Crowdsourcing MapReduce: JSMapReduce

Philipp Langhans
Institute for Informatics, LMU
Oettingenstr. 67
Munich, Germany
philipplgh@googlemail.com

Christoph Wieser
Institute for Informatics, LMU
Oettingenstr. 67
Munich, Germany
c.wieser@lmu.de

François Bry
Institute for Informatics, LMU
Oettingenstr. 67
Munich, Germany
bry@lmu.de

ABSTRACT

JSMapReduce is an implementation of MapReduce which exploits the computing power available in the computers of the users of a web platform by giving tasks to the JavaScript engines of their web browsers. This article describes the implementation of JSMapReduce exploiting HTML 5 features, the heuristics it uses for distributing tasks to workers, and reports on an experimental evaluation of JSMapReduce.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel programming; D.2.11 [**Software Architectures**]: Patterns

General Terms

Algorithms

Keywords

MapReduce, JavaScript, Crowdsourcing

1. INTRODUCTION

MapReduce [3] has evolved from a proprietary Google programming model for data parallel computation of PageRank [11] to a popular approach for solving data parallelizable problems with a cluster or grid of computers. In the meantime, the applications of MapReduce range from machine learning [4] to evaluating queries to databases like CouchDB [1]. Implementations of MapReduce have reached productive state among others Apache Hadoop [17] or Skynet [7].

Many web platforms are intended for an interactive use: They deliver data or services upon request of users browsing through texts such as catalogs, documentation, or data summaries. Such a functioning leaves much computing power at the users' side unused because of the latency inherent to human reactions. JSMapReduce aims at tapping in this computing power shifting load to the idle processors of website visitors. In the following, potential application areas for JSMapReduce are looked at in more detail.

E-Learning.

Most E-Learning platforms would be ideal candidates for deploying JSMapReduce because learning requires time inducing high latencies to user reactions, and because learning analytics, a key feature of today's E-Learning platforms, are mostly well amenable to data parallelism. Learning analytics can be used among others to adapt teaching material to learners' profiles. This can be done, e.g., using a clustering algorithm like K-Means [10]. [12] describes a MapReduce implementation of K-Means.

Social Network.

Social network analysis heavily relies on linear algebraic computation like eigenvector computations that are perfect candidates for data parallel computation using MapReduce. [14] for example describes an application of MapReduce to determine social influences in a social network.

E-Commerce.

E-Commerce often exploits the long tail of a market, that is, benefits from goods sold in small quantities which in turn requires to adapt the offers to the clients. Recommender systems used for this purpose by E-Commerce platforms are often based on algorithms amenable to data parallelism and MapReduce (see for example [9]) and therefore to JSMapReduce.

GWAP platforms.

Games with a Purpose [16][15] are the initial motivation for JSMapReduce. We are using several GWAPs for field research collecting data about artworks with ARTigo[1] and about the usage of the Italian language with Metropolitalia[2]. Data collected with GWAPs is processed with MapReduce for building a semantic search index.

1.1 Common Characteristics of the Use Cases

Human activity in the foreground and machine computing in the background are the key characteristics of the afore mentioned use cases. More generally, these characteristics are met on websites with high *screen time* and *low user interaction.* Many such websites rely on MapReduce. Most interestingly, in using JSMapReduce the computational power provided for running MapReduce scales with the number of users.

1.2 Client-side Technical Requirements

The barriers to participate in calculating JSMapReduce jobs are low. Basic requirements are a web browser implementing the *Web Worker* [6] API of HTML 5 [5] enabling

[1]`http://www.artigo.org`
[2]`http://www.metropolitalia.org`

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

multi-threaded JavaScript and an Internet connection. The Web Worker API allows processing MapReduce jobs in the background without affecting the functionality of a website in the foreground. For ensuring high quality of service, users should have long screen times, that is, stay on a website as long as possible. Additional software installations or network configurations (e.g., firewall settings) are not needed.

2. RELATED WORK

MapReduce [3] is a programming model for the concurrent processing of large data sets on clusters or grids. MapReduce has been implemented in several languages. The most famous implementations are the internal implementation of Google and its open source adaption Hadoop [17]. MapReduce applications are working on the basis of key/value pairs. The user of MapReduce provides a Map and a Reduce function. The application of the Map function on provided data during a first step yields intermediate data being input to the Reduce step. The Reduce step calculates the final result. The Runtime System of MapReduce is responsible for parallelizing, error handling, scheduling, and load balancing. One of the strengths of MapReduce is hiding the complexity of the execution from the user.

BOINC and similar systems also orchestrate PC grids and in particular allow solving MapReduce jobs [2]. However, these systems are not web browser-based.

To the best of the authors' knowledge, there has been no approach using a grid of web browsers for MapReduce.

A different system also called JSMapreduce[3] we got aware of after naming and referring to our system aims at testing MapReduce in one single web browser. Productive calculations are not performed on the web browser.

3. JSMAPREDUCE

JSMapReduce adapts MapReduce to the web context. The *User* submits MapReduce jobs via a web browser to the *Runtime System*, which manages the processing on many *Workers*. Each worker is a web browser with a JavaScript engine, that is displaying for example a website of the mentioned application areas in Section 1.

3.1 Challenges in the Web Context

In contrast to the controlled cluster or grid environments for traditional MapReduce calculations, the web as one factor and human beings as the other factor cause challenges for processing MapReduce jobs. In the web context HTTP 1.1 is the only widespread choice for client server communication for HTML and JavaScript usage. The first challenge is, that only clients can establish connections to the server for sending and receiving messages. As a consequence, the Runtime System of JSMapReduce can act as a managing instance, only if a client asks for new instructions. Therefore, decisions on distributing MapReduce tasks (i.e. a subproblem of a MapReduce job) to web browsers are based on uncertainty. Another challenge concerning HTTP is that the protocol is stateless. Hence, each request of a worker is treated independently and requires extra management effort. Uncertainty is also an issue concerning the perspectives of an efficient and exhaustive processing of MapReduce jobs. First of all the diversity of workers regarding computational power and the quality of the JavaScript engine affect the efficiency. Secondly the quality of the transmission rates between workers and the runtime environment for exchanging jobs and results has a high impact on the performance of a worker. In fact both the speed of calculations as well as the quality of the data transfer are crucial for completing MapReduce jobs successfully. Finally, human beings are a source of uncertainty. As mentioned above, JSMapReduce runs successfully if deployed in an environment where the screen time is long enough to complete the job because leaving a website with JSMapReduce support makes the run time engine wait for results it will never receive. Running a grid of web browsers with JSMapReduce means that one has a dynamic number of workers that are available to perform tasks, while the whole MapReduce job may have already started. This means that one has to handle new workers as well as vanishing workers. Fraud (intended or not) is an additional challenge in the JSMapReduce approach. Since we cannot control the clients' environment, we cannot ensure that the results we get were calculated as specified in a traditional cluster or grid environment.

These above-mentioned kinds of uncertainty require several strategies described below. We identified profiles of workers that can be detected in an early state of processing. These profiles allow a significant optimization during the execution of MapReduce calculations.

3.2 Architecture of JSMapReduce

Starting a JSMapReduce job means submitting input data and functions to the *JSMapReduce Client*, that interacts with the *Runtime System* being responsible for coordinating the *JSMapReduce Workers*. Solutions of Map or Reduce jobs calculated by the *JSMapReduce Workers* are aggregated by the *Runtime System* and presented to the user via the *JSMapReduce Client*.

JSMapReduce Client.

The JSMapReduce client is the interface for users that want to process MapReduce jobs. Obviously, the client allows submitting data as well as the Map and the Reduce function. The client manages the communication with the Runtime System via a simple protocol. To compensate the only stateless HTTP requests, the client can be set to states like listening.

Runtime System.

The Runtime System, consisting of the three components *Bridge*, *Master* and *Scheduler*, is responsible for creating, distributing, monitoring and bundling tasks.

The **Bridge** is a standalone socket-server receiving commands and MapReduce jobs from users. The bridge reports on the status of processing, initiates the distribution of data and actuates preprocessing steps in the master. The **Master** is the core component of the JSMapReduce framework and communicates with the remaining components to manage the processing of a MapReduce job. The master receives input data from the bridge and preprocesses them. Based on the preprocessing steps, the master creates tasks (subproblems) managed in a database. A task can be immediately processed or delayed. The master is also responsible for the registration of new workers. Only workers fulfilling minimal requirements are accepted to prevent management overhead and abuse. Additionally, the master manages interme-

[3]`http://www.jsmapreduce.com`

diate results and aggregates partial results. The **Scheduler** supervises each single worker by detecting timeouts (e.g., if a user left a website) and delivering tasks available in the database. The scheduler can assign tasks redundantly to ensure the quality of service.

JSMapReduce Worker.

If a web browser is online on a website with JSMapReduce support, it is called a JSMapReduce worker. Visiting a JSMapReduce website executes JavaScript code in the background via the HTML 5 Web Worker API and connects to the Runtime System. Due to the limited stateless HTTP protocol the server-push technique (COMET [13]) is used to keep the connection to the Runtime System alive for further communication. As soon as tasks are available, the worker receives them. This approach grants enough freedom to the runtime engine to control the amount of processing on the worker without busy waiting. Before being sent to the workers, original input data is partitioned into *chunks*. The size of a chunk varies according to the identified profile of the worker. For example, reliable workers with good Internet connections and fast processors will get the largest chunks. Chunks are compressed for the transfer.

The strategy for scheduling workers is hybrid. First choice are workers with high quality rating, afterward quantity replaces quality, and finally the Runtime System itself is the last standing worker to process the data chunks until new workers register, when visiting a JSMapReduce supported website.

3.3 Evaluation

For the sake of evaluating JSMapReduce, we used the classic video game Snake in the foreground. The implementation is based on the new HTML 5 canvas element. When playing Snake, users collect items by controlling a snake with the keyboard. In the background JSMapReduce tasks are processed without influencing the highly interactive game.

The 240 website visitors during the evaluation processed more than 70.000 tasks with a failure rate of 0.25 %. Each player on average processed 449 MB (uncompressed), playing 2:11 minutes. The data chunks were compressed to a tenth of their original sizes for transfer. The average transfer rate was 1 GB in 113 seconds. The average processing time of the chunks was 26 seconds / 1 GB, while the users stayed for 131 seconds on the website on average. Hence, more than 1 GB of (uncompressed) data was processed during one game session.

As a positive result of a subsequent online-survey, no user reported effects of the massive calculations in the background on the game application in the foreground. 21 % complaints about occasional glitches, the others felt confident. Concerning allowing additional calculations in the background only 15 % of the users raised doubts but none rejected the idea.

In our evaluation the server was a bottleneck and caused overheads of 98 seconds per GB. Optimizing the server is subject to further research on JSMapReduce. Our tests showed that the best performance could be obtained with 20 concurrent workers.

A bottleneck that cannot be influenced by software optimization of JSMapReduce (in contrast to controlled grid environments) is the transmission rate between the Runtime System and workers. This fact makes JSMapReduce suitable for MapReduce jobs that draw on exploiting the CPU rather than relying on massive data exchange.

The evaluation proved that the quality of the workers in terms of Internet connection and processor was of significant importance in comparison to the quantity of workers. Strategies, based on measurements of response time and time requirements (see Figures 1 and 2), to preselect workers of high quality are discussed in [8].

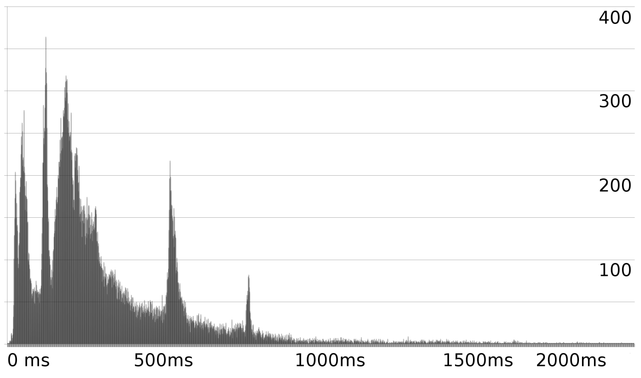

Figure 1: The histogram shows that the workers' response time was on average less than half a second.

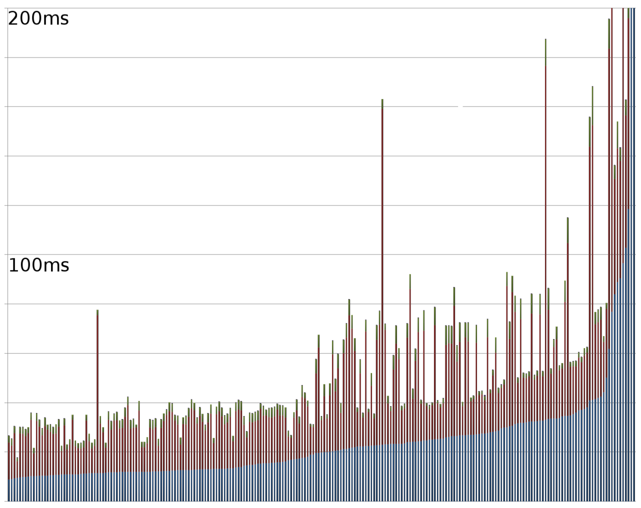

Figure 2: Time requirements of the steps Init (blue), Map (red), Group (green), and Reduce (purple).

4. DEMONSTRATION

The demonstration described in this Section is available as a screencast on `http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2013-1`. The classical example for using MapReduce is a logfile analysis. For the sake of demonstrating JSMapReduce, we generated sample logfiles that should be transformed to monthly statistics about website visitors. A basic logfile contains 1 million entries occupying 150 MB space. The data is provided via a database and also the Map and the Reduce functions for this demo case are also provided.

The demo case starts with declaring in the JSMapReduce Client how to portion the 1 million entries into smaller data chunks for the workers. We choose data chunks of 2000 lines each and start processing the logfile analyzing job. The progress of the whole job is visualized by dots arranged in a box. One dot stands for the status of one data chunk. White means *untreated*, blue means *in progress*, green means *completed*, and red means *error*. The number of concurrent workers in progress can be determined by the number of blue dots. At first we demonstrate a JSMapReduce job executed with only one worker rendering a session of the video game Snake in the foreground. Obviously this highly interactive game runs smoothly, while a data chunk is processed in the background at the same time, although multiple snake updates need to be rendered every second. As a second step we demonstrate processing data chunks with multiple workers as expected in MapReduce frameworks. Therefore several browser games are started at the same time registering themselves as workers on the Runtime System. When running the MapReduce job, several blue dots are displayed at the same time, corresponding to the active workers. As expected all Snake instances run smoothly.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented an implementation of MapReduce in JavaScript using the web browsers of common website visitors as workers. JSMapReduce was originally conceived for a distributed calculation of an index for the semantic search engine of the GWAP platform ARTigo.

The implementation benefits from the Web Worker API introduced in HTML 5 allowing multi-threaded processing of JavaScript programs. We proved that intense processing and transmission of data in the background has no perceptible effect on the web browser interactions in the foreground. This makes JSMapReduce suitable for application areas needing cheap and scalable computational power with a growing number of users on their website. The longer the a user stays on the website, the more MapReduce problems can be calculated. Despite having evaluated the approach with a prototypical implementation, the results are promising in comparison to established MapReduce implementations such as Hadoop. Subtracting the overhead of the web transmissions, the performance is similar to the Hadoop implementation of MapReduce as described in [8].

Further work on implementations of the JSMapReduce approach will benefit from our heuristics to ensure quality of service the on one hand, mainly by choosing reliable workers in an early stage of processing, and from coping with a dynamic number of workers on the other hand.

The JSMapReduce approach allows bundling human thinking and computational power, therefore upgrading the crowdsourcing approach to a higher stage. Having presented our first JSMapReduce prototype in this paper, many options for optimization are planned, amongst others analyzing OS performance, error rates, load balancing, intra-worker performance, performance variations, client-side storage, and overhead reduction to mention some of them.

Acknowledgements

This research has been funded by the Deutsche Forschungsgemeinschaft (DFG) within the Play4Science project number 578416 (cf. `http://www.play4science.org`).

6. REFERENCES

[1] J. Anderson, J. Lehnardt, and N. Slater. *CouchDB: The Definitive Guide: Time to Relax.* O'Reilly, 2010.

[2] F. Costa, L. Silva, and M. Dahlin. Volunteer Cloud Computing: MapReduce over the Internet. In *Intl. Symp. Parallel & Distributed Processing.* IEEE, 2011.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *6th Conf. Operating Systems Design and Implementation*, 2004.

[4] C.-T. C. et al. Map-Reduce for Machine Learning on Multicore. In *Proc. 20th Conf. on Neural Information Processing Systems*, 2006.

[5] R. B. et.al. HTML5. http://www.w3.org/TR/2012/CR-html5-20121217/.

[6] I. Hickson. Worker. http://www.w3.org/TR/workers/.

[7] H. e. a. Jin. The MapReduce Programming Model and Implementations. *Cloud Computing: Principles and Paradigms*, pages 373–390, 2011.

[8] P. Langhans. JSMapReduce - Distributed Computing with Web Clients and LAMP, Institute of Computer Science, University of Munich. bachelor thesis, 2012.

[9] H. Liang, J. Hogan, and Y. Xu. Parallel User Profiling Based on Folksonomy for Large Scaled Recommender Systems: an Impl. of Cascading MapReduce. In *10th Intl. Conf. Data Mining.* IEEE, Dec 2010.

[10] J. e. a. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, page 14, 1967.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, Nov 1999.

[12] R. Raju, V. Vijayalakshmi, and R. Showmya. E-Learning Using Mapreduce. *Intl. Journal on Computer Science and Engineering*, 3(4), 2011.

[13] A. Russell. Comet: Low Latency Data for the Browser. http://goo.gl/mXTdI, March 2006.

[14] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In *Proc. 15th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 807–816. ACM, 2009.

[15] L. von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8), 2008.

[16] L. von Ahn et.al. Improving Accessibility of the Web with a Computer Game. In *Proc. Conf. on Human Factors in Computing Systems.* ACM, 2006.

[17] T. White. *Hadoop: Definitive Guide.* O'Reilly, 2012.