split-pos position "␣" line
      let px (substring line 0 split-pos)
      let py substring line (split-pos + 1) (length line)
      create-points 1 [
        setxy (read-from-string px) (read-from-string py)
        set shape "dot"
        set color white
      ]
    ]
  ]
  file-close
end

to-report point-list []
  let #list []
  ; let id 0
  ask points [
   ; show xcor
   ; set id (id + 1)
   let cords list xcor ycor
   set #list (lput (list who (list cords)) #list)
  ]

  report #list
end

to-report euclidean-distance [#p1x #p1y #p2x #p2y]
  report sqrt ((#p1x - #p2x)*(#p1x - #p2x) + (#p1y -
      #p2y)*(#p1y - #p2y))
end

to-report vecsub [#p1 #p2]
  let p1x first #p1
  let p1y first bf #p1
  let p2x first #p2
  let p2y first bf #p2
  report list (p1x - (p1x - p2x) / 2) (p1y - (p1y - p2y) / 2)
end

to-report newMeans [#key #acc #value]
  let val read-from-string #value
  ifelse #acc = []
  [
    ; search for the point in the list of centers
    let cents centers
    let id (read-from-string #key)
    while [(not empty? cents) and (first first cents) !=
        id][set cents bf cents]
    ifelse not empty? cents [
      ;exract coordinates
    let centr first cents

    report vecsub (bf centr) val]
   ; there has been a mistake in the data. Keep accumulator
   [report #acc]
  ]
  [report vecsub #acc val]
end

to-report nearestCenter [#key #value]
  let nearest -1
  let mindist 99999
  foreach centers [
    let id first ?
    let x first butfirst ?
    let y first butfirst butfirst ?
    let dist 0
    set dist (euclidean-distance (first #value) (first butfirst
       #value) x y)
    if dist < mindist [
     set mindist dist
     set nearest id
    ]
  ]

  report nearest
end

to mapper [#key #value]
  ; Parse input
  let val read-from-string #value

  ; determine nearest center
  let nearest (nearestCenter (read-from-string #key) val)

  ; output
  mapreduce:emit nearest #value
end

to centroids-to-centers
  set centers []
  ask centroids [
    set centers fput (list who pxcor pycor) centers
  ]
end

to init
  ask centroids [die]
  ; initially guess centroids
  ask n-of k points [hatch-centroids 1[
       set shape "dot"
       set color 15 ]]
end

to kmeans
  repeat iterations [
    ; create the list of centers
    centroids-to-centers

    let res mapreduce point-list

    ; adapt centroids
    foreach res [
      let id first ?
      let x (first last ?)
      let y (last last ?)
      ask centroid id [ setxy x y ]
    ]
    show "recentering␣done"
    color-partitions
  ]
end

to color-partitions
  let cl []
  ask centroids [set cl fput who cl]
  set cl sort cl

  foreach point-list [
    ; show (first ?)
    ; show (first last ?)
    let nc nearestCenter (first ?) (first last ?)
    ; show "jjj"
    let cid nc
    let tcl cl
    let c 25
    while [(first tcl) != cid] [set c (c + 10) set tcl bf tcl]
    ask point (first ?) [set color c]
  ]
end

to-report mapreduce [#input]
  reset-ticks

  let res mapreduce:mapreduce "mapper" "newMeans" [] #input

  while [mapreduce:running?] [
   every 0.5 [
       ; print mapreduce:map-progress
       ; print mapreduce:reduce-progress
       ; plot 1
       tick
     ]
  ]
  tick

  print "done"
  report mapreduce:result res
end
```

A.7 PageRank

Listing A.10: PageRank

```
extensions [mapreduce table]

globals [outlinks ranks]

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; Mapper for the pagerank computation. Inverting the graph
;;;; Gets a line of the input file
;;;; #key filename
;;;; #values definition of a site,
;;;; format : site rank outlink_1 ... outlink_n
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to distributerank [#key #values]
  ;;; read site
  let site read-from-string first #values
  ;;; do not read rank since it changes between iterations.
  ;;; Use ranks table instead
  let rank table:get ranks site

  ;;; now loop over all outlinks and
  foreach bf bf #values [
    mapreduce:emit ? rank
  ]
end

to-report summer[#key #accum #value]
  report (#accum + read-from-string #value)
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; Determine the number of outlinks for a given page
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to map-outlinks [#key #values]
  let site first #values

  ;;; now loop over all outlinks and
  foreach bf bf #values [
    mapreduce:emit site 1
  ]
end

to extractrank [#key #values]
  let site first #values
  let rank first bf #values

  mapreduce:emit site rank
end

to-report identity [#key #accum #value]
  report #value
end

to pre-process [#input]
  mapreduce:config.valueseparator "␣"
  let res1 mapreduce:mapreduce "map-outlinks" "summer" 0 #input
  while [mapreduce:running?] [wait 1 tick]
  tick
  let res2 mapreduce:mapreduce "extractrank" "identity" 0 #input
  while [mapreduce:running?] [wait 1 tick]
  tick
  set outlinks table:from-list mapreduce:result res1
  set ranks table:from-list mapreduce:result res2
  show "Pre-Processing␣done"
  show ranks
end

to-report #outlinks [#page]
  report table:get outlinks #page
end

to iteration [#input]
  mapreduce:config.valueseparator "␣"
  let res mapreduce:mapreduce "distributerank" "summer" 0 #input
  while [mapreduce:running?] [wait 1 tick]
  tick
  let ranksums mapreduce:result res
  show ranksums
  foreach ranksums [
    let page first ?
    let rank first bf ?
    set rank (rank / (#outlinks page ))
    table:put ranks page rank
  ]
  show ranks
end

to pagerank
  reset-ticks
  pre-process input
  repeat iterations [
    iteration input ]
end
```

Bibliography

[1] E Ackermann. Constructing knowledge and transforming the world. *A learning zone of one's own: Sharing representations and flow in collaborative learning environments*, 1:15–37, 2004.

[2] Edith Ackermann. Piaget's constructivism, papert's constructionism: What's the difference. *Future of learning group publication*, 5(3):438, 2001.

[3] Meurig Beynon and Chris Roe. Computer support for constructionism in context. In *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*, pages 216–220. IEEE, 2004.

[4] George Bodner, Michael Klobuchar, and David Geelan. The many forms of constructivism. *Journal of Chemical Education*, 78(8):1107, 2001.

[5] P Boytchev. Logo tree project, rev 1.97, 2013.

[6] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[7] Wei Cao and Dennis Shasha. Appsleuth: a tool for database tuning at the application level. In Giovanna Guerrini and Norman W. Paton, editors, *EDBT*, pages 589–600. ACM, 2013.

[8] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[9] Shimin Chen and Steven W. Schlosser. Map-reduce meets wider varieties of applications. Technical report, 2008.

[10] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[12] David DeWitt. Mapreduce: A major step backwards. The Database Column, `http://www.cs.washington.edu/homes/billhowe/mapreduce_a_major_step_backwards.html`, 01 2008. last-seen: 2012-07-19.

[13] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[14] Peter E Doolittle and David Hicks. Constructivism as a theoretical foundation for the use of technology in social studies. *Theory & Research in Social Education*, 31(1):72–104, 2003.

[15] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[16] Christos Faloutsos, Kevin S McCurley, and Andrew Tomkins. Connection subgraphs in social networks. In *SIAM International Conference on Data Mining, Workshop on Link Analysis, Counterterrorism and Security*, 2004.

[17] The Apache Software Foundation. Hadoop. `http://hadoop.apache.org`.

[18] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, October 2003.

[19] Project Gutenberg. Project gutenberg. `http://www.gutenberg.org`.

[20] Garry Hoban, Wendy Nielsen, and Charles Carceller. Articulating constructionism: Learning science though designing and making "slowmations" (student- generated animations). In *Proceedings of ASCILITE - Australian Society for Computers in Learning in Tertiary Education Annual Conference 2010*, pages 433–443. ascilite, 2010.

[21] Nathan Holbert, Lauren Penney, and Uri Wilensky. Bringing constructionism to action game-play. *Proceedings of Constructionism*, 2010.

[22] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[23] B. Inhelder and J. Piaget. *The Growth of Logical Thinking: From Childhood to Adolescence. An Essay on the Construction of Formal Operational Structures.* Developmental psychology. Routledge & Kegan Paul, 1958.

[24] Beaumie Kim. Social constructivism. *Emerging perspectives on learning, teaching, and technology*, pages 1–8, 2001.

[25] Jimmy Lin and Chris Dyer. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2010.

[26] Christian Schauer Markus Leitner, Mario Ruthmair. Lecture notes for "algorithmen auf graphen", 10 2011.

[27] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[28] T.R.G. Green P. Mendelsohn and P. Brna. *Psychology of Programming*, chapter Programming Languages in Education: The Search for an Easy Start, pages 175–200. European Association of Cognitive Ergonomics and Academic Press.

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[30] Seymour Papert. Constructionism vs. instructionism. `http://www.papert.org/articles/const_inst/const_inst1.html` last-seen 2013-07-25.

[31] Seymour Papert and Idit Harel. Situating constructionism. In Seymour Papert and Idit Harel, editors, *Constructionism*, chapter 1. Ablex Publishing Corporation, Norwood, NJ.

[32] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[33] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

[34] Jörg Schad. Flying yellow elephant: Predictable and efficient mapreduce in the cloud, 2010.

[35] Robert Sedgewick. *Algorithmen*. Addison-Wesley, 1991.

[36] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[37] Karen Swan. A constructivist model for thinking about learning online.

[38] W. Stroup U. Wilensky. Hubnet. `http://ccl.northwestern.edu/netlogo/hubnet.html`, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.

[39] Ernst Von Glasersfeld. An introduction to radical constructivism. *The invented reality*, pages 17–40, 1984.

[40] Barry J. Wadsworth. *Piaget's theory of cognitive and affective development : foundations of constructivism*. Longman, 1996.

[41] Uri Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo/`, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.