university of groningen

faculty of science and engineering

Highly Distributed In-Browser Computing

Bachelor's Thesis

July 2019

Authors
George Argirousis
Tolga Parlan

Primary Supervisor
Who?

Secondary Supervisor
Frank Blaauw

CONTENTS

ABSTRACT

Here be the abstract

1

INTRODUCTION

Modern web relies increasingly on Javascript, which is used almost ubiquitously in client-side of any website and is becoming a progressively more popular tool for server applications through Node.js. Client-side Javascript often has the purpose of making websites more reactive to user actions by offloading some application logic to the client's browser. This approach is increasingly popular due to user computing devices getting ever more powerful, and serious innovations in Javascript engines (such as Chrome V8) and related technologies making it ever more sensible to invest development time into shifting computational tasks to the client-side. Performance problems that used to plague webpages with heavy computations and necessitated solutions like Java applets can now be ignored in most cases [6]. This has been exposing a considerable amount of computing power to the website owners in the form of their users' devices, since all the users visiting their page run code written by them. This exposed computing power can grow to serious proportions for a decently popular website that coordinates the computations across different user machines smartly. This brings up the concept of **Browser Based Voluntary Computing** [6], which is a subset of more general **Voluntary Computing**.

Distributing a large pool of computing tasks to many computers across the internet and letting them run the calculations is an idea which is now more than two decades old and includes successful projects such as BOINC [3]. Such projects have been usually termed voluntary computing since the participants would actively volunteer to run such tasks on their machines, usually via signing up on a website and downloading a purpose-made program.

Another idea in this field of distributing tasks to browsers, which comes from a similar premise while addressing a different situation is given the name **Gray Computing**. [10]. Gray computing refers to a non-voluntary category of task distribution on browsers where users don't explicitly consent to running the given computations on their browsers, but the implicit consent which comes from the fact that any website can run arbitrary Javascript on it's visitors computers is used to create a similar architecture. This environment presents different technical challenges and opportunities, which are explored in the next chapter.

This project aims to develop a powerful, simple and highly customizable open-source gray computing tool which can help to unearth the aforementioned underused computing potential of billions of devices that access internet via web browsers. We envision that harnessing this processing power, with the right tools, can create a serious alternative revenue stream for many websites, nullifying or

decreasing the need for ads, while serving useful scientific, social and commercial purposes.

For these ends, we have implemented an npm package [2] which integrates seamlessly with any Node.js http server instance [1] and has an easy to use and highly customizable interface for programmers, behaving like a native Node.js package. We believe that this unopinionated approach that allows other implementers to run their own tasks in their own way with no need to change anything else in their codebase to accommodate our package is a serious advantage over other similar projects, and can help to resolve the relative lack of real life popularity of past browser based distributed computing projects. We discuss this topic more in the next chapters.

Our code is open-source and can easily be reached by anyone who wants to contribute with new features or performance improvements, or simply fork from our project and use it as a starting point in their own projects. With this aim we are housing our project as a public repository on github and licensing it under the permissive MIT License.

Going further, this paper will be structured as follows: In Chapter 2 we give a summary of related work, explaining how we have benefited from the design and technology choices of similar projects to ours. The chapter further quotes from survey papers which have examined many past projects, and goes on to explain in which ways we have attempted to differ in our project in order to produce a solution that can be adopted easily by people for diverse tasks. In Chapter 3 we summarise our code architecture, and talk about the main components of our program in more detail. Chapter 4 focuses on evaluating different types of tasks and their performance on our system. Finally in Chapter 5 we present our conclusions and in Chapter 6 we present ideas that can be considered to improve the project in the future work.

2

RELATED WORK

As mentioned previously, the concept of a distributed system that uses other, previously unknown computers as nodes through running code on their web browsers and communicates with them over internet is nothing new. There have been many attempts at creating such distributed networks, with different approaches, architectures and aims. Here we give a quick breakdown of the past projects that this project has directly or indirectly benefited from. We also attempt an investigation of why browser based distributed computing projects haven't caught on popularity-wise, and speculate how our architecture may provide a more practical alternative.

We follow the three generation paradigm proposed in [6], and concern ourselves mainly with what they define as the **Third Generation**. This generation is distinct from the previous ones because of the new Javascript features it can benefit from (and the fact that Javascript is used instead of Java applets) such as a fast compiler, thread support (via WebWorkers) and WebSockets. The reason for this is our project highly resembles most other such examples in terms of the tech stack, and thus they are the most relevant examples we can use to examine how to improve current systems. We also mention only projects that are meant for general use and aren't simply prototypes or benchmark platforms. After we examine several projects, we delve into arguing about the major problems such projects faced and what direction can be taken in the future to avoid them. Later we briefly mention the Map-Reduce paradigm which has been influential in our approach.

2.1 SIMILAR PROJECTS

2.1.1 *QMachine [12]*

QMachine is an up and running implementation of the distributing jobs to browser clients idea. It has three different components: **browsers** which do the actual computations and submit new computations, an **API Server** which receives computation requests from the browsers and distributes these computations to other, available browsers. Also a **Web Server** is needed to serve the initial code to the browser clients. The API Server should be installed and ran individually by the user who wants to create a network for their own purposes, and it can use only a given set of databases with pre-implemented integrations. The author assumes that QM will be used by closed scientific groups who will then invite only trusted volunteers, thus the main security feature involved is verifying the participants are indeed only the trusted volunteers. The communication between the API server and the browsers use plain AJAX requests to submit jobs

to the server, and to poll the server for new jobs. The QM class written by the authors can be used as outside of a browser as well, using Node.js.

Judging from its github activity, npm downloads and how long it took for another user to connect to its main API Server and process basic test jobs submitted by us, the project doesn't seem very alive. We suspect several factors contributed to this relative inactivity.

- It is not clear how this system would generalize to open internet, given its lack of any sabotage-tolerance features. In an environment where not everyone is a trusted volunteer as assumed by the system, one browser busying up the network with very large jobs or sending wrongly calculated results back would easily sabotage the computation. Distributing computational tasks makes sense practical only if there is an abundance of computing nodes. However a system in which only trusted volunteers can participate is severely limited in scalability.

- The system has little flexibility in how the computed data is later on stored or handled. The computed data has to be stored on a database, and the API Server owner needs to provide a database system already supported by the QMachine. This imposes an unnecessary limitation for what sort of use cases this system can be used.

- MAYBE MENTION POLLING DISADVANTAGES

We speculate that these factors stop the system from being viable in most practical scenarios. However the project definitely presents useful ideas such as having a separate server running the distributed system independently of any other web server. We think that this modularity is very important for easy integration into any other project. Furthermore packaging the client code so that it can simply be a part of any web page helps in this aspect too. [TALK ABOUT THEIR USER NUMBERS]

2.1.2 *MLitB [9]*

MLtiB, Machine Learning in the Browser is a prototype Machine Learning that is intended to provide an easy platform for researchers to run their machine learning jobs on, using the computing power provided by volunteers. When users connect to the system, they interact with a UI from which they can create workers (which are implemented as WebWorkers) to perform various tasks. The system relies on the assumption that the contributors will voluntarily go to a web page intended for machine learning and create these workers. After the initial transfer of data (which is compressed to speed up this transfer), all communication happens through WebSockets. They have two servers, one for distributing data and another master server for coordinating the process. Both of them are written in Node.js, and the authors justify this design decision with the event-driven and lightweight architecture of Node.js servers: "Since the main computational load is carried by the clients, and not the server, a light-weight

server that can handle many clients concurrently is all that is required by MLitB."

mapreduce

2.2 COMMON PROBLEMS

2.2.1 *User Retention*

A common problem that is introduced by using browsers as the computation medium for a distributed network is convincing the users to visit the relevant website and stay there. This is a problem that earlier projects such SETI@HOME [CITE] or BOINC did not face, since in these projects volunteer download and install a program which then runs in the background. In contrast, computation stops when a tab is closed in a browser. We think that the common design approaches usually do not help with this problem.

A broad survey on the state of Browser Based Voluntary Computing by Fabisiak & Danilecki (2017) concludes that "the problem of recruiting the users is the most important challenge faced by browser-based platforms". One common theme amongst all the projects they have inspected is either a very low number of volunteers, or the numbers not being reported at all. In the paper the authors suggest that incentive systems employed by older generations of voluntary computing systems, such as reputation systems, virtual credits or real monetary awards based on computing power contributions could to some degree be adapted to the newer systems. However this suggestion in our opinion ignores one of the most important properties of browser based systems. Browser based systems present a much lower barrier of entry than installing a program on one's computer, however this property brings a lot much higher retention rates, with most users not contributing more than once [CITE NUMBERS]. This is rather understandable when an average internet surf routine is inspected. Browser tabs are designed to be easily disposable and many users open and close plenty of tabs routinely. Getting a user to keep a tab open for a meaningful amounts of time, and routinely come back is difficult and we suspect most projects fail at this, even when they have successfully attract a decent number of contributors initially such as QMachine.

We think that gray computing practices can be a solution to the aforementioned user retention problem. It has proven difficult to attract volunteers, who would spend meaningful amounts of time, we think a decent solution would be to embed such systems in already popular websites with which users engage for a long time. Thus we have strived to make our system easily adaptable into any enot contributing more than once [CITE NUMBERS]. This is rather understandable when an average internet surf routine is inspected. Browser tabs are designed to be easily disposable and many users open and close plenty of tabs routinely. Getting a user to keep a tab open for a meaningful amounts of time, and routinely come back is difficult and we suspect mostxisting website with adding only a small script to the client-side code and keeping the server implementation as simple as possible.

2.2.2 *Server Adaptability*

We have observed that many projects in the field of Browser Based Volunteer Computing are done with one sort of calculation in mind. Apart from MLitB examined before, most other projects we have examined such as Zorrilla et al. (2013) [13] with their social networking emphasis, Krupa et al. (2012) [7] with their investigation limited to web search, Duda & Dlubcaz (2012) [5] targeting only evaluationary algorithms and so forth. While the investigative value of these papers are indisputable, they are not really adaptable to many other purposes.

A similar problem that arises from this problem-centric approach is that the program architectures aren't built with much modularity. This is a common theme that might be harming the adaptability of the projects to different environments, as trying to provide a Swiss army knife solution, which attempts to be very simple to use by providing as many as features as possible is not necessarily the best software development approach. [GIVE EXAMPLES]. This is understandable since many projects have other scientists working on diverse fields who might need computing power to run their calculations on. However this often leads to projects that are rather complicated, bloated and possible bug-prone [GIVE EXAMPLES]. As projects grow and get more complicated, it becomes increasingly more difficult to adapt for other users, and maintain for the initial creators. For example a user of the QMachine system who doesn't want to persist the data in one of the presented database options is going to have to employ some software hacks. Someone who would like to use their own servers cannot run the MLitB system since it requires its own data and machine learning servers to be used.

2.2.3 *Performance*

Langhans et al. [8], embed their Map-Reduce solution doing large calculations in the back-ground while the user is busy playing a short browser game, and then survey the users. In their survey, "no user reported effects of the massive calculations in the background on the game application in the foreground. (...) Concerning allowing additional calculations in the background only 15% of the users raised doubts but none rejected the idea". They explain this by WebWorkers moving the calculations to the background effectively. Indeed we agree that the WebWorker technology makes gray computing running background tasks much more feasible for websites that do not want the user experience to take a hit. A set of projects that we have gone over such as [FIND EXAMPLES] do not use the WebWorkers technology, making them difficult to adapt in situations where web surfing quality of the users is a consideration. WebWorkers is currently supported in every single major desktop or mobile browser (LINK) and we have implemented them in our project as well.

Bottlenecks

TALK ABOUT Server being the bottleneck, the need for an efficient server implementation, advantages of NODEJS and streams

Maybe also about http calls being slow, and using sockets instead?

2.3 MAP-REDUCE

2.3.1 *MRJS [11]*

MRJS is one of many projects inspired by Google's popular Map-Reduce architecture [4], intending to move this type of task distribution into the browsers. Map-Reduce paradigm is useful in a broad range of computational tasks, and has the side-advantage that the it presents a very high level and intuitive interface for the programmers to interact with, hiding away details that would require experience with distributed systems to understand. MRJS also aims to give job creators control over parameters that might affect performance. The idea of hiding the distribution details from the programmer who is using the system, while giving optional access to parameters related to performance is a principle we have tried to adhere to in our program as well. [WRITE MORE] ALSO THE OTHER MAP REDUCE JS

3

ARCHITECTURE

4

EVALUATION

5

CONCLUSION

6

FUTURE WORK

BIBLIOGRAPHY

[1] Node.js v12.3.0 documentation.

[2] npm.

[3] David P Anderson. Boinc: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10. IEEE Computer Society, 2004.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[5] Jerzy Duda and Wojciech Dłubacz. Distributed evolutionary computing system based on web browsers with javascript. In *International Workshop on Applied Parallel Computing*, pages 183–191. Springer, 2012.

[6] Tomasz Fabisiak and Arkadiusz Danilecki. Browser-based harnessing of voluntary computational power. *Foundations of Computing and Decision Sciences*, 42(1):3–42, 2017.

[7] Tomasz Krupa, Przemyslaw Majewski, Bartosz Kowalczyk, and Wojciech Turek. On-demand web search using browser-based volunteer computing. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 184–190. IEEE, 2012.

[8] Philipp Langhans, Christoph Wieser, and François Bry. Crowdsourcing mapreduce: Jsmapreduce. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 253–256. ACM, 2013.

[9] Edward Meeds, Remco Hendriks, Said Al Faraby, Magiel Bruntink, and Max Welling. Mlitb: machine learning in the browser. *PeerJ Computer Science*, 1:e11, 2015.

[10] Biswajit Pathak and Debajyoti Barooah. Texture analysis based on the gray-level co-occurrence matrix considering possible orientations. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2(9):4206–4212, 2013.

[11] Sandy Ryza and Tom Wall. Mrjs: A javascript mapreduce framework for web browsers. *URL http://www. cs. brown. edu/courses/csci2950-u/f11/papers/mrjs. pdf*, 2010.

[12] Sean R Wilkinson and Jonas S Almeida. Qmachine: commodity supercomputing in web browsers. *BMC bioinformatics*, 15(1):176, 2014.

[13] Mikel Zorrilla, Angel Martin, Iñigo Tamayo, Naiara Aginako, and Igor G Olaizola. Web browser-based social distributed computing platform applied to image analysis. In *2013 International Conference on Cloud and Green Computing*, pages 389–396. IEEE, 2013.