AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY
Faculty of Information and Natural Sciences
Degree Program of Computer Science and Engineering

Visualization of Data Processing in Distributed Computing Environments

Bachelor's Thesis

Jyri Tuulos

jyri.tuulos@tkk.fi

Department of Media Technology
Espoo, Finland 2010

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY
Faculty of Information and Natural Sciences
Degree Program of Computer Science and Engineering

ABSTRACT OF BACHELOR'S THESIS

Author: Jyri Tuulos

Title of thesis:
Visualization of Data Processing in Distributed Computing Environments

Date: December 2, 2010 **Pages:** 18
Professorship: WWW Technologies **Code:** IL3012
Supervisor: Prof. Ilkka Niemelä
Instructor: Denis Shestakov, PhD

Distributed data processing has become indispensable and ubiquitous with the substantial growth of public data. An increasing number of users need intuitive tools to help understand and analyze distributed computing systems. This thesis covers the use of interactive visualizations as a method of easing the learning curve of distributed data processing. Previous work on the principles of algorithm visualization and MapReduce profiling tools is presented in a literature review. In addition, findings from these related fields are combined to create a proof-of-concept visualization prototype.

The JavaScript-based prototype displays a step-by-step visualization of a MapReduce job using log files of a computing framework. The prototype illustrates the execution of a distributed data processing task using an animated graph. The animation focuses on representing each line in the log file using intuitive visual steps.

The visualization prototype presents the potential of improving MapReduce profiling tools with interactive visualizations. However, this proof-of-concept implementation does not address some essential issues, such as error handling and scalability. A list of requirements is offered to target these issues and facilitate further development of distributed computing visualizations. This thesis proposes that the main considerations for visualizing distributed data processing are usability, aesthetics and organization of information.

Keywords: visualization, distributed computing, MapReduce, log, algorithm visualization
Language: English

Abbreviations

MR	MapReduce
DCS	Distributed computing system
AV	Algorithm visualization

Contents

Chapter 1

Introduction

The amount of accessible data increases every year thanks to various public sources, such as web services and scientific measurements. With the substantial growth of publicly available data, methods for extracting useful information from large amounts of raw data have become indispensable. These methods utilize distributed computing to enable processing vast amounts of data in a reasonable time. An increasing number of non-experts have to learn how to utilize a distributed computing system (DCS), as distributed data processing becomes a common solution for information extraction tasks.

Various tools, frameworks and services have been developed to facilitate distributed data processing. At present, anyone with a basic knowledge of computers and programming can install a free distributed data processing framework on a set of personal computers and execute a ready-made example program. Furthermore, the same program can be run in a cluster of machines rented from a cloud computing service. All this can be done without actually understanding the technical details of distributed computing. However, the learning curve suddenly rises when faced with the task of developing advanced programs and optimizing their processing time.

Data processing in distributed environments is challenging even for experienced computer scientists (Topol et al., 1995). In order to understand distributed programs, one must comprehend a large amount of variables, non-deterministic elements and their relations. Distributed computing frameworks can handle some basic aspects, such as concurrent data handling and scheduling processing tasks, but users still need to have an understanding of what happens in the system in order to solve problematic situations. Solving all issues automatically is not feasible, since nodes in a DCS can not distinguish between different types of problems due to the lack of information. For example, if a node does not respond to remote

calls, the cause can be a crashed process, a connection problem or a hardware failure (Vantuyl, 2010). Therefore, processing data effectively in a DCS always requires some user interaction and programming. However, in order to be helpful to users, the complex information gathered from the distributed system has to be represented with multiple levels of abstraction, such as tables and visualizations.

Distributed data processing has characteristics that cause visualizations to be especially effective in helping users understand distributed programs: First, processing and analyzing data is by nature an iterative process, since there are always variables that can be changed to optimize the execution and results. The process of iterating over these variables can be aided by an easily interpretable representation of how they affect the program execution. Another characteristic is that problems appearing in complex DCSs may be difficult or impossible to reproduce in smaller clusters (Konwinski and Zaharia, 2008). Thus, the final optimizations have to be performed in a full-size system with a complete data set. A method for intuitive visual feedback helps to improve the program and reach satisfatory results in fewer time-consuming iterations.

This thesis covers the use of visualizations as a method of easing the learning curve for developing and optimizing distributed programs. The main goal of this thesis is to formalize a set of basic requirements for an interactive visualization of data processing in a distributed system. These requirements are based on earlier work on related areas, such as MapReduce profiling methods and algorithm visualization. Also, a proof-of-concept prototype is developed as a starting point for future work. In contrast to professional DCS profiling tools (Massie et al., 2004; Newman et al., 2003), which are comprehensive but cumbersome, the prototype presented in this thesis is designed to be light-weight, approachable and intuitive, resulting in an interface that can be used for both learning how the program works and evaluating the processing performance.

The structure of this thesis is as follows: Chapter 2 is a literature review of the different methods and guidelines used in algorithm visualization and MapReduce profiling. Chapter 3 presents the background, functionality and evaluation of a visualization prototype. Finally, Chapter 4 proposes a list of requirements for visualizing DCSs and presents future directions for developing approachable visualizations for data processing in distributed environments.

Chapter 2

Related Work

This thesis discusses the use of interactive visualizations as a method of understanding and optimizing distributed data processing programs. Currently, no widely used implementations exist to address this specific purpose. Moreover, most of the previous studies focus on non-visual profiling tools, that is, programs which gather execution data from distributed programs and analyze it automatically to detect issues. Little research has been conducted to determine the usability and applicability of visual and intuitive approaches for optimizing DCS performance.

This chapter presents two fields of study that have been extensively researched separately: Algorithm visualization (Section 2.1), a sub-field of software visualization, aims to visualize algorithms used in program implementations and data processing. Another field, MapReduce performance profiling (Section 2.2) covers the methods of gathering and analyzing execution data from the MapReduce distributed computing framework. The data gathered by profiling tools can be visualized according to the guidelines of algorithm visualization. Thus, previous findings from these fields can be combined to visualize distributed data processing using novel approaches.

2.1 Algorithm Visualization

The goal of algorithm visualization (AV) is to provide high-level graphical representations of algorithms for learning and education purposes (Lazaridis et al., 2010). Since the first implementations in the 1970s, AVs have evolved from simple animations to interactive programs that provide a dynamic multi-perspective view of the algorithm (Hundhausen, 2002). Shaffer et al. (2010) define that a

"good" AV gives an animated representation of the states of an algorithm and the transitions between those states. They also state that good AVs "illustrate data structures in natural, abstract ways instead of focusing on memory addresses and function calls" (Shaffer et al., 2010).

Algorithm visualization has several benefits in understanding computational software. First, providing a clear abstraction of an algorithm helps non-expert users to understand the functionality of a program. The visualization should only include the information necessary to represent the algorithm, while abstracting theoretical and technical details. Second, interactive visualizations keep the users' attention while enabling them to examine the algorithm at their own pace (Shaffer et al., 2010). Third, users can input their own data to see how the program would solve their problem. Fourth, different states of the algorithm and their relationships can be visualized using animation and smooth transitions, illustrating the internal changes in the program during its execution. Finally, important structures and activities are highlighted with colors and various visual effects, helping the user distinguish the main parts of the algorithm. (Lazaridis et al., 2010).

2.1.1 Evaluating Algorithm Visualizations

The current algorithm visualization research mainly concentrates on using AVs in computer science education (Shaffer et al., 2010). The most effective results are achieved when using a quality AV in the right context. The factors influencing the quality and effectivity of AVs have been researched extensively, revealing multiple considerations for AV developers:

- The main steps of the algorithm should be clearly represented (Gloor, 1997).
- The initial view should include as few elements as possible to make the visualization approachable (Lazaridis et al., 2010).
- To make sure that the user can follow the steps in the AV, they should contain at most one interesting event and change at most seven visual elements (Faltin, 2001).
- The visualization should give the user time to think about every step of the algorithm, preferably enabling the users to explore the visualization at their own pace (Gloor, 1997).
- AVs work best as a catalyst for learning the algorithm and ultimately completing a given task (e.g. educational excercise, understanding a program) (Hundhausen, 2002).

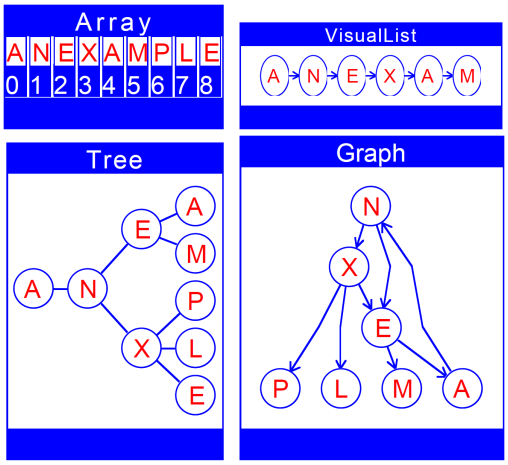


Figure 2.1 – Fundamental data types, such as arrays, lists, trees, and graphs are important reusable abstractions regularly used in computer science. (Korhonen et al., 2004)

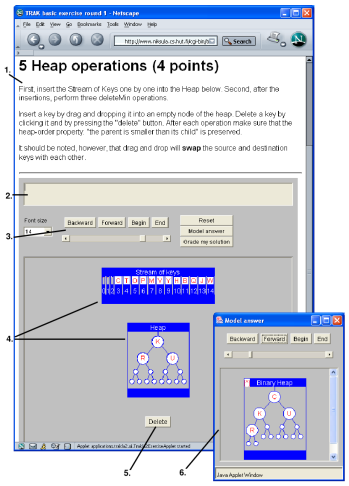


Figure 2.2 – TRAKLA2 applet page and the model solution window. Both list and graph visualizations are used to represent the data structures of the algorithm. (Korhonen et al., 2004)

Shaffer et al. (2010) provide a list of the most prominent AV systems currently available. The majority of these systems focus on showcasing a collection of algorithms for educational purposes. Some implementations include methods for developing custom AVs from algorithm scripts or programming code. Even though most AVs are developed as standalone software using Sun's Java programming language, Shaffer et al. (2010) argue that visualizations directly available on the web reach a greater number of potential users. They conclude that there is a need for more high quality AVs, especially for more advanced subjects.

2.1.2 Algorithm Visualization Methods

There are countless methods for algorithm visualization, since AVs cover algorithms from various fields (Shaffer et al., 2010). Most algorithms and data structures have a natural representation which can be used as the basis for a visualization (Figure 2.1); linear structures (i.e. lists), graphs, trees and grids cover a large part of the existing AVs. In addition, AVs can combine several visualizations to a single interactive presentation to represent all the different parts of an algorithm. For example, Figure 2.2 shows the basic problem solution view in the algorithm education program *TRAKLA2*, including the usage instructions, playback controls and an interactive visualization (Malmi et al., 2004).

Visualizing advanced algorithms and structures, such as nonplanar graphs, can introduce challenges to AV design (Dickerson et al., 2002). In particular, multiple challenges have to be considered when visualizing distributed algorithms: DCSs are complex and non-deterministic by nature, which is mirrored on the behaviour of distributed algorithms; the same inputs can give varying outputs on each execution. In addition, the AV has to match the scalability of the represented distributed software, causing issues in resource requirements and the amount of visual elements displayed. Other challenges include concurrency handling (i.e. event synchronization) and fault recovery (e.g. network or hardware failures) (Topol et al., 1995). However, there are benefits in creating an AV for a specific algorithm, such as MapReduce, compared to visualizing any arbitrary distributed process. Following a tightly limited paradigm in a specific domain eliminates the need for generalization in every part of the visualization.

2.2 MapReduce Performance Profiling

Topol et al. (1995) argue that the rarity of distributed computing visualizations is mainly caused by the difficulty of gathering the raw data needed for constructing an accurate representation. In the past decade, efficient methods have been developed for profiling DCSs, that is, analyzing the execution of distributed software. Currently, two major approaches exist for profiling distributed programs: Execution data can either be gathered with a runtime tracing tool (Konwinski and Zaharia, 2008) or extracted from the log files generated during the program execution (Tan et al., 2010). Since this data is readily available, it can be visualized and utilized for learning and performance analysis.

Profiling is an important aspect of administrating distributed computing systems. Systems consisting of multiple computers may exhibit performance issues and hardware failures, which increase the processing time of any executed task. These issues can often be detected by examining the output of a profiler, that is, a tool analyzing the behaviour of a program using information gathered from its execution. Profilers are built separately for each program and framework to capture software-specific information. This section presents a brief introduction to MapReduce, a distributed computing framework, and current methods for profiling it.

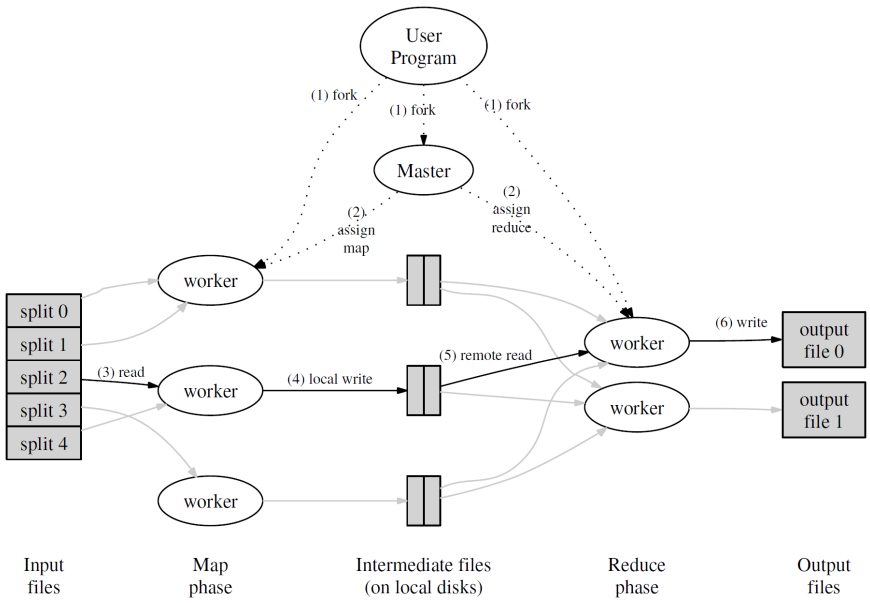


Figure 2.3 – The MapReduce execution overview in steps 1–6. In the figure, workers represent computing nodes, where the user implementation is forked (installed). (Dean and Ghemawat, 2004)

2.2.1 Introduction to MapReduce

MapReduce (MR) (Dean and Ghemawat, 2004) is a framework developed at Google to facilitate data processing in distributed systems. The main goal in the development of MapReduce was to address parallellization, data distribution and error handling in a single library. Since its publication in 2004, MapReduce has become widely used especially for processing data gathered from the web (Lawton, 2010). Dean and Ghemawat (2004) present several features which have contributed to its wide-spread adoption:

- Simple data processing jobs can be implemented and executed quickly with MR, enabling rapid development and short iterations in solving a problem.
- The MR framework runs on common hardware, including clusters of personal computers with varying specifications and resources.
- MR is highly scalable and is developed to recover from hardware failures.
- Experience about distributed computing is not required from the user, since the framework automatically handles tasks such as data partitioning and task scheduling.

Processing data with MapReduce is straightforward, since the user only has to implement two functions to create a distributed program. Figure 2.3 shows the main steps and operations (numbered 1–6) of the MapReduce job execution: In the beginning of the job, the user implementation is forked to the master and worker nodes (1). The master node ensures that the data is distributed correctly and the functions are called on the right machines (2). The first function, *Map*, reads the raw input data (3) and stores it into intermediate key/value pairs (4). These pairs are grouped by key in the shuffle phase and passed to the second function (5), *Reduce*, which produces the final output by merging the intermediate values (6). Numerous problems can be solved using this pattern since the data processing functions are written by the user to output the desired results. (Dean and Ghemawat, 2004).

The simple interface of MapReduce enables inexperienced users to utilize and experiment with the processing power of distributed environments. However, users may have difficulties understanding how the data and function calls are distributed between the computing nodes. That information about the system states is recorded in plain text log files in a counterintuitive and fragmented form. The log files have to be processed and visualized with a separate profiling tool to extract useful conclusions about the executed process.

2.2.2 MapReduce Profiling Methods

Various methods have been used for profiling the performance of MapReduce jobs. Huang et al. (2010) divide the methods for tracing MapReduce into static and dynamic approaches: Static methods, such as *Mochi* (Tan et al., 2009) and *Artemis* (Creţu-Ciocârlie et al., 2008) analyze and visualize the log files collected after the computing process. In contrast, dynamic methods use a network tracing tool to get a real-time representation of the activity in the DCS (Huang et al., 2010; Konwinski and Zaharia, 2008).

Tan et al. (2009, 2010) present a log-based debugging tool, Mochi, for analyzing causal flows in MapReduce jobs. Mochi parses data and control events from the log files produced by the MR nodes and links the events to form control and data flows through the system. These flows represent how a single data partition is processed and transmitted in the DCS. Mochi uncovers MapReduce-related problems in the system by analyzing the total resources and computing time of each flow. Such problems include uneven data partitioning between nodes and unoptimal task scheduling. Mochi provides visualizations for analyzing the statistical distributions of the flow characteristics, detecting the outliers, and examining how the data is passed between nodes.

Real-time profiling methods provide a way to identify potential problems in the MapReduce job during its execution. Konwinski and Zaharia (2008) have developed a MR profiler using the *X-Trace* tracing framework (Fonseca et al., 2007), which is able to capture causal relations between events, thus forming similar execution paths (flows) as the log-based approaches. Their trace analysis user interface presents multiple metrics gathered from the MR tasks, including lists of slowest machines, graphs of system performance, and contingency tables of the events and their relations.

The need for real-time monitoring is the main reason for choosing between static (log-based) and dynamic (trace-based) MR profiling methods. Monitoring the MR job in real time enables fault detection before the execution is completed. This is especially useful when the processing time for a single job is measured in hours or days. However, runtime methods require added instrumentation for capturing and processing the changes in the system and may introduce computing overhead to the DCS. Konwinski and Zaharia (2008) also report several issues in the development of a dynamic profiler: First, the tracing architecture has to be highly scalable to process the reports transmitted simultaneously from all nodes in the DCS. Second, events have to be reported reliably in order to prevent problems in constructing the event graph. Moreover, the MR framework implementation can cause additional issues, such as event ID collisions.

Tan et al. (2010) present multiple benefits of MapReduce-specific log-based tracing over dynamic methods: First, the data needed for the profiling is automatically collected by the MR framework itself in the log files of each node. Second, no training data is needed to extract the event information from the log files since the MR event types are limited and can be hardcoded as *a priori* knowledge. Finally, the profiling results can be analyzed after the MR job has been completed, eliminating the issues of processing numerous events concurrently in real-time.

MapReduce profiling tools are under active research and there are subjects that have not yet been covered in detail. For example, more work is needed to evaluate the usability and efficiency of the profiler user interfaces. Interactivity and better visualizations could be key approaches for designing better interfaces for error detection and performance optimization (Cawthon and Moere, 2007).

2.2.3 Visualizing MapReduce Profiling Data

Representing MapReduce jobs effectively and intuitively is a non-trivial task due to the multiple metrics and relationships involved in a distributed system. Most earlier applications provide a user interface for analyzing the profiling results using traditional static visualizations such as histograms, tables, and box plots with no added interactivity (Tan et al., 2009, 2010; Konwinski and Zaharia, 2008). These visualizations can be effective in analyzing problems related to a single metric or characteristic of the system. However, providing an overview of the causal relations in a MR job would make the detailed results more understandable for non-expert users (Konwinski et al., 2007).

Chapter 3 demonstrates an interactive visualization prototype that represents the causal event structure of a MR job, while having the possibility to display detailed monitoring information in the event graph. The prototype is strongly influenced by the algorithm visualization guidelines (Section 2.1.1) and utilizes state animation to illustrate the time dimension of MR control and data flows.

Chapter 3

Visualization Prototype

This chapter presents a proof-of-concept prototype that was developed to demonstrate the feasibility of visualizing distributed data processing. Implementing the prototype also helped to determine requirements for the development of DCS visualizations. The visualization prototype displays a step-by-step animation of a single MapReduce job beside the original log messages from the framework. The prototype was implemented as a web page[1] that runs a JavaScript program developed using the *jQuery* library[2].

The prototype combines the advantages of MapReduce profiling tools with the methods of algorithm visualization by providing a clear visual representation of the information extracted from log files. The prototype implementation incorporates multiple approaches from the related work presented in Chapter 2: First, the layout of the visualization is based on the figure depicting the MapReduce execution by Dean and Ghemawat (2004) (Figures 2.3 and 3.1). Second, the prototype is constructed using the log files generated by the MapReduce framework, similarly to the log-based profiling tools presented in Section 2.2.2. By matching the lines in the log file to corresponding steps in the algorithm animation, the visualization helps to interpret the meaning of each event in the program execution.

[1] `http://jyrituulos.com/mrVis/mrVis.html`
[2] `http://jquery.com/`

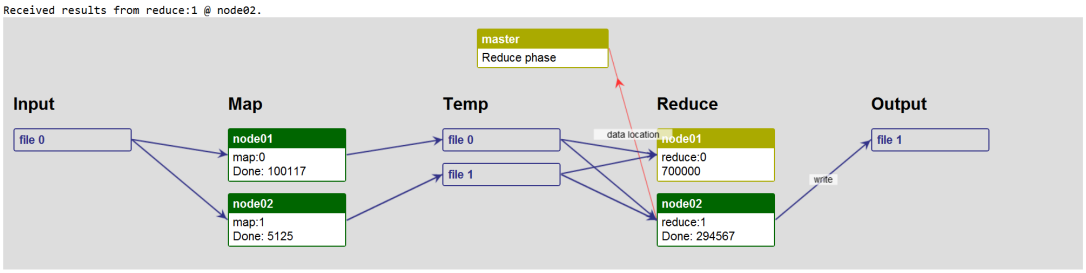
3.1 Prototype Functionality

The visualization is based on a log file generated by a MapReduce job running on two computing nodes. The log file was produced by running a program that counts the word frequencies in a long text file using the *Disco* framework[3]. The word counting task was selected for its simplicity and short processing time. However, the prototype can be used to visualize any MR job, since the log file always contains the same events regardless of the input data and MapReduce function implementations. Log events representing a third node were added manually later in the development to better illustrate the relationships between multiple nodes.

The visualization is initialized when the prototype web page is loaded. The example log file is inputted as raw text and parsed to JavaScript objects that represent events in the program execution. Typical MapReduce log lines contain a timestamp, a node name and a message describing the event that has happened in the node. In the initialization stage of the prototype, these messages are compared to known keywords appearing in the Disco framework log files. Based on this comparison each line is assigned an event type when it is converted to an event object; for example, these event types include `job-start`, `phase-map-start` and `task-assign`. The event types are framework independent, since they are based on the generic definition of the MapReduce algorithm. Thus, the visualization can be modified to support other MR frameworks, such as Hadoop[4], by writing another set of keywords for the event types.

After the log file is loaded and mapped to an array of event objects, the prototype starts the step-by-step animation of the MapReduce job. The user can advance the animation to the next step by pressing a key. In each step, the prototype reads an event from the event array and renders the corresponding step based on the event type and other metadata parsed from the log message, such as task id and target node. In addition, the original log message is shown above the animation and the row of the current event is highlighted in the table of events below the visualization. As the animation progresses, it draws a complete graph illustrating the flow of data in the computation from input to output according to the MapReduce algorithm.

[3] `http://discoproject.org/`
[4] `http://hadoop.apache.org/`



All Events

id	timestamp	node	task	type	targetNode	value	message
0	2010/10/13 01:45:46	master		job-initialize			New job initialized!
1	2010/10/13 01:45:46	master		job-start			Starting job
2	2010/10/13 01:45:46	master		phase-map-start			Map phase
3	2010/10/13 01:45:46	master	map:0	task-assign	node01		map:0 assigned to node01
4	2010/10/13 01:45:46	master	map:1	task-assign	node02		map:1 assigned to node02
5	2010/10/13 01:45:47	node01	map:0	task-receive			[map:0] Received a new map task!
6	2010/10/13 01:45:47	node02	map:1	task-receive			[map:1] Received a new map task!

Figure 3.1 – Visualization prototype page, including the animation view on the top and a table of events at the bottom. The animation is in the Reduce phase; `node02` has finished its Reduce task while `node01` is still computing.

3.2 Visual Design of the Prototype

The visualization was designed according to the guidelines of algorithm visualization presented in Section 2.1.1. The initial view includes only the master node and instructions on how to advance to the next step. This gives the user the possibility to expand their understanding of the MapReduce algorithm gradually, without initial confusion from visual clutter. The prototype enables users to examine the changing states at their own pace by splitting the execution to user-controlled steps. The steps are further grouped to five phases representing the main stages of the algorithm. Additionally, the animation is tightly integrated to the log messages from the framework. The log files are the main source of information for debugging a DCS; therefore, connecting the log events with a visual representation gives the users a motivation to explore and use the visualization to solve their problem.

The graph visualization consists of four types of elements: Computing nodes (white with colored labels) and files in the file system (grey) are drawn as the vertices of the graph. In addition, there are two kinds of arrows connecting the vertices: blue arrows represent data transfers, and red arrows are temporarily shown to represent control messages to and from the master node. These connectors are drawn using *jsPlumb*[5], an external jQuery plugin. The computing nodes are colored according to their state: grey nodes are inactive, orange nodes are running a task, and green nodes have completed their task. In a full-featured visualization, red nodes could indicate nodes in an error state.

[5] `http://code.google.com/p/jsplumb/`

The elements in the visualization are laid out according to the main phases of the MapReduce algorithm. The file and worker nodes are shown in five columns: input files, map tasks, temporary files to be shuffled, reduce tasks and output files. The master node is located above these columns, since it controls all nodes through the whole job. This layout represents both the order of the computation phases and the data flow through the system. In addition, the currently active phase in the animation is highlighted in order to direct the attention of the user to the main stages of the execution.

Each step in the visualization makes only a small number of visual changes to prevent overloading the short-term memory of the user; the steps change the state and connections of one node at a time. In addition, the textual information in the nodes is kept to a minimum to prevent visual clutter. Additional information could be graphically represented; for example, the amount of data transferred between nodes could be visualized as the thickness of the connecting arrow. Furthermore, detailed information about a single node or file could be linked to a separate view, which could be accessed by clicking the element.

3.3 Prototype Evaluation

As with all visualization tasks, the primary goal for implementing the prototype was to display the most relevant information in the original data. A step-by-step animation was chosen as the basis for the design, since MapReduce event logs are sequential by nature. The prototype focuses on displaying the main steps of the MR algorithm, thereby persuading non-expert users to understand the program execution through the underlying framework. While the prototype lacks essential information, such as amount of data transferred and node resource usage, further additions to the visualization should be thoroughly evaluated. The effectiveness of the visualization can be easily impaired by inconsiderately displaying all available profiling information.

The scope of this thesis does not include a complete usability analysis of the prototype, thus leaving its effectiveness open to interpretation. Nevertheless, both interactivity and aesthetics have been proven to better engage users and improve the usability of a visualization (Lazaridis et al., 2010; Cawthon and Moere, 2007). The prototype demonstrates an interactive and natural approach for visualizing the structure of a distributed computing program, in comparison with the traditional static visualizations used in existing tools (Massie et al., 2004; Tan et al., 2010). In order to develop more approachable visualizations for non-expert users, the prototype should be further iterated and evaluated with proper usability testing.

The prototype was implemented from the start as a modular system, meaning that the implementation consists of multiple parts that function independently. The parsing of the raw log data, the visual representation of the events and the event triggering functionality are all separate modules that can be modified separately. This enables adding support for additional event types, visualization methods and MR frameworks.

A number of external JavaScript libraries were used to reduce development time and provide the required functionality for the prototype. Several JavaScript-based graph visualization toolkits, including *Protovis*[6] and *JavaScript InfoVis Toolkit*[7], were considered for rendering the nodes and their connections. However, none of them had sufficient support for visualizing generic directed graphs. The selected plugin, jsPlumb, is not designed for rendering complete graphs, but drawing directed connectors between two elements on a web page. Thus, the prototype is only able to render the graph one step at a time and does not store a complete graph structure in memory.

Due to the simplified nature of the prototype, several issues were ignored on purpose during its development: First, the prototype was only tested with a basic log file representing a job running on three machines. Since this file contained no erroneous events nor a significant amount of nodes, the issues of error visualization and scalability should be addressed during further development. Second, the implemented visualization only allows advancing in the animation one event at a time. In order to skip to an arbitrary step in the animation, the state of each step should be stored while initializing the visualization.

The purpose of the visualization prototype was to demonstrate the feasibility of visualizing data processing in distributed environments. The prototype was implemented using the default event log of the Disco framework, a basic knowledge of the MapReduce algorithm and earlier experience of web development techniques. Thus, the main challenge was not the implementation itself, but designing the visualization to be helpful for finding the necessary information. In the case of MapReduce profiling, an ideal visualization would provide a clear view to the issues and optimization aspects of the program execution. Although the prototype presented in this thesis does not contain the information required from a complete profiling tool, it serves as an educational representation of the MapReduce algorithm and a basis for future development of DCS visualizations.

[6] `http://vis.stanford.edu/protovis/`

[7] `http://thejit.org/`

Chapter 4

Conclusions

This thesis presented a literature review of the principles of algorithm visualization and MapReduce profiling methods. In addition, Chapter 3 introduced a proof-of-concept visualization prototype, which is based on the concepts and guidelines presented in the literature review. The prototype shows that log-based profiling techniques can be augmented using algorithm visualization to represent profiling data from the MapReduce framework effectively. The findings from the prototype implementation are further formalized in Section 4.1 as requirements for developing a DCS visualization tool.

4.1 Requirements for DCS Visualization

As Topol et al. (1995) stated in their research, visualizing distributed programs can help in understanding, verifying and optimizing them. These goals coincide with the purpose of current DCS profiling tools (Section 2.2). Thus, more research should be performed on the possibility of improving DCS profiling tools with advanced visualization methods. This section covers considerations for developing DCS visualizations and modifying current profiling tools in order to better support them.

Visualization-based profiling tools require sufficient execution data from the program to serve as an effective profiling method. Ideally, all data gathered from the system should be included in the log files generated by the distributed computing framework. However, current implementations do not output all possible execution data by default, since their log files are not designed to be visualized, but to be read in plain text. For instance, the default log files of the Disco framework do not include the amount of data transferred in the read and write operations.

By providing all possible data, the framework would leave the task of presenting the relevant information for the visualization. Furthermore, users should not be required to provide the necessary data manually; nevertheless, programmers could output custom log messages and metrics from their program code, which would be displayed in the visualization.

In addition to the acquisition of raw data, there are multiple aspects that have to be considered when designing DCS visualizations:

- Usability has to be one of the main concerns in the development of visualization-based tools in order to make them more efficient to use and prevent user frustration (Cawthon and Moere, 2007).
- User-centered design is essential in choosing suitable visualization methods and designing their aesthetics (Cawthon and Moere, 2007).
- Visualizations running in a web browser are readily accessible and can be integrated to the web-based profiling tools of existing distributed computing frameworks (Tan, 2009; Konwinski and Zaharia, 2008).
- The visualization requires a versatile web-based library for drawing directed graphs to represent the network graph, i.e., the nodes and their connections.
- Multiple levels of detail are required to keep the structure of the information clear; the visualization should include a clear main overview of the execution, which can provide links to detailed information about single nodes or events.
- The visualization should be scalable, that is, handle a large number of computing nodes and events. The most challenging part in respect to scalability is drawing a usable network graph.

4.2 Future Research

The visualization prototype demonstrated in this thesis proves the viability of intuitively representing MapReduce data processing. This prototype could be evolved to a more focused and effective tool with the help of further research. Several advanced features were considered in the design of the prototype, but were left out in the implementation: First, the visualization could show more information about the distributed computation, especially the amount of data transferred between nodes. A separate view could show node-specific monitoring and resource usage information, which could be accessed by clicking the node on

the main visualization. Second, the visualization could include the option to play through the whole animation automatically according to the timestamps of each event. This method would represent the duration of the tasks and phases more intuitively than a step-by-step animation.

Another noteworthy development path would be to enable the real-time visualization of events, in contrast to log-based analysis after the execution. Real-time visualizations could be built using the methods of trace-based profiling tools (Section 2.2.2), since both require concurrent gathering and analysis of DCS events. The visualization could be optionally turned off by the user before starting the computation job in order to reduce profiling overhead.

The future development of the visualization prototype is facilitated by its modular design. Adding support for additional frameworks, such as Hadoop, would be relatively straightforward and could increase the adoption and utilization rate of the visualization. A more developed version could even be integrated in the framework itself, as Topol et al. (1995) has proposed. The scalability of the visualization is another essential issue that must be addressed in a complete profiling solution. In conclusion, there are multiple paths of development that could be evaluated and explored to develop a tool for better understanding data processing in distributed environments.

Bibliography

Nick Cawthon and Andrew Vande Moere. The Effect of Aesthetic on the Usability of Data Visualization. *11th International Conference Information Visualization (IV '07)*, pages 637–648, 2007.

Gabriela F. Creţu-Ciocârlie, Mihai Budiu and Moises Goldszmidt. Hunting for problems with Artemis. *Proceedings of the First USENIX Workshop on Analysis of System Logs (WASL '08)*, 2008.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *USENIX Symposium on Operating Systems Design and Implementation*, volume 53, pages 137–150, 2004.

Matthew Dickerson, David Eppstein, Michael T. Goodrich and Jeremy Meng. Confluent Drawings: Visualizing Non-planar Diagrams in a Planar Way. *Journal of Graph Algorithms and Applications*, 9(1):10, 2002.

Nils Faltin. Algorithmen lernen mit interaktiven Visualisierungen. *Informatikunterricht und Medienbildung, INFOS 2001*, pages 87–96, 2001.

Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker and Ion Stoica. X-Trace: A Pervasive Network Tracing Framework. *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation*, pages 271–284, 2007.

Peter A. Gloor. User Interface Issues For Algorithm Animation, In *Software visualization: Programming as a multimedia experience*, pages 145–152. MIT Press, 1997.

Dachuan Huang, Xuanhua Shi, Shadi Ibrahim, Lu Lu, Hongzhang Liu, Song Wu and Hai Jin. MR-Scope: A Real-Time Tracing Tool for MapReduce. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, pages 849–855, 2010.

Christopher Hundhausen. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.

Andy Konwinski and Matei Zaharia. Finding the Elephant in the Data Center: Tracing Hadoop, 2008. URL `http://radlab.cs.berkeley.edu/w/upload/1/16/Hadoop-tracing-class-report294.pdf`. [Accessed 30 September 2010].

Andy Konwinski, Matei Zaharia, Randy Katz and Ion Stoica. Monitoring Hadoop using X-Trace, 2007. URL `http://www.cs.berkeley.edu/~matei/hadoop_tracing_poster.pdf`. [Accessed 21 October 2010].

Ari Korhonen, Lauri Malmi, Panu Silvasti and Ville Karavirta. Matrix - A Framework for Interactive Software Visualization, 2004. URL `http://www.cs.hut.fi/Research/SVG/publications/B154.pdf`. [Accessed 30 September 2010].

George Lawton. Distributed Data-Analysis Approach Gains Popularity, 2010. URL `http://www.computer.org/portal/web/computingnow/archive/news050`. [Accessed 20 October 2010].

Vassilios Lazaridis, Nikolaos Samaras and Angelo Sifaleras. An Empirical Study on Factors Influencing the Effectiveness of Algorithm Visualization. *Computer Applications in Engineering Education*, 2010.

Lauri Malmi, Ville Karavirta, Ari Korhonen, Jussi Nikander, Otto Seppälä and Panu Silvasti. Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, 2004.

Matthew L. Massie, Brent N. Chun and David E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30 (7):817–840, 2004.

Harvey B. Newman, Iosif Legrand, Philippe Galvez, Ramiro Voicu and Catalin Cirstoiu. Monalisa: A Distributed Monitoring Service Architecture. *Computing in High Energy and Nuclear Physics 2003 Conference Proceedings*, 2003.

Clifford Shaffer, Matthew Cooper, Alexander Joel Alon, Monika Akbar, Michael Stewart, Sean Ponce and Stephen Edwards. Algorithm Visualization: The State of the Field. *ACM Transactions on Computing Education (TOCE)*, 10(3), 2010.

Jiaqi Tan. *Log-based Approaches to Characterizing and Diagnosing MapReduce Systems.* Master's thesis, Carnegie Mellon University, 2009.

Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi and Priya Narasimhan. Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop. *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 18–18, 2009.

Jiaqi Tan, Soila Kavulya, Rajeev Gandhi and Priya Narasimhan. Visual, Log-Based Causal Tracing for Performance Debugging of MapReduce Systems. *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 795–806, 2010.

Brad Topol, John T. Stasko and Vaidy Sunderam. Integrating Visualization Support Into Distributed Computing Systems. *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 19–26, 1995.

Jayson Vantuyl. Distributed Dragons: Why Distributed Algorithms Are Different, 2010. URL `http://needlesslymessianic.com/2010/08/29/distributed-dragons-why-distributed-computing-is-so-hard`. [Accessed 11 November 2010].