S. Krajči (ed.): ITAT 2018 Proceedings, pp. 161–167
CEUR Workshop Proceedings Vol. 2203, ISSN 1613-0073, © 2018 Jakub Šiller and Jaroslav Kuchař

Framework for Distributed Computing on the Web

Jakub Šiller and Jaroslav Kuchař

Web Intelligence Research Group, Faculty of Information Technology
Czech Technical University, Thákurova 9, 160 00, Prague 6, Czech Republic
{sillejak|jaroslav.kuchar}@fit.cvut.cz

Abstract: This work is a brief summary of a master thesis that focuses on design and implementation of a framework that uses computers of website visitors as computing nodes through web browsers. It contains an analysis of the Web environment, summarization of previous approaches and projects, design and implementation of the framework. The work describes the solution of computing node failure, reaction to slow computing node, possibilities of controlling the load of the framework on a website visitor's computer, strategies for work distribution and security of the framework. At the end of the work, the experiment results and proposal of improvements are listed.

1 Introduction

Nowadays, web technologies and web services are an inseparable part of human life. We are using web browser for communication, entertainment, shopping and a many other activities. A lot of web users have powerful devices but they don't use their full power often. Thus, in the world, there is huge computing potential that is idle. Framework presented in this work is able to utilize this computing power.

In section 2 there is a summarization of previous works and current project in the field of distributed computation in web browsers. Analysis of web environment and framework requirements are content of section 3. Section 4 is focused on design of the framework. Finally, experiment results are presented in section 5.

2 Related works

Utilizing idle power via the Internet to create distributed computer is not a new idea. According to [1] the first project about this topic was *Great Internet Mersenne Prime Search* [2] that started in 1996. A few years later projects *SETI@Home* [3] that was focused on the Search for Extraterrestrial Intelligence and *Folding@home* [4] that was using computers of volunteers for medical research was launched. After that *BOINC (Berkeley Open Infrastructure for Network Computing)* [5] was created. It is probably the biggest and the best known platform for volunteer computing. All of mentioned projects are still alive. In order to join a computation in these projects a volunteer have to install some additional software.

Since 2007 several works related to utilizing computation power using web browser have been published. They were focused on various types of computing tasks. For instance projects [6], [7], [8] and [9] were using web browsers for simulated evolution. Work [10] was focused on image processing. Machine learning in web browsers was the topic of [11]. Authors of [12] created web search engine using web browsers. In [13] and [14] map reduce frameworks were presented. There were also works that were focused on creating general framework: [15], [16], [17]. Architecture of most of presented works was client-server.

Developers of commercial project *Computes* are trying to create distributed decentralized supercomputer from all kind of devices.

There are also several commercial tools for mining altcoins.

Published works are usually just proof of concept and authors don't deal with every aspect of systems. For instance security is often omitted. The main contribution of this work is to create complex framework that could be deployed.

3 Analysis

The web has several major characteristics. One of them is *diversity*. Web users are using various browsers of various versions. Each user has different device, different connection speed etc.

Another strong aspect of the Web is *dynamics*. Web technologies are still evolving and continuously new technologies are emerging. Also web browsing is very dynamic. According to [18] users often stay on a page just for 10-20 seconds.

The Web is *free and open*. Everyone can join and publish and consume data.

And the last but not least aspect is its' *enormous size*. [19] and [20] states that the Web has more than 3.7 billions users and this number is getting bigger every year.

These aspects of the Web have impact on the framework requirements. Dynamic browsing will cause frequent computing node failures. The framework have to be able to detect failure and solve the situation. The framework also should be able to work with computing nodes of different performance. Security mechanisms should be involved on server side and on client side as well. Framework also have to ensure correctness and reliability of tasks results.

4 Design

4.1 Computational model

The framework is focused on types of tasks in which server sends work to client, client processes the work and sends the result back to the server. There is no communication between clients and no communication between client and server regarding the computation but distributing works and receiving results. Framework user can define dividing and merging functions. In case that a task has too big data the framework will use dividing function in order to automatically recursively divide the data and create partial subtasks called works. Works are then distributed to the client. Results of works from clients are then merged by framework using merging function. Described model is shown in figure 1.

In order to compute a tasks there will be a lot of messages between clients and the server. Speed of communication depends on connection quality. It might be time consuming. Therefore, the framework is more suitable for computing intensive tasks rather than data intensive tasks.

4.2 Users

Users of the framework are divided into two groups. Users from the first group are creators of task prototypes. A task prototype consists of definition of code that should be executed in client and optional dividing and merging function. This group of users should know the framework - its' advantages, disadvantages and some technical details in order to create effective code for the framework.

The second group are common users. They use framework for computations. They don't need to know anything about the framework but the id of task prototype they want to use.

This division has several reasons. The first of them is security. We can assume that number of task prototypes creators will be much smaller than number of other users. Therefore it might be relatively easy to make sure that creators of task prototypes are trustworthy. And then we can assume that their code is probably trustworthy as well. Another reason for the division is quality of code. If there are users who are focusing on creating code for the framework there is high probability that the code will be efficient, without bugs and there will be no task prototypes for tasks that are not suitable for the framework. Moreover common users don't need to know anything about the framework.

4.3 Framework in a nutshell

Basic architecture of the framework is client-server. The server consists of two main parts - *ProgrammerServer* and *VolunteerServer*. *ProgrammerServer* is responsible for communication with users of the framework. *VolunteerServer* is the main part of the framework. It is responsible for communication with clients, processing tasks, distributing works to clients and processing results.

The framework utilizes replication and majority voting in order to ensure correctness and reliability of works results. User can for each task specify the replication factor. In each work object server holds information that indicates to how many more clients the work should be distributed in order to reach the replication factor. This information is in attribute *remaining*.

In order to compute a task a user sends data and id of task prototype to the server. Server insert the task to the task queue. The server holds collection of works that are currently being distributed to clients. If there is enough place for more works server prepare another task from the queue for distribution. Preparation of a task for distribution consists mainly of dividing task's data. When a client sends work request the server response with several works. Works that are returned to the client have attribute *remaining* greater then zero. After assigning a work to a client the attribute is decremented. Number of works that are returned to the client depends on strategy that is used. Strategies implemented in the framework are described in following section. When a client receives works it starts to process them. The client doesn't wait until all works are done but each result is sent back to the server as soon as it is available. This is shown in figure 3.

When the server received results from all works it merge them to the result of the task. The task's result is stored in database. When the user send request for the result to the server it responds with the result from the database.

When a client processes all assigned works it sends new work request to the server.

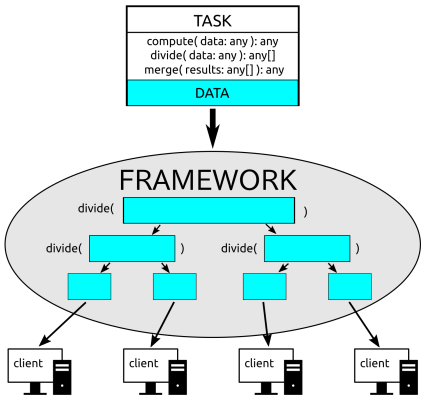
Strategies for work distribution There are several strategies in the framework for determining number of works that should be returned to a client.

The most easy one is a strategy that returns fixed number of works.

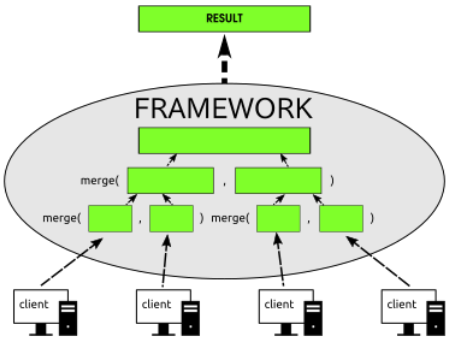
Another strategy is based on fixed sum of works sizes. The framework uses statistics to estimate maximum number of works so that sum of their sizes is less then configured threshold.

The most complex strategy uses time elapsed from start of a client session. According to [18] a distribution of session duration follows Weibull distribution with negative aging. That means that at the begging of a web page visit the probability that user will leave the page is very high and it decrease over time. Therefore this strategy increase number of works that are sent to the client accordingly to the session duration.

Modes of work distribution Work distribution can run in two modes. In the first mode data with full work code are sent to the client. In the second mode data along code identifier are sent. In this mode client have to make another request to the server in order to get the code. However, in this mode it is possible to cache the code in the



(a) Dividing and distributing phase



(b) Receiving and merging phase

Figure 1: Visual example of the computation model of the framework

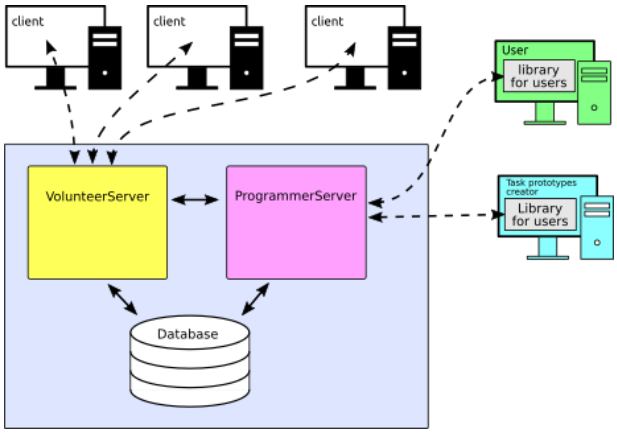


Figure 2: Base architecture of the framework.

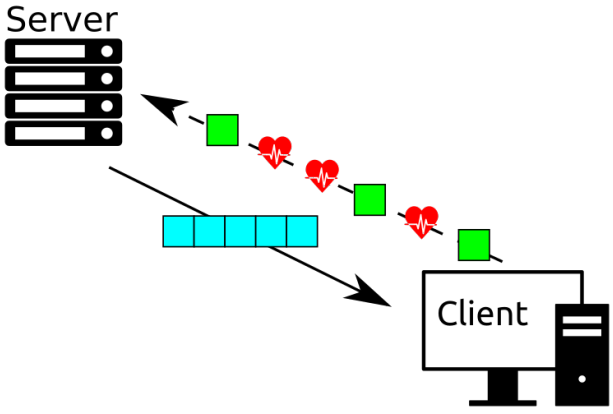


Figure 3: Simplified process of distributing works to the client and subsequent sending results back to the server. During computation of a work heartbeats are sent to the server.

browser and in intermediate network devices. Therefore the amount of data on wires can be decreased. This mode is efficient only if a client or a group of clients are performing tasks of a single or a few task prototypes. In other cases it might be inefficient because of more requests.

4.4 Computing node failure

In order to detect computing node failure the framework holds in each session object time of last access. It is updated with each request from the session. During computation a client code running in a session periodically sends empty request to the server - a heartbeat - so the server is informed the session is still alive as is shown in figure 3.

A server module *DeadSessionCollector* periodically collects dead sessions. A session is considered to be dead if time elapsed from the last access is higher than a configured threshold. When a session die the framework increments attribute *remaining* of all unprocessed works that were assigned to the session. So the work will be assigned again to some session.

4.5 Slow computing node

Because of different performance of web user devices, different connection speed and different utilization of the device by it's user it may happen that computation of a work would last on one client much longer than computation of the same work on other client. Thus, when some client is processing some work too long it may be efficient to assign the work again to another client.

The framework's module *LongRunningSessionCollector* periodically checks whether there is a slow client for some work. A client is considered too slow for a work if

the time elapsed since the work was assigned to the client is several times longer than average time of computation of the work on other clients. If a client is marked as slow for a work the framework increment attribute *remaining* of the work.

4.6 Client side

At client side the framework is using Web Worker technology [21]. A web browser create new thread for each Web Worker object. Therefore experience of browsing a web page should not be affected by the framework. The framework creates several Web Worker objects in which works are processed. When a client receive works it put them in a queue. The Web Worker objects are taking objects from the queue and processing them. As soon as a Web Worker object computes a work the result is sent to the server. This is shown in figure 4.

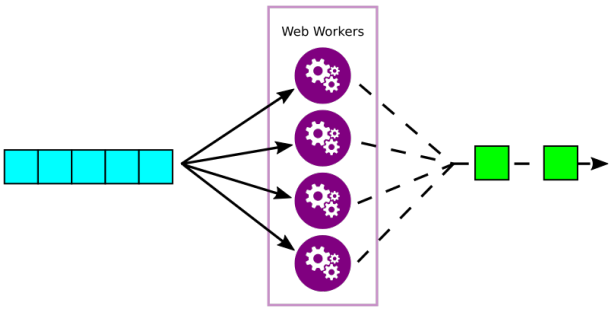


Figure 4: Illustration of processing works assigned to a client on multiple Web Worker objects

For computation of a work the framework is able to use new client-side web technologies asm.js [22] and WebAsembly [23] instead of JavaScript [24]. The creator of task prototype can implement working function in C++ which is then at the server compiled into asm.js and WebAssembly modules.

4.7 Controlling of client utilization

There are two ways how the framework is able to control stressing of client's device. The first one is controlling the number of Web Workers that are used. The second option is throttling. JavaScript code can not control stressing of CPU at particular time but it is possible to control stressing from long-term view. For instance if x seconds is processor fully stressed by the framework and then x seconds is idle we can say that the framework stressed CPU by 50% during $2x$ seconds. The configuration attribute *throttleFactor* holds information how many times the duration of the last computation the server has to wait before it could again assign a work to a client. In other words it is inverted value to the desired CPU usage.

4.8 Security

The framework contains several security mechanisms. The server accepts user request only with valid API keys. List of API keys is in configuration of the framework. If a request doesn't contain valid API key the server responds with HTTP code 403 forbidden.

The framework executes user code on server-side and also on client-side. The framework must ensure that the code will cause no harm to server or client's device. Therefore, user code is executed in a sandbox. At the server-side VM2 module [25] is used. At the client-side a sandbox is implemented as a white-list of allowed function and objects. Every other function or object is disallowed.

Communication between a client and the server is encrypted. This decrease probability that content of messages is changed by malicious third-party.

As was mentioned earlier in order to ensure correctness an reliability of results majority voting is involved. A user can specify replication factor but he or she should be aware that probability that data are correct is never 100%.

Securing data is complicated topic. The purpose of web browsers is to serve data to its user so the framework can not hide data from the user of the browser it is running in. Therefore the only way how to secure the data is computing on encrypted data. There are mathematical models that enables it. Some of them are described in [26]. There is no need to change the framework in order to compute on encrypted data - it is responsibility of task prototypes implementation. Computing on encrypted data can also ensure 100% probability that result is correct excluding bugs in the task prototype.

5 Experiments

Experiments were performed on 60 computers in classrooms at FIT CTU. Configuration of computers is in table 1. Test task was naive algorithm for computing determinant of a matrix. Maximum size of a matrix that was sent to clients was 11×11. If the matrix was bigger it was recursively divided to matrices of size 11×11.

Classroom	CPU Model	size of RAM
T9-350	Intel® Core™ i5-6500 CPU @ 3.20GHz	16 GB
T9-351	Intel® Core™ i5-3470 CPU @ 3.20GHz	8 GB
T9-303	Intel® Core™ i5-3470 CPU @ 3.20GHz	8 GB
T9-349	Intel® Core™ i5-4570S CPU @ 2.90GHz	8 GB
server	Intel® Core™i5-2410M CPU @ 2.30GHz	6 GB

Table 1: Configuration of test computers

Tests were performed in the mode in which just identifier of a code is sent to the client. Replication factor was set to 3. The framework was using all CPU cores of the client devices. Test environment refreshed test web page at a client after randomly chosen time from interval $[0, maxDuration]$ where *maxDuration* is parameter. Test Settings of earlier mentioned parameters are in table 2.

Parameter	Value
deadSessionCollector:	
interval	2 sec
deadTime	10 sec
LongRunningSessionCollector:	
interval	5 sec
factor	5
Other framework settings:	
throttleFactor	0
heartBeatInterval	5 sec
test environment settings:	
maxDuration	960 sec

Table 2: Test setting of mentioned parameters

The purpose of the first experiment was to test how would change computation time of the task from user point of view with increasing number of clients. In figure 5 we can see that with increasing number of clients computation time is decreasing. During tests in classrooms T9-350, T9-351 and T9-303 speeding up stops around number 30. That probably happened because computers in T9-350 were more powerful that others (shown in figure 7) so the framework considered the others to be slow and started to assign one work to more clients that was necessary. This is shown in figure 6. Also there was a mistake in the configuration. Interval of sending heartbeat was equal to the time after which a session was considered as dead. After some changes in configuration the computation becomes again a little bit faster.

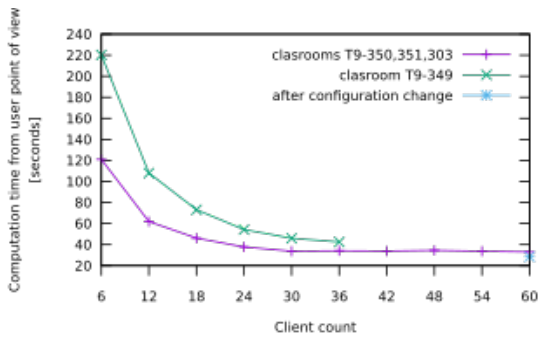


Figure 5: Influence of number of clients to computation time from user point of view.

The second experiment tests how size of a matrix would affect computation time from user point of view. It also compares computation time of the task using framework and computation time using local computation that was

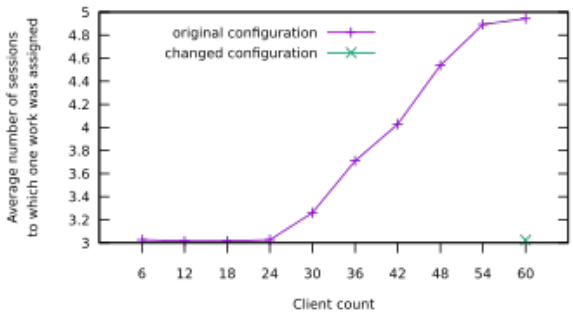


Figure 6: Influence of number of clients to average number of sessions to which one work was assigned

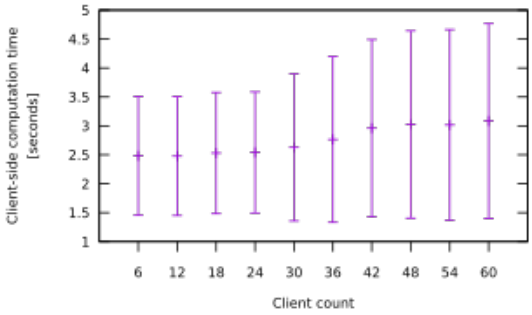


Figure 7: Statistics of a computation time of one matrix at client-side. For each number of client on axis X mean and mean $\pm$ standard deviation of computation time at client-side is shown

implemented in C++ using OpenMP [27] in order to utilize all CPU cores of a computer. Local computation was executed on the server's computer. In figure 8 we can see that for small matrices the local computation was more efficient but for big matrices it was more efficient to use the framework.

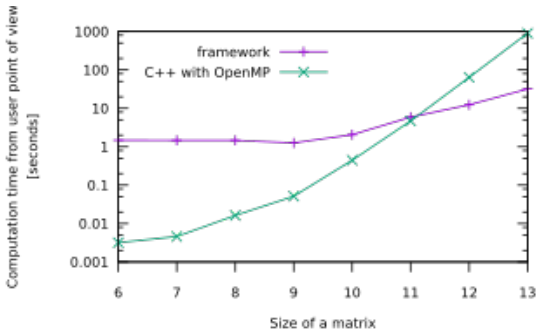


Figure 8: Influence of matrix size to computation time from user point of view.

The third experiment tests influence of *throttleFactor* on the computation time from user point of view. In figure 9 we can see that there is linear dependency. This test prove that influence of *throttleFactor* is expected and predictable.

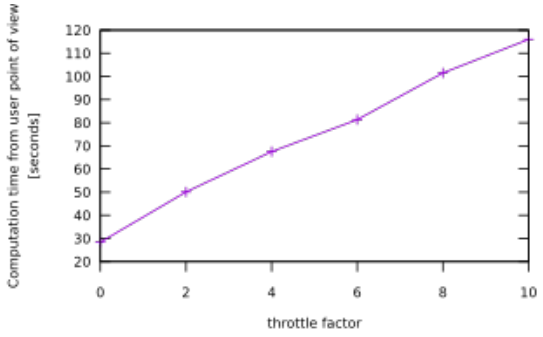


Figure 9: Influence of attribute *throttleFactor* to computation time from user point of view.

The last experiments tests how computation time from user point of view is changing when maximal session duration is changing. In figure 10 we can see that with increasing session duration computation time is decreasing as it was expected.

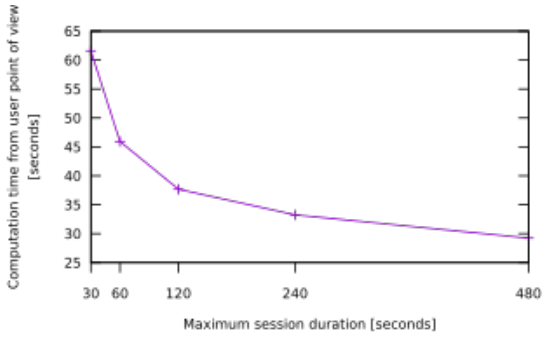


Figure 10: Influence of maximal session duration (attribute *maxDuration* of test environment) to computation time from user point of view.

6 Future work

Experiments have shown that framework is useful and it may be deployed. However there are still some limits and drawbacks of the implementation that should be solved. For instance database queries can be optimized or new strategies for distributing works can be implemented.

The framework can be also extended in several ways. For instance GUI and monitoring extension would be very practical. There might be client, user and web sites administration. Extensions of the framework may lead to computing platform with market that would act similar as application markets.

7 Motivation for joining computations

There can be several reasons for a web user to join the framework but probably the most likely situation is this: A web page could profit from involving it's visitors to the framework. So the web page could offer a discount of their services to visitors that allow joining to the framework.

Another use case could be in a company that have a lot of computers for it's employees. In this case a proxy could inject web pages with framework's client-side code and so create company's big distributed computer with minimum additional costs.

8 Conclusion

In this work a framework for distributed computation using web browsers is presented. It contains mechanisms that solve computing node failure and reaction to a slow computing node. It describes strategies for distributing works within clients and modes in which the distribution can be done. The work deals with controlling of client's device stressing. Security mechanisms are described as well. Experiment results that are presented have shown that the framework is useful and deployable. The future of the framework may be a computational platform with market of task prototypes and computation power.

Acknowledgments

This research was supported by Faculty of Informatics, Czech Technical University in Prague.

References

[1] Miller, D. M.: The Online Community Grid Volunteer Grid Computing with the Web Browser. Georgia Institute of Technology, 2008. Available on: `https://smartech.gatech.edu/handle/1853/33477`

[2] Mersenne Research, Inc.: Great Internet Mersenne Prime Search. [cit. 30.4.2018]. Available on: `https://www.mersenne.org`

[3] University of California : About SETI@home. [cit. 30.4.2018]. Available on: `https://setiathome.berkeley.edu/sah_about.php`

[4] Pande Lab: Folding@home. [cit. 30.4.2018]. Available on: `http://folding.stanford.edu`

[5] Anderson, D. P.: BOINC: a system for public-resource computing and storage. In *Fifth IEEE/ACM International Workshop on Grid Computing*, Nov 2004, ISSN 1550-5510, s. 4–10, doi:10.1109/GRID.2004.14.

[6] Merelo, J. J.; García, A. M.; Laredo, J. L. J.; aj.: Browser-based Distributed Evolutionary Computation: Performance and Scaling Behavior. In *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-698-1, s. 2851–2858, doi:10.1145/1274000.1274083. Available on: `http://doi.acm.org/10.1145/1274000.1274083`

[7] Klein, J.; Spector, L.: Unwitting Distributed Genetic Programming via Asynchronous JavaScript and XML. In *Proceedings of the 9th Annual Conference on Genetic and*

Evolutionary Computation, GECCO '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-697-4, s. 1628–1635, doi:10.1145/1276958.1277282. Available on: http://doi.acm.org/10.1145/1276958.1277282

[8] Merelo-Guervos, J. J.; Castillo, P. A.; Laredo, J. L. J.; aj.: Asynchronous distributed genetic algorithms with Javascript and JSON. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, June 2008, ISSN 1089-778X, s. 1372–1379, doi:10.1109/CEC.2008.4630973.

[9] Duda, J.; Dlubacz, W.: Distributed Evolutionary Computing System Based on Web Browsers with Javascript. In *Proceedings of the 11th International Conference on Applied Parallel and Scientific Computing*, PARA'12, Berlin, Heidelberg: Springer-Verlag, 2013, ISBN 978-3-642-36802-8, s. 183–191, doi:10.1007/978-3-642-36803-5_13. Available on: http://dx.doi.org/10.1007/978-3-642-36803-5_13

[10] Zorrilla, M.; Martin, A.; Tamayo, I.; aj.: Web Browser-Based Social Distributed Computing Platform Applied to Image Analysis. In *2013 International Conference on Cloud and Green Computing*, Sept 2013, s. 389–396, doi:10.1109/CGC.2013.68.

[11] Meeds, E.; Hendriks, R.; al Faraby, S.; aj.: MLitB: Machine Learning in the Browser. *CoRR*, ročník abs/1412.2432, 2014, 1412.2432. Available on: http://arxiv.org/abs/1412.2432

[12] Turek, W.; Nawarecki, E.; Dobrowolski, G.; aj.: WEB PAGES CONTENT ANALYSIS USING BROWSER-BASED VOLUNTEER COMPUTING. *Computer Science*, ročník 14, č. 2, 2013: str. 215, ISSN 2300-7036. Available on: https://journals.agh.edu.pl/csci/article/view/278

[13] Pan, Y.; White, J.; Sun, Y.; aj.: Gray Computing: An Analysis of Computing with Background JavaScript Tasks. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, ročník 1, May 2015, ISSN 0270-5257, s. 167–177, doi:10.1109/ICSE.2015.38.

[14] Ryza, S.; Wall, T.: MRJS: A JavaScript MapReduce Framework for Web Browsers. 2010. Available on: http://static.cs.brown.edu/courses/csci2950-u/f11/papers/mrjs.pdf

[15] Cushing, R.; Putra, G. H. H.; Koulouzis, S.; aj.: Distributed Computing on an Ensemble of Browsers. *IEEE Internet Computing*, ročník 17, č. 5, Sept 2013: s. 54–61, ISSN 1089-7801, doi:10.1109/MIC.2013.3.

[16] Wilkinson, S. R.; Almeida, J. S.: QMachine: commodity supercomputing in web browsers. *BMC Bioinformatics*, ročník 15, č. 1, Jun 2014: str. 176, ISSN 1471-2105, doi:10.1186/1471-2105-15-176. Available on: https://doi.org/10.1186/1471-2105-15-176

[17] Fabisiak, T.; Danilecki, A.: Browser-based Harnessing of Voluntary Computational Power. ročník 42, 03 2017. Available on: https://www.degruyter.com/downloadpdf/j/fcds.2017.42.issue-1/fcds-2017-0001/fcds-2017-0001.pdf

[18] Nielsen, J.: How Long Do Users Stay on Web Pages? [cit. 30.4.2018]. Available on: https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/

[19] Liedke, L.: 100+ Internet Stats and Facts for 2018. [cit. 30.4.2018]. Available on: https://www.websitehostingrating.com/internet-statistics-facts-2018/

[20] Stevens, J.: Internet Stats & Facts for 2017. [cit. 30.4.2018]. Available on: https://hostingfacts.com/internet-facts-stats-2016/

[21] Mozilla and individual contributors: MDN web docs: Using Web Workers. [cit. 30.4.2018]. Available on: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

[22] Herman, D.; Wagner, L.; Zakai, A.: asm.js: Working Draft. [cit. 30.4.2018]. Available on: http://asmjs.org/spec/latest/

[23] WebAssembly. [cit. 30.4.2018]. Available on: http://webassembly.org/

[24] Mozilla and individual contributors: MDN web docs: JavaScript reference. [cit. 30.4.2018]. Available on: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

[25] Šimek, P.: VM2. [cit. 30.4.2018]. Available on: https://www.npmjs.com/package/vm2

[26] Vaikuntanathan, V.: How to Compute on Encrypted Data. In *Progress in Cryptology - INDOCRYPT 2012*, editace S. Galbraith; M. Nandi, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-34931-7, s. 1–15.

[27] OpenMP ARB: OpenMP. [cit. 30.4.2018]. Available on: http://www.openmp.org/