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 [5]. 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** [5], 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**. [6]. 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 system proposed in [5], 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.

QMACHINE [8]

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.

MRJS [7]

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]

MLTIB

WRITE THIS

SUMMARY[BETTER NAME?]

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. [TALK ABOUT LOW PARTICIPANT NUMBERS].

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

[6] 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.

[7] 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.

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