Madoop: Improving Browser-Based Volunteer Computing Based on Modern Web Technologies

Hiroyuki Matsuo, Shinsuke Matsumoto, Yoshiki Higo and Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
{h-matsuo, shinsuke, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Browser-based volunteer computing (BBVC) is one of the distributed computing paradigms, attracting researchers' and developers' attention for its portability and extraordinary potential of computing power. However, BBVC still has two significant challenges: low programmability and performance. These challenges are a heavy burden for users and prevent BBVC from wide-spreading. In this paper, we propose a novel BBVC framework to solve the challenges by using MapReduce and WebAssembly. Our framework reduces the total execution time by 64% compared with a traditional BBVC mechanism. We also show a practical scenario and its performance.**

***Index Terms*—Volunteer computing, web browser, MapReduce, WebAssembly, JavaScript.**

I. Introduction

In the field of computer science, we are facing numerous demands for computational power to process compute-intensive and data-intensive problems. To meet the demands, there has been a state-of-the-art distributed computing paradigm, called *browser-based volunteer computing (BBVC)* [1], [2], which extends *volunteer computing (VC)* [3], [4] to working on web browser. BBVC has a tremendous potential that every web user could easily become a worker of distributed computing [2].

However, BBVC remains a significant challenge for software developers to implement parallel processing programs due to the heterogeneity of BBVC workers. The parallel programming has two kinds of difficulties: *how to parallelize a given problem?* and *how to manage multiple workers?* The former difficulty is a fundamental problem of the parallel programming. A developer needs to consider task and data parallelism [5] according to the characteristics of her/his problem. In addition, BBVC workers are heterogeneous in terms of their runtime environment (e.g., CPU, memory, network, and software platform). Such kind of heterogeneity makes it more difficult to achieve efficient parallel processing.

BBVC also has a performance issue because at present only JavaScript is used as an implementation language for processing program. The runtime performance of JavaScript is known to be insufficient [6], [7] for compute-intensive applications like numerical calculation. JavaScript code is dynamically interpreted and optimized on web browser on demand of user requests. Thus, the execution of JavaScript requires additional preprocessing phases compared to statically-compiled programming languages. Moreover, in the compiling phase, JavaScript engine spent much time to generate multiple binaries of the same source code because variable types are determined at runtime. This generation also leads to performance degradation.

In this paper, we propose a novel BBVC framework, named **Madoop**, to address the above challenges by incorporating concepts of MapReduce [8] and WebAssembly [7]. MapReduce is a programming model which simplifies the development of parallel programming by abstracting away the low-level complexity involved in distributed systems. Based on MapReduce, parallel processing can be achieved only by implementing map and reduce functions in MapReduce. The performance issue is also relieved by introducing WebAssembly as an implementation language instead of JavaScript. WebAssembly is an emerging web standard which enables to execute low-level and assembly-like language on web browser.

As an evaluation, we conduct an experiment to show the performance improvement by Madoop compared with the traditional BBVC. The results show that the execution performance improved about 50 to 64% for a compute-intensive problem.

II. Preparations

A. VC: Volunteer Computing

VC is a traditional paradigm to perform network-based distributed computing. VC is constructed from many and unspecified heterogeneous computers connected to the internet. Generally, most computers consume their uptimes as an idle state. As the computing resources (i.e., worker nodes), VC uses these unused processing powers provided by volunteers. These worker nodes are physically distant from each other, so they are interconnected over the internet, not a LAN.

B. BBVC: Browser-Based Volunteer Computing

BBVC is an extension of VC for web browsers. VC runs on a dedicated client-side application, while BBVC runs on various web browsers accessing a specific URL. Each web browser keeps performing BBVC computation during the user's stay on the page.

Web browsers are generally installed in most computers, so that users do not have to install any additional programs for participating in BBVC. Hence, BBVC targets internet surfing users all over the world, and its potential computing ability is immeasurable. Gray et al. [2] estimate that the total BBVC computing ability on YouTube could reach about 46.4 PFLOPS, if 25% resources of each YouTube user's computer were available for BBVC. This is about half of the

978-1-7281-0591-8/19/$31.00 © 2019 IEEE

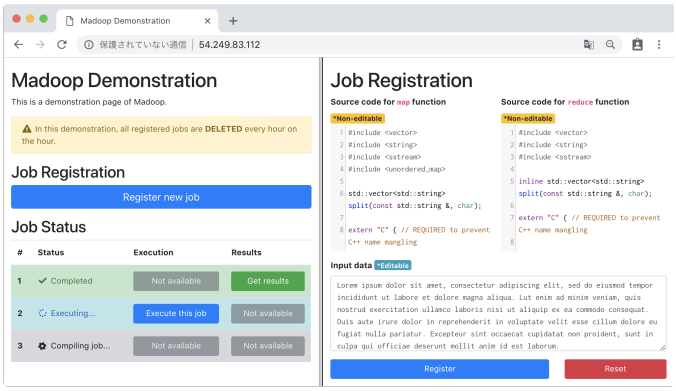

Fig. 1. Screenshot of Madoop demonstration page

computing power of Sunway TaihuLight, the world's fastest supercomputer as of April 2017 [2].

There is another benefit in BBVC, which is that developers can develop the client-side program just as a web application. Compared to VC, it costs much less to develop a client-side program for BBVC since web browsers hide each user's operating system or architecture.

C. Challenges in BBVC

As we mentioned, BBVC provides promising solutions for some VC problems. However, BBVC still remains the following challenges.

1) Difficulties in developing distributed computing programs (P_1): Distributed computing requires complicated mechanisms such as synchronization of workers, appropriate data division and distribution, tolerability of worker failures and integration of processed results [1], [2]. MapReduce paradigm is famous as a technology to support the implementation of distributed computing. However, there are few MapReduce libraries for web browsers [1]. As a result, when developers prepare BBVC programs, they have to implement the whole system by themselves with the above considerations in their minds. This is a heavy burden for developers and can be a factor preventing BBVC from being spread.

2) Low runtime performance of JavaScript (P_2): In BBVC, there have been no options except for JavaScript as the processing language, because of the nature that processing is performed on web browser. However, JavaScript is a dynamically-typed scripting language and is inferior to statically-typed compiled languages in terms of execution performance due to runtime analysis overhead. For example, Merelo et al. pointed out that JavaScript is 30% slower than Java when processing numerical calculation [6].

III. Madoop

A. Overview

This paper proposes a novel BBVC framework called Madoop[1] which introduces MapReduce and WebAssembly to solve problems P_1 and P_2 mentioned in the previous section. Madoop enables to facilitate the development of parallel programming based on the concept of MapReduce. The performance issue is also relieved by introducing WebAssembly as the processing language on worker nodes. Figure 1 shows the screenshot of Madoop demonstration.

[1]Named for the author's name **M**atsuo and Apache **Hadoop** which is famous for MapReduce implementation. Madoop is published as open-source software: https://github.com/h-matsuo/madoop/

B. Adopted Technologies

1) MapReduce: Madoop leverages MapReduce to solve the problem P_1: difficulties in developing distributed computing programs.

MapReduce [8] is one of the distributed computing paradigms proposed by Google. MapReduce abstracts a series of operations in distributed computing into two functions: map and reduce. The overview of MapReduce processing is below.

1) MapReduce system divides input data into appropriate units.
2) Each node receives input data and map function. The map function outputs a set of $(\texttt{key}, \texttt{value})$ pairs as an intermediate value.
3) MapReduce system collects all intermediate values associated with the same key and generates a set of $(\texttt{key}, \textit{list}(\texttt{value}))$ pairs.
4) Each node receives a pair of $(\texttt{key}, \textit{list}(\texttt{value}))$ and reduce function. The reduce function aggregates $\textit{list}(\texttt{value})$ and outputs the results.

Developers only need to implement map/reduce functions and do not have to consider complicated mechanisms in distributed computing. Thus, MapReduce succeeded in making the development of distributed computing programs easier. The idea of MapReduce is famous in the world, so that any developers who have developed MapReduce programs can begin to use Madoop with low learning cost.

2) WebAssembly: We introduce WebAssembly to solve the problem P_2: low runtime performance of JavaScript.

WebAssembly [7] is a binary format which is executable on web browser. Strictly speaking, it is designed as a binary format for a stack-based virtual machine, and we can consider that this virtual machine is installed in every web browser. Some languages such as C/C++ already support WebAssembly as a compilation target. All major modern web browsers are compatible with WebAssembly and its support rate have reached about 80% as of December 2018[2]. In other words, Web-Assembly now can be executed in many users' environments.

The most significant advantage of leveraging WebAssembly is a performance improvement. WebAssembly enables web browser only need to interpret and execute the pre-compiled binary so that the performance gets equivalent to the statically-compiled languages. Indeed, there are the benchmark results that the performance of WebAssembly is equivalent to that of the native applications when comparing the duration of numeric calculation [7].

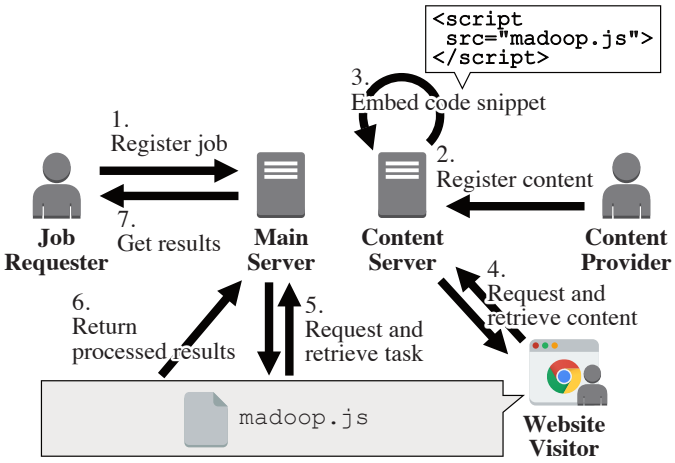

Fig. 2. Architecture of Madoop

C. Architecture

Figure 2 depicts the architecture of Madoop. This architecture contains three actors and two servers highlighted in boldface.

Job requester. Those who would like to perform distributed computing on Madoop.

Content provider. Those who post web contents such as blog articles.

Website visitor. Those who access the web contents using a web browser.

Main server. A web server with Madoop installed.

Content server. A web server which stores and serves the web contents.

The following list describes the procedure for processing the job. The numbers in the list correspond to the numbers in the figure.

1) A job requester registers a job request to the main server. The job request includes map/reduce functions and input data to be processed on Madoop. High performance can be expected by writing the map/reduce functions in C/C++ and compiling them into the WebAssembly format[3]. Madoop divides the registered job into appropriate units and holds them as tasks to be processed on each worker node.
2) A content provider creates and posts a web content such as a blog article to the content server.
3) The content server embeds a code snippet in the provided web content in order to let browsers perform distributed computing.
4) A website visitor accesses the content server using her/his web browser and the browser downloads various files such as HTML files and JavaScript files. Then, the code snippet embedded in step 3) begins to perform distributed computing. During a stay on the page, this web browser functions as a worker node for BBVC.
5) A worker node retrieves a task from the main server then processes the task. This task is safely executed because web browser provides a sandboxed environment and restricts the task from accessing the entire computer.
6) The worker node returns processed results to the main server. The worker node repeats steps 5)–6) until the website visitor leaves the page or there are no tasks to be processed.
7) After all tasks are processed, the job requester can check the results.

[2]https://caniuse.com/#search=wasm

[3]Madoop is also designed to be able to register the map/reduce functions written in JavaScript.

D. Scenarios of Madoop

The key factor to process computational complexity problems on Madoop is to ensure a great number of volunteer workers. In this section, we introduce three scenarios of ensuring Madoop workers.

1) Use Your Website Visitors: As we mentioned in Section III-C, only a modern browser is necessary to become a Madoop worker. Therefore, if you have your own website, you can make all visitors become workers by embedding Madoop's JavaScript snippet in the webpage. This portability enables that Madoop has a possibility to be an alternative to the online advertising platform, such as Google AdSense[4]. These days many website owners place advertisements on their websites and make revenue from their contents. However, these ads may become negative factors for website visitors. Goldstein et al. point out that the animated ads make users annoyed [9]. Madoop and traditional online ads have trade-off relationship between borrowing computational resource (including battery consumption) and sacrificing UI/UX for website visitors.

There needs to provide a means for visitors to opt out of being a worker. This kind of ethical issue has been discussed in the field of BBVC. However, there is also an opinion that it is essential to show the potential of BBVC-related technologies before deeply exploring the ethical topics [2].

2) Use Cloud Instances: You can also use cloud services, such as Amazon Web Services and Microsoft Azure, as Madoop workers. You do not have to set up the messy clustering environment. You can obtain your own distributed computing environment with high scalability, by just instantiating some virtual machines with a modern web browser installed. In the experiments, described in the next section, we introduce more details of this scenario.

3) Build BBVC Community: The last scenario is building a special BBVC community where many members participate with a common purpose. For example, if a researcher of software repository mining creates a specific BBVC community with many participants, they acquire their high-performance computing environment. Participants are not required to install a dedicated client program. By keeping launching a web browser all the time, every member can contribute to mining studies.

[4]https://www.google.com/adsense

IV. Experiments

A. Overview

The purpose of this experiment is to evaluate the performance improvements by introducing WebAssembly, compared to the conventional method which uses JavaScript. We set up the main server with Madoop installed and multiple client computers with a web browser installed. Then, we measure the duration from distributing a job to the completion of its computation to compare Madoop and the conventional method.

B. Experimental Environment

We use an Amazon EC2's t2.large instance as the main server. We install Madoop version 0.1.6 on the instance and make it public so as to be accessed from the outside. For simplicity, the main server also works as a content server in this experiment. We also use Amazon EC2's m3.medium instances for workers. Each worker has Google Chrome version 67.0.3396.87 and Mozilla Firefox version 60.0.2, then the web browser on each instance will access to the main server.

C. Experimental Objects

Table I shows the parameters for each experimental object. The details will be described below.

1) Rainbow Table Generation (E_1): Rainbow table [2] generation is known as a numerical calculation processing for efficiently performing brute force attack on a login password of a website. For security reasons, the user's login password is usually stored in a hashed form in the website database. Rainbow table is obtained by calculating the correspondences between password candidates and their hash values so that the memory efficiency is improved.

In this job, each worker only takes a string and the number of times for calculating the hash value as an input data. Its data size is so small that it can be neglected. On the other hand, since each worker calculates a large amount of hash value every time, the total computational complexity is very high. In this way, rainbow table generation has the characteristic of small amount of data and high complexity.

2) Word Count (E_2): Word count is one of the typical jobs in big data processing using MapReduce, which analyzes a large capacity text file and outputs a list of words with the number of their occurrences.

In this job, the amount of input data for each node is large (several MB to tens of MB), but the complexity of processing to count the words is low. This characteristic is in contrast to the previous E_1: rainbow table generation job.

TABLE I
Parameters for Experimental Objects

Common parameters	
# of worker nodes	1, 2, 3, 5 or 10
# of trials	10
E_1: Rainbow table generation	
Total # of hash calculations	10,000,000
# of hash calculations per task	1,000,000
Hash algorithm	MD5
E_2: Word count	
Total data size of text data	380 MB
Data size per task	44 MB

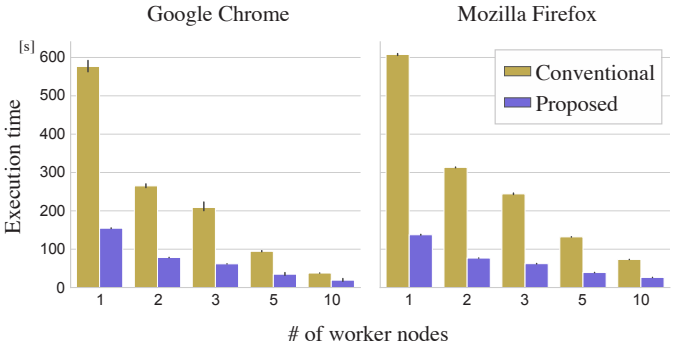

Fig. 3. Results of E_1: rainbow table generation

D. Results and Discussion

1) Rainbow Table Generation (E_1): Figure 3 shows the results of E_1. We conduct each experiment for 10 times and average the results. In this bar chart, the vertical axis represents the execution time of the entire job, and the horizontal axis represents the number of workers. The yellow-colored bar and the purple-colored bar shows the results of the conventional method using JavaScript and the proposed method (i.e., Madoop) using WebAssembly, respectively. The left figure shows the results in Chrome, and the right figure shows the results in Firefox. Let n be the number of workers. Figure 3 reveals that as n increases, the execution time decreases in inverse proportion. In both browsers, the proposed method takes less time than the conventional method. Focusing on the results of $n = 10$, Chrome completes the job in 19.2 [s] with the proposed method and 37.8 [s] with the conventional method. Firefox also finishes in 26.3 [s] with the proposed method and 73.2 [s] with the conventional method. These results show that the proposed method has a definite advantage for jobs with high complexity and small input data, such as E_1, and improves the runtime performance by about 49 to 64%.

It is also remarkable that the proposed method of $n = 5$ takes less execution time than the conventional method of $n = 10$. Figure 3 also shows that the performance appropriately scales according to the number of workers. For example, the execution time ratio of $n = 10$ to $n = 1$ is almost equivalent to the theoretical value, which is $1/10 = 0.1$.

2) Word Count (E_2): Figure 4 shows the results of E_2. Similarly to E_1, the execution time becomes shorter in inverse proportion to n. However, there are many results that the execution time of the proposed method, colored in purple, is longer than that of the conventional method, colored in yellow. When $n = 10$, Chrome executes the job in 53.6 [s] with the proposed method and 49.5 [s] with the conventional method. Firefox also executes in 54.0 [s] with the proposed method and 51.7 [s] with the conventional method. This shows that the proposed method takes 4 to 8% more execution time compared to the conventional method.

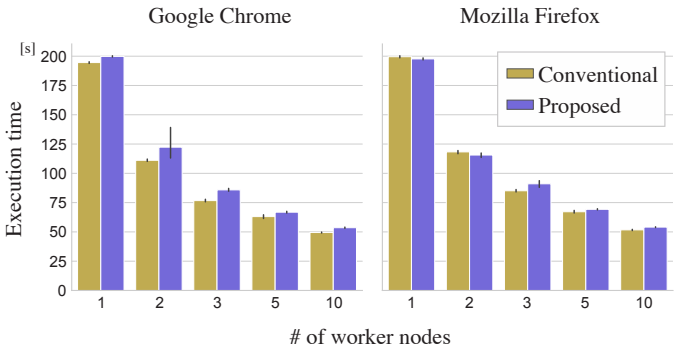

Fig. 4. Results of E_2: word count

This increase is caused by the overhead when web browser interprets and executes a WebAssembly binary. Compared to the execution of pure JavaScript programs, WebAssembly requires web browser to do extra preparations such as recompiling the binary for optimization, allocating memory space for WebAssembly stack machine and freeing the allocated memory space. Hence, if executing a low-complexity task with WebAssembly, the time increased by this overhead may overtake the time reduced by introducing WebAssembly. E_2 is a low-complexity job, so that the use of WebAssembly leads to slow down the computing.

V. Related Work

Gray Computing [2] is a well-designed and well-established BBVC framework using some modern web technologies. A fundamental open question of Gray Computing is to confirm feasibility and cost-effectivity of data processing using website visitors. In contrast to that, Madoop addresses two crucial BBVC challenges, runtime performance and ease of development. Therefore, our work can be used in combination with Gray Computing.

Popular browser vendors, such as Chrome, Firefox, and Safari, have been attempting to improve the performance of JavaScript engines. This attempt includes; Native Client (NaCl) [10] and WebAssembly [7]. These technologies share the same concept to execute low-level code on a web browser to avoid just-in-time compilation and runtime type inference. Our proposed Madoop addresses the performance problem by incorporating WebAssembly which is the latest web standard to enable near-native performance on a web browser.

VI. Conclusion

In this paper, we proposed a novel BBVC framework named Madoop and evaluated how much it improves execution performance compared to a traditional BBVC method. The results show that execution time improves about 50 to 64% for a compute-intensive job. However, it worsens about 4 to 8% for a data-intensive job. These results indicate that Madoop is better suited for high-complexity problems.

Madoop has some limitations. One limitation is that job requesters cannot know when their jobs will be accomplished. Madoop's main server follows HTTP's request and response model, so that its tasks are "pulled" from the main server by web browsers. There is also a risk of the task being killed before its completion. Thus, it is harder to estimate the remaining time to complete a job, compared to the traditional grids or clusters, whose tasks are "pushed" to slaves by a master node [1]. Another limitation is the less number of implementation languages for MapReduce jobs. Currently, there are not many languages which support WebAssembly as its compilation target. However, many languages, including Java, are under active development to support WebAssembly[5]. It is only a matter of time before developers can freely select the implementation language.

As future work, we plan to invent an effective scheduling algorithm which considers the properties of each worker node. Madoop currently distributes the same size tasks equally to all workers. However, there are many characteristics across the workers (i.e., web browsers) such as communication situation, processing performance, and dwell time on the page. These characteristics may lead to a severe performance slowdown because one worker could be a bottleneck for the whole progress. There need to be improvements in the scheduling algorithm in order to boost Madoop's performance more.

Acknowledgements

This work was supported by JSPS/MEXT KAKENHI Grant Number 16H02908 and 18H03222.

References

[1] T. Fabisiak and A. Danilecki, "Browser-based harnessing of voluntary computational power," *Foundations of Computing and Decision Sciences*, vol. 42, no. 1, pp. 3–42, 2017.

[2] Y. Pan, J. White, Y. Sun, and J. Gray, "Gray computing: A framework for computing with background javascript tasks," *IEEE Transactions on Software Engineering*, 2017.

[3] O. Nov, D. Anderson, and O. Arazy, "Volunteer computing: A model of the factors determining contribution to community-based scientific research," in *Proceedings of the 19th International Conference on World Wide Web*. ACM, 2010, pp. 741–750.

[4] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[5] J. Subhlok, J. M. Stichnoth, D. R. O'Hallaron, and T. Gross, "Exploiting task and data parallelism on a multicomputer," in *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 1993, pp. 13–22.

[6] J. J. Merelo Guervós, M. G. Valdez, P. Á. C. Valdivieso, P. García-Sánchez, P. de las Cuevas, and N. Rico, "Nodio, a javascript framework for volunteer-based evolutionary algorithms: first results," *CoRR*, vol. abs/1601.01607, 2016.

[7] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with webassembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2017, pp. 185–200.

[8] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, 2004, pp. 137–150.

[9] D. G. Goldstein, R. P. McAfee, and S. Suri, "The cost of annoying ads," in *Proceedings of the International Conference on World Wide Web*, 2013, pp. 459–470.

[10] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native client: A sandbox for portable, untrusted x86 native code," in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 79–93.

[5]https://github.com/appcypher/awesome-wasm-langs