Computer Science • 14 (1) 2013 http://dx.doi.org/10.7494/csci.2013.14.1.143

Roman Dębski
Tomasz Krupa
Przemysław Majewski

ComcuteJS: A WEB BROWSER BASED PLATFORM FOR LARGE-SCALE COMPUTATIONS

Abstract *The paper presents a new, cost effective, volunteer computing based platform. It utilizes volunteers' web browsers as computational nodes. The computational tasks are delegated to the browsers and executed in the background (independently of any user interface scripts) making use of the HTML5 web workers technology. The capabilities of the platform have been proved by experiments performed in a wide range of numbers of computational nodes (1–400).*

Keywords volunteer computing, parallel computations, large-scale computations, augmented cloud computing

1. Introduction

For many years, scientists have been solving large-scale[1] computational problems with the use of supercomputers or, more recently, computer clusters or grids. These computational platforms are still the best options in many cases, but they have their limitations (e.g. related to scalability). In such situations one can utilize the concept of *Augmented Cloud*[6] or, in general, a web browser based volunteer computing.

In this paper a new, lightweight and cost effective approach to building a highly scalable computing platform is presented. It utilizes the concept of web browser based volunteer computing and is based on some of the HTML5 technologies[2].

In the first section an overview of the platforms used for large-scale computations is presented. It also includes a brief description of the web browser based volunteer computing concept. The main part of the article starts with an overview of the platform architecture. Next, the main use case description and the platform implementation details are presented. The platform capabilities discussion, demonstrated on an example computation[3], constitute the final part of the paper.

2. Platforms for large-scale computations

Many large-scale computations are performed using supercomputers. This approach has many advantages but, unfortunately, also one big disadvantage – it has always been expensive. That is why computer clusters (as a cost-effective[4] supercomputer substitute) started being used (for example [3]). Yet, sometimes the total cost (and effort) of building a computer cluster can also be too high (e.g. when thousands of computers are needed to perform a computation effectively). In such cases the ideas of *Grid Computing* [9], *Volunteer Computing* [1], *Cloud Computing* [10], [7], [13], [11], [2] or *Augmented Cloud Computing* [6], [4] can be utilized.

The platform presented in this paper (i.e. *ComcuteJS*) utilizes the idea of the web browser based volunteer computing. A brief description of this concept is presented in the next two paragraphs.

Volunteer Computing. *Volunteer Computing*[5] is a type of distributed computing in which some of the computational resources come from the number of nodes dynamically connecting to a network.

It is worth mentioning two Volunteer Computing projects:

- *Great Internet Mersenne Prime Search*[6], which was the first project (started in 1996) utilizing the idea of Volunteer Computing,

[1]The term 'large-scale problem' is used here in a broad sense, referring to all problems considered difficult for all known solution methods

[2]mainly on web workers as a way of introducing multi-threading into a web browser

[3]it was a search for prime numbers in range $\langle 2, 10^7 \rangle$

[4]mainly because it is based on commodity hardware

[5]or a similar idea – *Sideband Computing* [14]

[6]`http://mersenne.org/`

- *SETI@Home*[7] – searching for signs of extra-terrestrial intelligence (launched in 1999), which is by far the most famous one.

Volunteer Computing projects may be implemented using several middlewares, such as Berkeley Open Infrastructure for Network Computing[8] (BOINC) (open source, base of SETI@Home), Xgrid[9] (a proprietary software prepared by and for Apple) or Grid MP[10] (a commercial product).

Web Browser Based Volunteer Computing. Unfortunately, the existing Volunteer Computing environments are usually dedicated to one project/application[11] and, what is very often unwelcome nowadays, require some software to be installed on each volunteer's PC. The idea of utilizing a web browser as an environment for executing a volunteer's computational task addresses the last issue. This idea is not new. In the past it was implemented by using Java Applets as computational workers (e.g. [12], [5]). In the next sections the new approach – based on JavaScript – is presented.

3. ComcuteJS: the architecture and processing

The architecture of the ComcuteJS platform is based on the Comcute Reference Architecture[12] shown in Figure 1.

It is comprised of the following three layers:

Comcute-Core – can consist of many nodes; it is responsible for controlling computations via the system console (starting, stopping, monitoring), data partitioning and aggregating of results; the system console offers two interfaces: one for a system administrator (mainly user accounts management) and the other one for programmers,

Computational Task Dispatcher(s) – can consist of many nodes, grouped in pools (each pool is controlled by one node from the core layer); it plays the role of the Core gateway (it can also be seen as a layer of communication servers),

(Web Browser Based) Computational Workers – responsible for executing the computational tasks.

A programmer's perspective. All *computational jobs* in the system are defined in *projects* and stored in a database. Each project (computational job) consists of a JavaScript code and input data. Jobs are divided into *tasks* which are the computation units in the system. In case of a *SPMD* computation (Single Program, Multiple Data), each task comprises the same code and a subset of the input data (the subsets are created by a data partitioner in the *job preprocessing* step shown in Fig. 2).

[7] `http://setiathome.berkeley.edu/`

[8] `http://boinc.berkeley.edu/`

[9] `http://www.apple.com/pl/server/macosx/technology/xgrid.html`

[10] `http://www.univa.com/`

[11] And so, it cannot be considered even as a quasi-general purpose platform.

[12] the Comcute Reference Architecture has been defined in order to provide a base for a family of volunteer-based platforms for large-scale computations

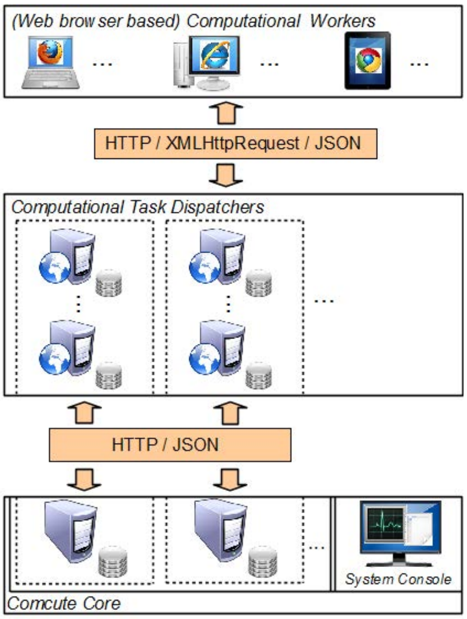
(Web browser based) Computational Workers
HTTP / XMLHttpRequest / JSON
Computational Task Dispatchers
HTTP / JSON
System Console
Comcute Core

Figure 1. Comcute Reference Architecture.

The main processing in the system is performed in two simultaneous loops repeated as long as there are some tasks to be computed. In the first loop (see Fig. 2), each dispatcher downloads computational tasks (as JSON messages) to its local task queue and sends partial results to the master node, which aggregates them and constantly monitors the computation status.

In the second loop, each web worker (running in a volunteer's browser) downloads a task (as a JSON message) from a dispatcher, performs the task and uploads the result to the dispatcher.

4. ComcuteJS: implementation details

ComcuteJS was created with the use of ComcutePDK (see the next paragraph), which is a member of the three Software Development Kits (SDK) family intended to support building web browser based platforms for large-scale computations. The family of SDKs (called *Comcute Environment*) shares the same programming model and assumes the same architecture of the underlying run-time platform (see *Comcute Reference Architecture* in Fig. 1). These three SDKs are called: *ComcuteJDK*, *ComcutePDK* and *ComcuteCDK* (see Fig. 3). Each SDK is based on a different technology and is dedicated to different kinds of systems (differing for instance in scalability and a deployment cost).

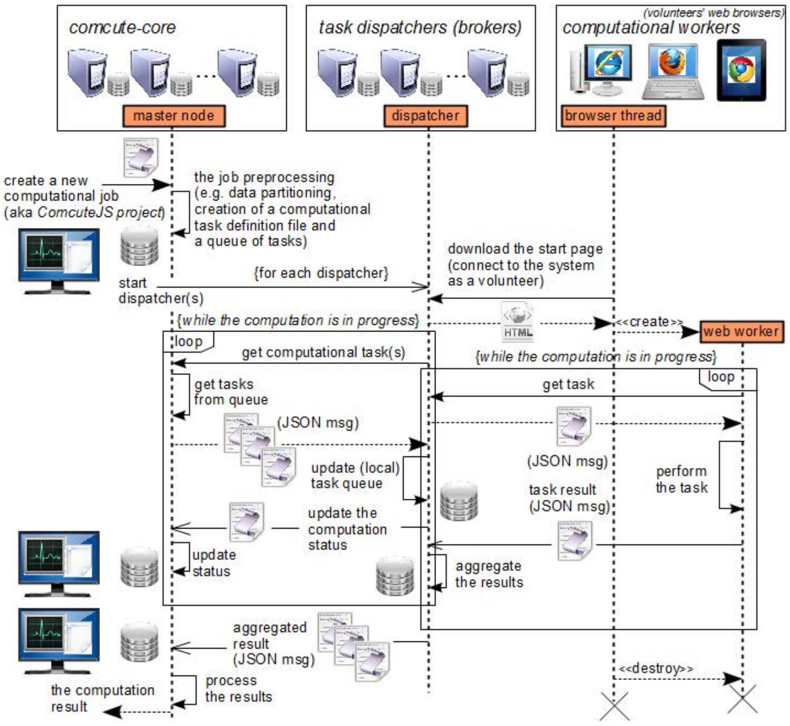
comcute-core
master node
task dispatchers (brokers)
dispatcher
(volunteers' web browsers)
computational workers
browser thread
create a new computational job (aka ComcuteJS project)
the job preprocessing (e.g. data partitioning, creation of a computational task definition file and a queue of tasks)
start dispatcher(s)
{for each dispatcher}
download the start page (connect to the system as a volunteer)
{while the computation is in progress}
HTML
<<create>>
web worker
loop
get computational task(s)
{while the computation is in progress}
loop
get tasks from queue
get task
(JSON msg)
update (local) task queue
(JSON msg)
perform the task
task result (JSON msg)
update the computation status
update status
aggregate the results
aggregated result (JSON msg)
<<destroy>>
process the results
the computation result

Figure 2. *ComcuteJS* main use case: performing a computational task.

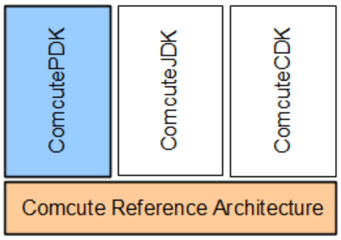

Figure 3. ComcutePDK as a part of Comcute Environment.

ComcutePDK. This SDK is intended to support building volunteer-based computational platforms which should be:

- lightweight (regarding the core and the dispatchers),
- easy to deploy and administer,
- inexpensive in usage

and are based on the following technologies:

- Apache HTTP Server + PHP + MySQL (the comcute back-end),
- HTML5/WebWorkers (web-based computational workers).

5. Experimental results

In order to demonstrate the platform capabilities the prime numbers search problem in the interval $\langle 2, 10^7 \rangle$ was solved[13]. The infrastructure used to perform the computation consisted of one comcute-core server, nine task dispatchers and forty computational nodes[14] (PCs, Windows 7), each running from one to ten web browsers[15] (in each computation all forty nodes were used, so one browser means $40 \times 1 = 40$ computational workers, two means $40 \times 2 = 80$ computational workers and so on). The solution was based on domain (data) decomposition – the interval was divided into 10^4 sub-intervals (each having 10^3 numbers).

To conduct an analysis of the parallel computation, three classical metrics were calculated: speedup S, efficiency E and serial fraction (Karp–Flatt metric [8]) f. They are defined as follows:

- **speedup**:
$$S(p,n) = \frac{T^*(1,n)}{T(p,n)}$$
- **efficiency**:
$$E(p,n) = \frac{S(p,n)}{p}$$
- **serial fraction**:
$$f(p,n) = \frac{\frac{1}{S(p,n)} - \frac{1}{p}}{1 - \frac{1}{p}}$$

where:

n – the problem size,

p – the number of processors,

$T^*(1,n)$ – the execution time of the sequential algorithm,

$T(p,n)$ – the execution time of the parallel algorithm with p processors.

There were three series of tests performed: for one, five and nine task dispatchers. In each series 10 measurements of the corresponding execution times were taken: for 40, 80... and 400 computational workers. Next, these measurements were used to calculate the aforementioned metrics. They are shown in Figures 4, 5 and 6. One can notice that the parallel solution of the prime numbers search problem is relatively efficient, and (surprisingly) the best solution was achieved in the configuration with only one dispatcher.

[13]because the experiment was just a "demonstrator of any large-scale parallel computation", a very simple and ineffective (deliberately!) algorithm was chosen (see Listing 1.); for $n = 10^7$ the execution time of the sequential realization ($p = 1$) was equal to 9480 s

[14]the comcute-core server and task dispatchers were located in Gdansk (the office of Fido Intelligence), whilst all the computational nodes were located in Cracow (Department of Computer Science, AGH University of Science and Technology)

[15]Google Chrome

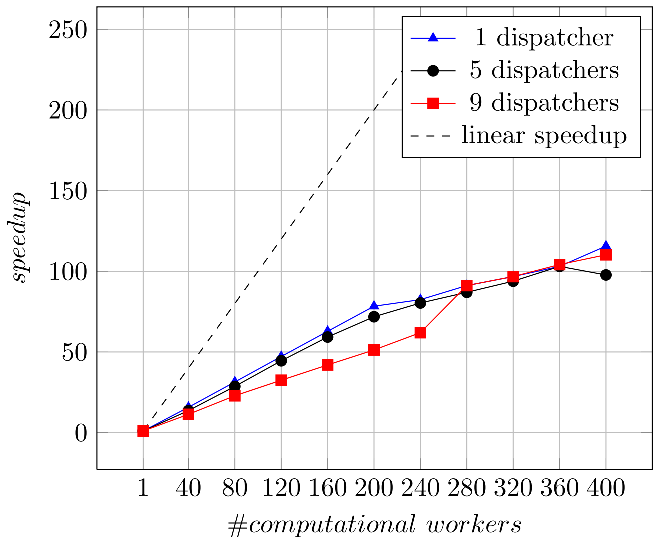

Figure 4. Speedup as a function of the number of computational workers for one, five and nine computational task dispatchers.

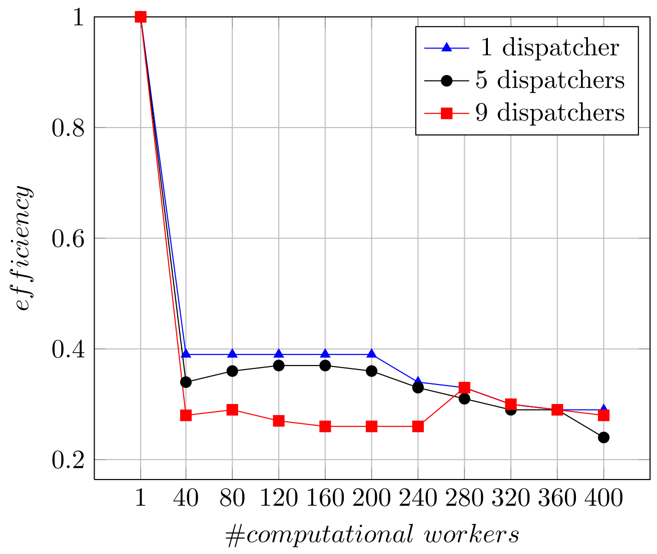

Figure 5. Efficiency as a function of the number of computational workers for one, five and nine computational task dispatchers.

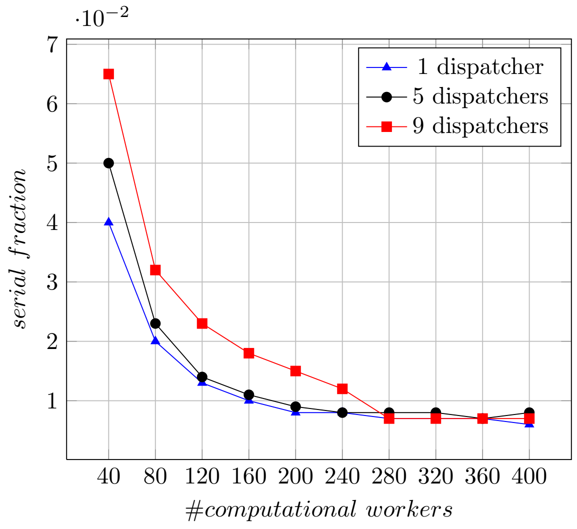

Figure 6. Serial fraction as a function of the number of computational workers for one, five and nine computational task dispatchers.

6. Conclusion

For many years, scientists have been performing large-scale computations with the use of supercomputers, computer clusters or grids. Recently, yet another option – web browser based computational platforms – has become popular. These new platforms can offer an extensive computing power and good scalability while remaining very cost-effective.

In this paper such a platform (called *ComcuteJS*) has been presented. It utilizes the concept of web browser based volunteer computing and is based on some of the HTML5 technologies (web workers). The capabilities of the platform have been demonstrated by execution of an example computation (prime numbers parallel search). The classic performance characteristics i.e. speedup, efficiency and serial fraction, presented in the final part of the article, proved that this approach can be effective.

It should be noted, however, that the current implementation is most appropriate for coarse granular problems, i.e. in which the computation-to-communication ratio is relatively high. Examples of such problems can be an evolutionary algorithm with a simulation-based evaluation of each individual or a *MapReduce*-like computation with a time-consuming mapping phase. Both the simulation-based evaluation and the mapping task can be delegated to the volunteers' web browsers.

Another important note is that the current implementation is only a proof of concept prototype and, in consequence, it does not address many issues which are often critical in distributed systems (e.g. the access control or secure communication).

```
function findPrimes(from, to) {
  var fBig = parseInt(from);
  var tBig = parseInt(to);
  tBig = tBig + fBig;
  var result="";
  var prime = true;

  while (tBig > fBig) {
        prime = true;
        if(fBig%2==0){
            prime = false;
        }
        if (prime){
            var tmp = fBig / 2;
            tmp = Math.floor(tmp);
            var i = 2;

            while (tmp > i) {
                if (fBig % i == 0) {
                    prime = false;
                    break;
                }
                i++;
            }
        }
        if (prime)
            result += fBig.toString()+ ' ';
        fBig++;
    }
    return result;
}
```

Listing 1: Computational worker's task (JavaScript)

In the near future the authors plan to conduct broader experiments with different classes of computational problems.

Acknowledgements

The work presented in this paper was partially supported by the Polish National Center of Research and Development grant No. 0108/R/T00/2010/11.

References

[1] Anderson D. P., Korpela E., Walton R.: High-performance task distribution for volunteer computing. In *Proc. of the First International Conference on e-Science and Grid Computing*, 2005.

[2] Armbrust M., *et al.*: *Above the clouds: A Berkeley View of Cloud Computing.* Technical Report, UC Berkeley, 2009.

[3] Bader D. A., Pennington R.: Cluster computing: Applications. *The International Journal of High Performance Computing Applications*, 15(2):181–185, May 2001.

[4] Byrski A., Dębski R., Kisiel-Dorohinicki M.: Agent-based computing in augmented cloud environment. *International Journal of Computer Systems Science and Engineering*, (1):7–18, 2012.

[5] Czerwiński B., Dębski R., Piętak K.: Distributed agent-based platform for large-scale evolutionary computations. In *Proc. of 5-th Int. Conf. on Complex Intelligent and Software Intensive Systems. IEEE Press*, 2011.

[6] Dębski R., Byrski A., Kisiel-Dorohinicki M.: Towards an agent-based augmented cloud. *Journal of Telecommunications and Information Technology*, (1):16–22, 2012.

[7] Jackson K., *et al.*: Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom'10*, 2010.

[8] Karp, Alan H., Flatt, Horace P.: Measuring parallel processor performance. *Communication of the ACM 33 (5)*, pp. 539–543, 1990.

[9] Kesselman C., Foster I.: *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, 1998.

[10] Mell P., Grance T.: *The nist definition of cloud computing (draft).* Technical report, National Institute of Standards and Technology, 2011.

[11] Napper J., Bientinesi P.: Can cloud computing reach the top500. In *UCHPC-MAW'09*, 2009.

[12] Sarmenta L. F. G., Hirano S.: Bayanihan: Building and studying web-based volunteer computing systems using java. *Future Generation Computer Systems*, 15:675–686, 1999.

[13] Vecchiola C., Pandey S., Buyya R.: High-performance cloud computing: A view of scientific applications. In *ISPAN 2009*, pp. 4–16. IEEE Computer Society, 2009.

[14] Xu Y.: Global sideband service distributed computing method. In *Proc. of the International Conference on Communication Networks and Distributed System Modeling and Simulation (CNDS98)*, 1998.

Affiliations

Roman Dębski
AGH University of Science and Technology, Krakow, Poland, roman.j.debski@gmail.com

Tomasz Krupa
Fido Intelligence, Gdansk, Poland, tkrupa@fidointelligence.com

Przemysław Majewski
Fido Intelligence, Gdansk, Poland, pmajewski@fidointelligence.com

Received: 27.06.2012
Revised: 19.10.2012
Accepted: 3.12.2012