Computer Assisted Methods in Engineering and Science, **24**: 17–40, 2017.

Volunteer computing in a scalable lightweight web-based environment

Pawel Chorazyk, Aleksander Byrski, Kamil Pietak,
Marek Kisiel-Dorohinicki, Wojciech Turek
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: pawel.chorazyk@joegreen.pl, olekb@agh.edu.pl, kpietak@agh.edu.pl, doroh@agh.edu.pl, wojciech.turek@agh.edu.pl

Volunteer computing is a very appealing way of utilizing vast available resources in efficient way. However, the current platforms that support such computing style are either difficult to use or not available at all, as a results of finished scientific projects, for example. In this paper, a novel lightweight volunteer computing platform is presented and thoroughly tested in an artificial environment of a commercially available computing cloud using two computing-related tasks and one web-crawling-related task.

Keywords: volunteer computing, distributed computing, metaheuristic computing, Javascript.

1. Introduction

When looking for efficient usage of computing power in the available infrastructure, one can consider not only utilizing multi- and many-core systems, supercomputing facilities, grids, and clouds, but also employing a specific computing infrastructure inspired by the socio-philosophical concept of volunteering.

Volunteer computing is a type of distributed computing where people (called volunteers) donate the computing resources of their devices to chosen projects. Thanks to volunteer computing, various research projects can perform large-scale computations without spending funds on creating or renting a computing infrastructure. This idea is also known as *public resource computing* [7, 11]. Sometimes, it is also referred to as *desktop grid computing* [25, 26], which seems to be a broader term.

A volunteer computing platform is a middleware that connects volunteers with people creating computational tasks. The main responsibility of the volunteer computing platforms is scheduling tasks to the volunteer devices. This has to be done in a way that fully utilizes the donated computing power and provides results to project owners in a timely manner.

One of the most popular volunteer computing systems is BOINC (Berkeley Open Infrastructure for Network Computing). Although it is already ten years old, it is still able to follow the current trends and adjust to the needs of volunteers. In addition BOINC supports GPGPU (General Purpose Graphical Processing Unit). In December 2015, nearly 250 000 volunteers were active, and the average computing power of the whole platform was around 158 000 TeraFLOPS [1] (which is a few times better than the performance of best supercomputer in the Top500 list [41]).

More recent and openly available volunteer computing platforms are either not mature enough, have lost community and support, or have even closed altogether. Moreover, while browsing through their features and requirements for running, it can be noticed that they are rarely easy-to-install or easy-to-use and are often operating system-dependent (to name just a few of their drawbacks).

Therefore, the need has arisen for a novel volunteer computing platform, free of such drawbacks, flexible, reliable, as portable as possible and without the need of installing sophisticated components on each volunteer's operational system. Additionally, it would be great if it only required a web browser to participate in the volunteer-computing tasks.

This paper presents a significantly extended version of a material presented in [16] in which only experiments consisting of Wikipedia crawling and a brief description of the whole infrastructure were presented. In this paper, we provide insightful details on the newly constructed computing platform supported by a broad state-of-the-art background review.

In the next section, the concept of volunteer computing is discussed, followed by a description of the most popular volunteer platforms (with an emphasis on web-based platforms). Next, the concept and implementation of the details of the presented platform (named Edward) are given. In Sec. 5, experimental verification shows the efficiency of the platform for three selected computing (or web-crawling) cases. Finally, Sec. 6 concludes this paper.

2. Volunteer computing concept

Volunteer computing is a type of distributed computing where people (called volunteers) donate the computing resources of their devices to chosen projects. Thanks to volunteer computing, various research projects can perform large-scale computations without spending additional funds on creating or renting computing infrastructure. The idea is also known as *public resource computing* [7, 11]. Sometimes, it is also referred to as *desktop grid computing* [25, 26], which seems to be a broader term.

A volunteer computing platform is a middleware that connects volunteers and people to create computational tasks. The main responsibility of volunteer computing platforms is to schedule tasks to volunteer devices. This must be done in a way that fully utilizes the donated computing power and provides results to project owners in a timely manner.

A simplified schema of the volunteer computing concept is shown in Fig. 1. The volunteers devices download tasks, perform computations, and send their results back. Project owners send batches of tasks to the platform servers and can download their results when they are ready. If a large number of devices are used to perform computations, high level of parallelism can significantly decrease the total computation time.

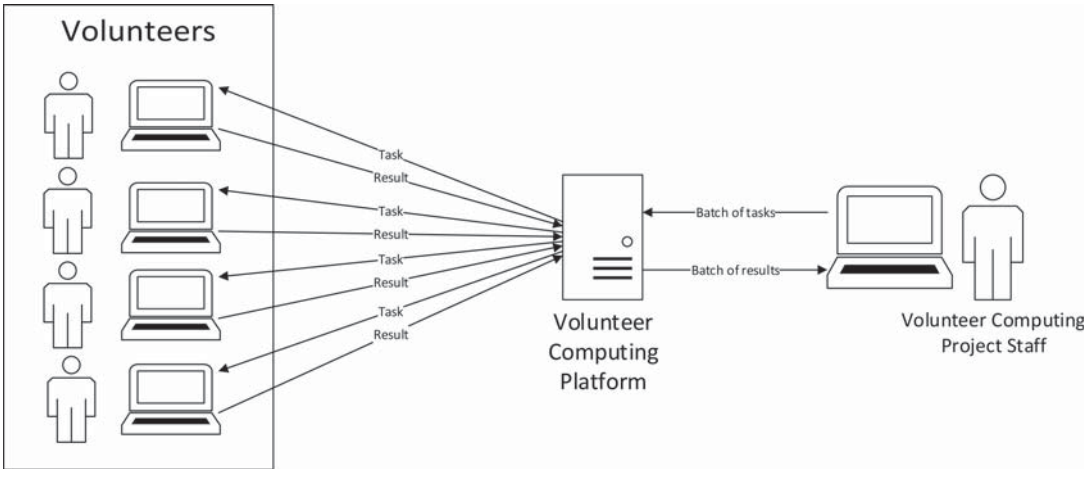


Fig. 1. Simplified scheme of volunteer computing.

In practice, architectures of large volunteer computing platforms are usually more complicated. Middleware platforms are often distributed among multiple physical servers (architectures of Folding@Home [12] or BOINC [10] are the examples). Some projects also use direct connections between

volunteers to efficiently distribute the data needed to perform tasks (for example, Bittorrent protocol is used [7, 25]).

Volunteer computing is not always a good choice for every computing-intensive project. According to the creators of SETI@home, projects that aspire to use volunteer computing should have the following features [9]:

- Tasks should be independent. There should be no need to synchronize computations on the communicated data between different tasks, so that they can be easily executed in parallel.
- Tasks should have a high computing-to-data ratio. Volunteer computing is most efficient when there is no need to transfer a lot of data yet the computations take a lot of time. If the input data or results are large, the time needed to send them can nullify the gain of faster computations.
- Project should tolerate errors. Tasks are executed on possibly insecure or unverified devices. Even though many methods of verifying results can be used ([39]), projects should not rely on the 100% accuracy results.
- Project should attract volunteers. Using volunteer computing makes sense only when there are many volunteers donating their computing resources, so a project should have features that will persuade people to take part in it. Volunteer computing is often used by scientific or humanitarian projects.

3. Existing volunteer computing platforms

In this section, the short history and existing platforms used for volunteer computing purposes are described in order to present the background for the current research.

3.1. History

The Great Internet Mersenne Prime Search (GIMPS) founded in 1996 was probably the first project that used volunteer computing (even though that term did not exist as of yet). The project's main goal was to find large Mersenne prime numbers using the Lucas–Lehmer primality test. At the beginning, the assignments and results were exchanged using e-mails; however, as the project grew, a more-efficient system was needed, so the PrimeNet software platform was introduced. It is worth noting that GIMPS is still active today and has already led to the discovery of 15 new Mersenne prime numbers [33].

The distributed.net project was founded in 1997 in response to the RSA Secret-Key Challenge contest organized by the RSA Laboratories. In this project, contestants were asked to break different types of encryption algorithms [21]. Volunteers who taking part in distributed.net were contest committed their computing resources to find cryptographic keys used in this contest. Up until now, the project has managed to find two of them: one in October 1997 (after only 250 days), and another in 2002 [22]. Nowadays, a significant part of the assignments is completed by using GPUs (Graphical Processing Units) with technologies like CUDA (Computed Unified Device Architecture), ATI Stream, and OpenCL (Open Computing Language) [24]. A volunteer whose device finds the key is given a monetary reward, which is an interesting way of attracting volunteers [23].

The term "volunteer computing" was probably used for the first time in 1998 by the creator of the Bayanihan volunteer-computing platform in the paper titled "Bayanihan: Web – based volunteer computing using Java" [38]. Besides describing altruistic volunteer computing, forced and paid volunteer computing were also described in this article.

The SETI@home (Search for Extra-Terrestrial Intelligence at Home) project has played a significant role in popularizing the idea of volunteer computing. The project uses volunteer computing to analyze signals from the Arecibo radio telescope in Puerto Rico to find signs of extra terrestrial

life. According to the official history [43], the idea of SETI@home was created in 1994. However, it took more than five years to collect the needed funds and implement it. The project officially started in 1999 at the UC Berkeley Space Sciences Lab. During the first week of existence, around 200 000 people downloaded and ran the client to become volunteers. In 2002, the official article describing the project and titled "Seti@home: An experiment in public-resource computing" [9] was published.

Founders of SETI@home have introduced many features that have helped to attract new volunteers and keep them engaged. Volunteers are rewarded for the resources they committed to the project with virtual points called "credits." Various statistics and leaderboards are generated based on the number of credits received. Moreover, volunteers can create and join teams to participate in group competition to collect the most credits, which further encourages them to tell their friends about the project. To reach even more people and help them get started, the project's home page was translated to more than 30 languages, and the *sci.astro.seti* usenet group was created. Client software running on volunteer's computers can also act as a screensaver showing visualizations of the currently-performed computations, which also adds to SETI@home's popularity. Thanks to all those features, SETI@home managed to attract many volunteers and persuaded them to commit their computing resources. Consequently, the idea of volunteer computing was popularized and SETI@home has even appeared in the mass media.

It quickly became clear that handling the project's infrastructure took too much time, so David Anderson (SETI@home's leader) joined United Devices in 2000 to create a better software solution. This did not work out, so David left United Devices in 2002 to start working on the BOINC platform (which later became an important middleware platform in the world of volunteer computing) [43].

The Folding@home project was founded in 2000 with the main goal of simulating protein-folding processes that could help in better understanding of diseases such as Alzheimer's or Parkinson's disease. Long and detailed simulations, such as those in the Folding@home project, require a lot of computing resources – these are provided by volunteers. The article describing the project titled "Folding@home: lessons from eight years of volunteer distributed computing" [12] was published in 2009. Similar to SETI@home, Folding@home provides many features to help attract and engage volunteers, such as visualizing simulations, giving virtual credit rewards, and creating individual and group leaderboards based on them. The article emphasises that the ability to create teams (collecting virtual credits in teams) helps in the recruitment of new volunteers, as information about Folding@home spreads by word of mouth and viral marketing. Some teams have even created their own websites and conduct their own marketing initiatives to attract more members. The results obtained on Folding@home are often presented in scientific publications. As of 2015, more than 100 had been published [35], which clearly shows the benefits that the project generates.

In 2003, IBM (along with United Devices and Accelrys) created the Smallpox Research Grid Project, which used volunteer computing to analyze potential drug candidates to develop anti-smallpox drugs [28]. The project turned out to be a great success – 44 potential treatments were identified. IBM decided to expand the initiative. In 2004, they announced the World Community Grid (WCG), a volunteer computing platform supporting research projects with humanitarian goals [17, 29]. In the beginning, WCG used the proprietary Grid MP middleware created by United Devices. This Grid MP did not support Linux, and so between 2005 and 2007, WCG migrated to the BOINC open-source platform [5, 6], where it remains active till now as an "umbrella project" (a project that has multiple sub-projects beneath it).

BOINC is a generic volunteer computing platform that can be used to create volunteer computing projects. It helps developers by solving many common back-end issues such as task scheduling, result validating, or dealing with security concerns. The platform provides a consistent and easy-to-use interface where volunteers can choose the projects they want to support and decide how much of their computing resources are to be committed to each of them. The client application is greatly configurable, and volunteers can set up constraints regarding how much computing power, RAM, disk, or Internet bandwidth can be used by the platform. This assures them that taking part in the computations will not interfere with their own activities. In addition, this configuration

can be synchronized between all of the devices owned by the same volunteer. BOINC provides a unified volunteer-reward system across its projects. Virtual credits are given to volunteers, and their total number is published as XML files, which are then used by third party websites to create, publish, and visualize leaderboards. The platform also delivers advanced mechanisms and policies for task scheduling, both for the server and client sides. BOINC-client software allows us to connect a computer to any set of BOINC-based projects. As each computer has various resources available, each project can supply clients with jobs that may use various combinations of computing resources. Based on this, resource-share accounting is implemented to assure fair access to the resources required for jobs coming from each executed project. Moreover, the weighted round-robin policy and other job-scheduling and fetching policies (broadly described in [8]) are implemented. BOINC has attracted many volunteer computing projects and a large number of volunteers. Currently, it is one of the most-popular platforms in the world of volunteer computing. It can be seen on various lists of volunteer-computing projects, such as the one on Wikipedia [44] and on distributedcomputing.info [36] – most of the projects described on these two sites are BOINC-based. At the end of 2005, the SETI@home project dropped its own software platform (called SETI@home Classic) and migrated to BOINC [43]. In 2007, a similar thing happened to WCG [5, 6]. Computations performed using BOINC projects were used in many scientific publications, which are listed on the project's web site [4].

Even though BOINC is already ten years old, it is still able to follow the current trends and adjust to the needs of volunteers. One example of this is the fact that BOINC supports GPUs, which play an important role in scientific computations thanks to technologies like OpenCL or CUDA [3]. Also, in an effort to capitalize on the development of the mobile device market, BOINC released its official volunteer application for the Android platform [2].

In December 2015, nearly a quarter million volunteers were active, resulting in an average computing power of around 158 000 TeraFLOPS for the whole platform [1], which was a few times better than the best supercomputer in the Top500 list [41]. More than 40 projects are currently listed as officially supported by BOINC [42], and some of these (e.g., WCG) are the so-called "umbrella projects" containing multiple sub-projects.

3.2. Web-based volunteer computing

Volunteer computing projects usually require volunteers to install dedicated software that downloads tasks from servers, executes them, and sends back the results. The need for installing additional software can be eliminated by using a web – based application. In this case, a webpage opened in a browser tab performs computing tasks. This may encourage the users to become volunteers. Some projects use approaches that can be classified as a hybrid of the two methods mentioned above. This is the case when volunteers must install a web browser plugin to take part in the computations. On one hand, a web browser is used by volunteers. On the other, additional software still must be installed in the form of a plugin to incorporate full functionality. All of these approaches have their advantages and disadvantages, which depend largely on kinds of computations performed and target group of people who are eyed as volunteers.

The advantages of installing dedicated software over using a web browser are as follows:

- Computation speed – dedicated software can be optimized for a particular hardware platform and operating system. There is no overhead of JavaScript interpretation or communication with browser APIs (Application Programming Interfaces). On the other hand, initiatives like asm.js and V8 engine are contributing to slowly reducing the difference in efficiency between JavaScript and low-level programming languages.

- Direct hardware access – dedicated software can use all of the available hardware resources (such as disk drives or graphic processing units) and is not constrained by browser APIs.

- "Stability" of volunteers – dedicated software can be started automatically with the operating system and always works in the background. Volunteers do not have to remember to turn it on, and there is no risk that it will be turned off by accident. After one-time installation and configuration, volunteers can be passive and forget about the project, and still their computing resources will still be shared. This is not the case with a web browser, where volunteers need to consciously open the web page every time they want to participate, and their participation ends as soon as they close their web browser or tab (accidentally or intentionally).

On the other hand, there are also advantages of using a web browser as a volunteer computing client:

- Easy start – the role of web browsers is increasing every year, and many applications that used to work only as standalone programs are moving into the web. A web browser is probably one of the most frequently used pieces of software on many computers, and its usage is often the first activity that non-technical people learn. Opening a web page is likely an easy task for most Internet users, while installing and configuring standalone software can be difficult (even for more experienced computer users). Web-based applications are easier to install and maintain, which makes them more attractive for potential volunteers. Making it easy to become a volunteer is very important in volunteer computing, as the number of volunteers has a big impact on the success of volunteer computing projects.

- Simple development – web browsers hide the differences between various hardware platforms and operating systems that volunteers may be using. Thanks to standardization, differences between the behaviors of different browsers are quite small. A programmer implementing a computational task to be run on volunteers' devices can use a high-level JavaScript language, and the same code will be run no matter what kind of device is used by a volunteer. The difficult task of supporting new hardware platforms can be only performed by web browser creators and not by people creating volunteer computing projects. On the other hand, one has to notice that many significant scientific algorithms have already been efficiently implemented in languages like C or Fortran, and similar implementations may not be available in JavaScript. Making existing code to work with volunteer computing always requires some amount of work. However, reimplementing the whole algorithm in a different language can be much more complicated. There are tools to transcript low-level programming languages into an effective JavaScript code (such as Emcscripten), which may be helpful in certain cases.

- Security – unfortunately, the process of installing new software always carries some risks. Every installed application access the hard drive data and harm the users device in many ways (intentionally or by programmer's error). Instead it is safer to run a JavaScript code in an isolated sandbox created by a web browser. The risks of harming volunteer's device in this case are constrained, as web browsers are designed to be protected from malicious sites that their users may visit by mistake. The risk is also reflected by the accessibility of both solutions: installing dedicated application may sometimes be impossible without administrator's rights, while accessing a web page is usually allowed for every user.

Bayanihan

The Bayanihan project described in 1998 in an article titled "Bayanihan: Web-based volunteer computing using Java" [38] was probably the first platform that allowed volunteers to use web browsers in the computations. Java applets were used to achieve this goal. Although the most popular computer technology is nowadays different, the advantages of using a web browser for the volunteer computing mentioned in the Bayanihan paper still seem to be true:

- Ease of use – accessing a web page requires only basic technical knowledge.

- Platform independence – ability to compute tasks on different platforms is provided by the Java Virtual Machine.
- Security – tasks are isolated from the operating system (Java applets are more constrained than standalone Java programs).

Currently the Bayanihan project is probably not being maintained. According to the Internet archive (archive.org), the last news was added to it in 2002, and the official Bayanihan site (bayanihancomputing.net) disappeared in 2013.

GridBee

GridBee was created in 2011 with the goal of creating a way to support BOINC-based projects with web browsers. The official BOINC client is a dedicated software that must be installed. GridBee offers an alternative client that can be started in a browser, and it is able to communicate with standard BOINC servers. GridBee is described on its web page [13] and was also introduced in the presentation during "The 7th BOINC Workshop" [40]. The project is open source, and its sources are available on GitHub. GridBee authors claim that creating a web client will make easier to become a volunteer, ultimately increasing the number of volunteers.

GridBee is written in a high-level programming language called Haxe, which can be compiled to multiple target languages such as JavaScript, PHP, C++, Java, or Python. In the case of GridBee, Haxe is compiled into the JavaScript code that is executed by the volunteer's browser and acts as a client. GridBee uses HTML5 technologies such as "Web Workers" (running tasks in separate threads) and Local Storage (saving users' configuration and partial task results). The client connects with the BOINC servers to download tasks and send the results back.

GridBee can be used to compute tasks written in JavaScript or C/C++. JavaScript tasks can be run in every modern web browser, while C/C++ tasks require the Native Client (NaCl) technology available only for Google Chrome. NaCl makes it possible to run compiled code in a sandbox. Using C/C++ tasks and requiring NaCl support decreases the number of potential volunteers, but makes it easier to reuse existing C/C++ sources.

CrowdCL

CrowdCL (Crowd Computing Language) was described in the article entitled "CrowdCL: Web-based volunteer computing with WebCL" (Web Computing Language) [32], published in 2013. The project aims not only at creating a volunteer computing platform that makes it possible to become a volunteer while using a web browser but also allows volunteer computing project maintainers to utilize the volunteers' graphic processors.

CrowdCL client software uses WebCL, which makes it possible to use OpenCL from the JavaScript code executed in a web browser. OpenCL is a framework that (among other functionalities) makes it possible to perform computations on GPUs and CPUs supporting that standard. In summary, thanks to WebCL, a web page creator can perform computations on the GPUs of the web page visitors' devices.

CrowdCL authors created an additional layer of abstraction over WebCL called KernelContext, which is assumed to be easier to use by developers. Another abstraction is the CrowdCLient library, which can perform a given task in the background and send the result back to the project's server.

The server software is written in node.js and exposes HTTP interfaces to which task results can be sent. The server can store data in MongoDB or MySQL. A project maintainer can supply a validation function that checks if the results are within the expected range. The server executes that function, and rejects invalid data before saving the results.

It is important to notice that the CrowdCL server does not take part in task distribution. The platform assumes that a task's code will be embedded in the JavaScript code of the page that the

volunteer visits. CrowdCLient is only responsible for executing that code and sending results to the server.

Comcute

Comcute was created in 2012 by researchers from Gdansk University of Technology. One part of this project is a volunteer computing platform with the main goal of providing large computing resources during a crisis (for example, a nuclear plant accident). Comcute authors claim that the project can become a real Polish alternative to BOINC. The functionality and architecture are described on the project's official web page [14] and in multiple publications [18–20].

Volunteers can join the project using a web browser or dedicated client software supporting Windows 8. The source code of the client is to be embedded into the web page advertisements and executed by the software that renders them. To accomplish this, the project authors utilize JavaScript along with Adobe Flash and web browser plugins supporting that technology to execute tasks.

In ComcuteJS [20] (part of Comcute), the web volunteer client is based on "Web Worker" technology that executes the JavaScript code in a separate thread. The server side is based on the PHP language and MySQL database. The project architecture consists of two types of servers: Comcute Core (responsible for managing the computations and aggregation of the results) and Computational Task Dispatchers (responsible for the direct communication with volunteers' web browsers). Each of these groups can consist of multiple nodes, and the servers communicate by HTTP, sending messages in the JSON format.

CrowdProcess

The CrowdProcess platform (crowdprocess.com) is a volunteer computing platform that uses modern web applications such as REST interfaces and HTML5 technologies. One instance of the platform is publicly available, and each person can create a new account. Neither the source code nor binary files can be downloaded. To create a new platform instance for private purposes, one has to buy a commercial version called "Enterprise Grid", which is not further described on the official website of CrowdProcess.

After creating an account and logging in, each user can submit a new task to be computed by the platform. A task consists of JavaScript code defining a function and JSON data that will be provided as an input to that function. The function is executed by the volunteer's browser.

CrowdProcess users who own web pages can generate a code snippet. Upon adding this snippet to these web pages, every person visiting them can automatically become volunteer. Such users can see in their dashboard how much computing time has been donated to the platform by the visitors to their pages. Platform authors claim that embedding code into volunteer web pages does not have negative impact on the user experience of the visitors. The tasks are executed in an isolated environment of a Web Worker and an invisible *iframe* HTML tag. The scope of a task inside a Web Worker is constructed in a way that makes it impossible to use some potentially dangerous HTML5 features from the task code.

4. Lightweight volunteer computing platform

In this paper, we present volunteer computing platform named **Edward**, which connects volunteers with people creating computational tasks. The most-important feature that makes Edward stand out from the other solutions mentioned in Subsec. 3.2 is the possibility of quickly creating a new platform instance to be used for private purposes. The platform is free and open source. Starting the platform requires only opening a single file and there is no need to go through a complicated

installation or configuration process. Moreover, Edward has a unique combination of features that cannot be found in other similar solutions:

- Volunteers do not have to install any additional browser plugins, and the technologies used in the volunteer application are natively supported by modern browsers.
- Sending tasks and downloading results can be performed by a dedicated web application and an HTTP interface, which makes it possible to integrate the platform with existing software.
- Platform behavior is configurable (for example, users can specify task priorities or redundant computing factors).
- Volunteer code can be added to an existing web page, which automatically enables all visitors to become volunteers.

Edward is written in Java and uses the H2 database (also written in Java). Therefore, the platform can be run in every environment supported by the Java Virtual Machine.

4.1. Terminology

"Project" is the first object that the platform user must create. Projects are used only for grouping purposes.

A project can contain many jobs. "Job" contains code that will be executed on the volunteers' devices. This should be a valid JavaScript code containing global function *compute(input)*. Results returned by this function will be sent back to the platform server and can be downloaded from there.

A job can contain many tasks. "Task" contains input data in the JSON format, which will be passed as a parameter to the function defined by the job to which it belongs.

Each time a volunteer's device downloads a new task to execute, the platform creates a new "execution" object that represents the state of a single execution and can contain an error message or a result. A task is considered to be finished as soon as at least one execution of this task is successful and contains a result. Multiple executions can be assigned to a single task in the cases when redundant computations are configured or an execution finishes with an error and the platform retries the computation.

For example, to use the platform to compute the squares of the numbers 1 through 5, the user should:

- Create a project, for example, "Simple Math"
- Create a job "Square" with the code:

```
function compute(input){
    return input*input;
}
```

- Assign tasks to "Square" in the form of a JSON array:

```
[1, 2, 3, 4, 5]
```

In the most favorable scenario, each of the five tasks will be executed only once, and the execution objects in the platform database will contain a result with the square of the appropriate number.

4.2. High-level view of platform architecture

The application is divided into modules that can be dependent on each other. Dependencies are shown in Fig. 2. The arrow pointing from Module A to Module B indicates that Module A uses Module B (A depends on B). For clarity, the transitive dependencies are not stem.

- Core – the main module on which all other modules depend. Contains the data model (classes like Job, Task, Execution). Manages the platform configuration.
- Persistence – responsible for communicating with the database. Contains Data Access Object classes, creates database schema, and manages connection pool.
- Management – contains the business logic. Handles task scheduling and monitors volunteer devices.
- Communication – responsible for the platform communication with the outside world. Exposes HTTP (Hypertext Transfer Protocol) interfaces to the platform web application management and to volunteer client. Contains web application configuration, HTML (Hypertext Markup Language) and JavaScript sources.
- Rest-client – can be used by developers to integrate existing software written in Java with the platform. Facilitates methods that can send tasks to the platform and download the results.
- Shell-client – exemplary module using rest-client that shows the platform functionality in a command line.

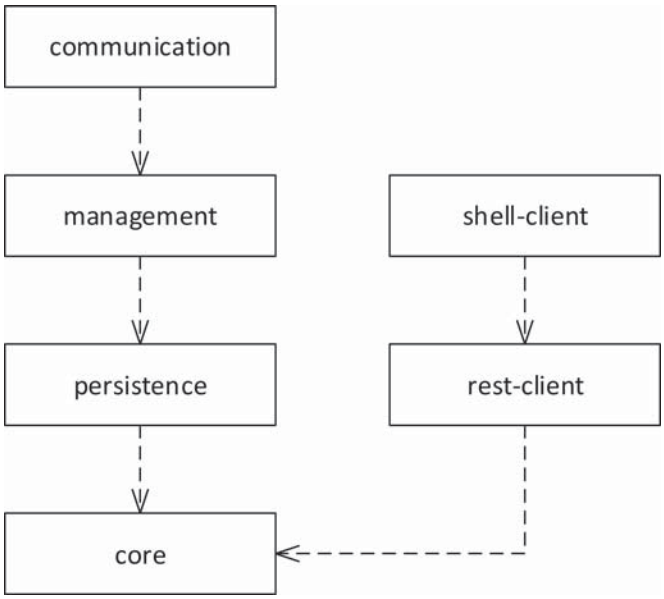


Fig. 2. Diagram of module dependences.

Running the HTTP server is handled by the Spring Boot project. Spring Boot can run a Java web application in an embedded servlet container (Tomcat by default). It can also package all dependencies into a single JAR file. Thanks to this, the whole application can be run using a single command:

```
java -jar edward-executable-VERSION.jar
```

This makes it easy to start a new instance of the platform (which is one of its main goals).

4.3. Task scheduling

The most important responsibility of a volunteer computing platform is task scheduling. The platform now focuses on the reliability of task executing and provides policies to cope with very unreliable devices and errors in the provided client-side code, which could cause infinite task execution time.

The platform retries a task execution if a volunteer device does not send a result within a predefined time frame. Additionally, it is possible to send a task to multiple volunteers at the same time, which increases the probability of getting the result without the need for further repetitions. If the task code is written incorrectly and the task executions always finish with errors or never finish at all, the platform gives up retrying the task after several attempts.

Users can provide several configuration parameters when adding a task that influences task scheduling:

- Timeout – if the volunteer device does not complete the assigned task within a given amount of time, the platform marks the execution as failed and sends the task to another volunteer.
- Priority – tasks with higher priority are performed before those with lower priority.
- Concurrent executions – if equal to 1, the platform waits for execution failure before trying to send the task to other volunteers. If greater than 1, the platform immediately sends the task to the corresponding number of volunteers (which increases the probability of a quick execution).

The next task to be performed is determined by using the current database state and this requires a lot of work. Therefore, the platform does not dynamically compute the next task each time a volunteer asks for it. Instead, the platform recomputes a queue of tasks to be performed in a configurable time interval and stores it in memory. Each time a volunteer asks for a new task, the first task from the queue is allocated. This solution has a significant drawback as a database change might occur that could alter the order of the tasks during the time between queue reloadings (for example, a new result is sent, or a new task with high priority is created). However, this is not reflected until the next queue refreshes. In practice, the impact of this drawback is lessened if the refresh interval is small.

4.4. Volunteer web client

One of Edward's main features is making possible to become a volunteer by using only a web browser. Therefore, the client code is written in JavaScript and it is executed after the user visits an HTML web page.

Volunteer code should be written in a way ensuring that it does not negatively impact the user's browsing experience in other open browser tabs.

JavaScript is single-threaded by default. So, if a web page script invokes a function that takes long time to execute, the web browser will not be able to handle user interface interactions (such as clicking) during its execution. In this case, the user experiences a web page that becomes non-responsive or completely "freezes". Modern browsers usually detect long function calls automatically and display a dialog box asking users if computations should be stopped.

This constraint makes it difficult to run long complicated computations in a browser. One way to work around this is to divide long-method calls into shorter computations and, at the end of each part's schedule, to execute the next part using the "setTimeout" method. This enables the JavaScript thread to handle the user interactions between performing computations, and the web page remains thereby responsive. Unfortunately, this approach has some disadvantages that rule out its use in the volunteer code. Firstly, frequent "breaks" make computations less efficient and take longer to finish. Secondly, function code has to be written in a way that divides computations

into smaller parts and executes them in an asynchronous way. Therefore, writing the code could be more difficult for people using the platform.

Modern browsers offer a "more-elegant" way of solving this issue with technology called "Web Workers". This technology makes it possible to create a "Web Worker" that executes JavaScript code in the background, independently of the main script that handles user interface interactions. A Web Worker does not have direct access to a document's DOM tree, and it communicates with the "main" script using message passing. Objects sent in messages are copied and not shared (the sender serializes them, and the receiver deserializes them). Therefore, there are no synchronization issues in accessing data from multiple threads. Web Workers turned out to be a good choice to be used in the volunteer JavaScript code.

A sequence diagram describing the procedure of a volunteer script is shown in Fig. 3.

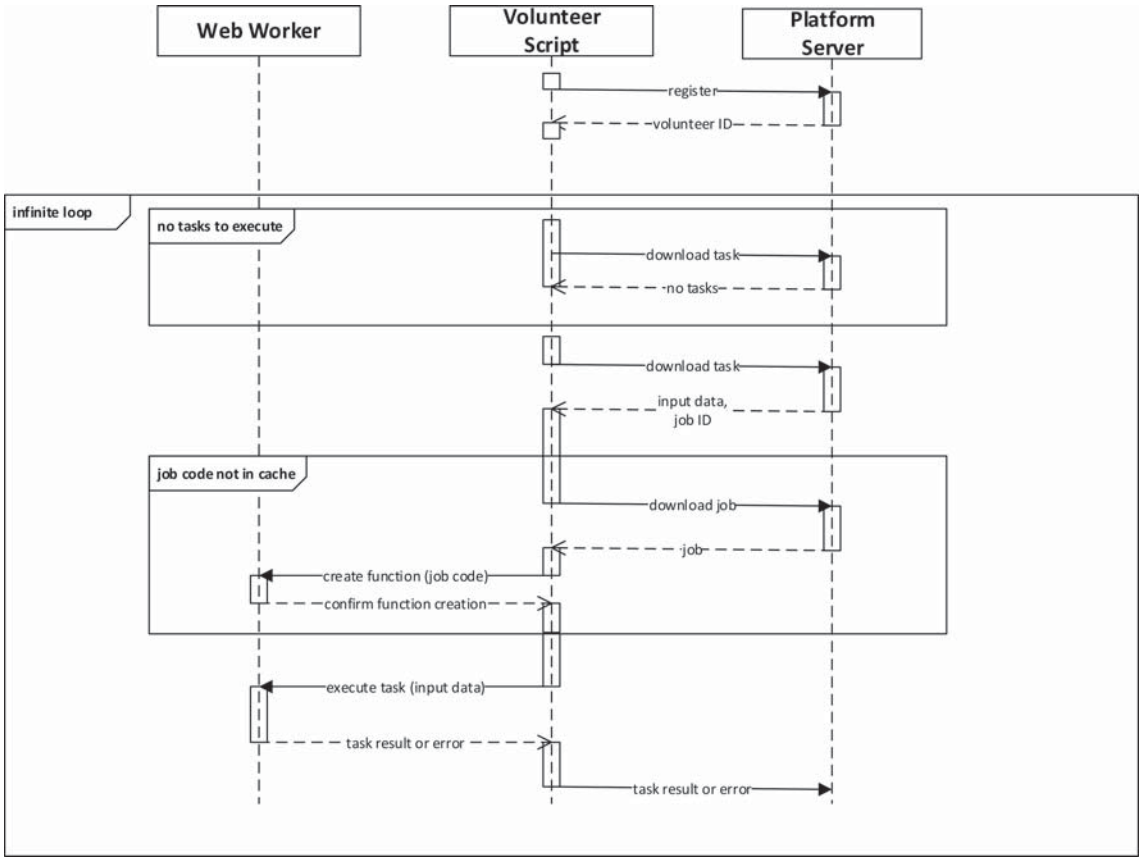


Fig. 3. Sequence diagram of a volunteer script.

At the beginning, the volunteer script registers on the platform and receives a volunteer identifier, which is later added to each message sent to the server. Then, the volunteer script asks the server from time to time if there are any tasks to do. If there are some tasks, the server responds with the tasks and a job identifier. The volunteer script checks if the JavaScript code for that job is already cached (the code can be found in the cache if that browser instance has already performed any task for the same job in this session). If the job code is not yet cached, it is downloaded from the server and sent to the Web Worker to create the appropriate function in the web worker's context. When the Web Worker is ready, input data is sent to it, and the task is executed in the web worker's context. When the Web Worker finishes its task execution or an error occurs, it sends the appropriate message to the main script, which then passes this message on to the platform server. After the result or error is delivered to the server, the volunteer script asks again for tasks to perform. This closes the loop.

Theoretically, all communication with the platform server could be done by the Web Worker, as it is able to perform HTTP requests. Unfortunately, the fact that a web worker is unable to access the DOM tree makes it impossible to use popular JavaScript libraries that perform HTTP calls using AJAX (Asynchronous JavaScript and XML) (for example, jQuery cannot be used in a web worker context). Therefore, to make the code clear and concise, communication with the server is performed in the main script in the current version of the platform, and only the task executions are delegated to the Web Worker. In practice, this does not have a negative impact on user experience, as both communication with the server and communication with the Web Worker are performed asynchronously and do not block the main thread for too long.

To facilitate volunteering, a code snippet has been implemented, which can be embedded into an existing web page. That enables its visitors to automatically become volunteers. The code creates an invisible iframe (with the appropriate privileges) and opens a volunteer page inside.

```
<script>
var volunteerPageUrl = "http://localhost:8080/volunteer/";
(function () {
    document.addEventListener("DOMContentLoaded", function () {
        var iframe = document.createElement("iframe");
        iframe.src = volunteerPageUrl;
        iframe.sandbox = "allow-scripts␣allow-same-origin";
        iframe.style.display = "none";
        document.body.appendChild(iframe)
    });
})();
</script>
```

This code snippet makes it possible to increase the number of volunteers participating in volunteer computing projects by cooperating with the owners of popular web services.

4.5. Data persistence

The data used by the platform is stored in a relational database. It contains information about projects, jobs, tasks, and results.

The platform communicates with the database using the JDBC (Java DataBase Connectivity) protocol and jOOQ (Java Object Oriented Querying) library. jOOQ is a library exposing a simple Java DSL (Domain Specific Language), forming an abstraction over SQL and providing type safety during compilation. Thanks to its simplicity, jOOQ gives the programmer full control and understanding of which queries are executed in the database and when this happens, which in many cases can be a better choice than using a full ORM (Object-Relational Mapping) solution such as Hibernate. The drawback is that it is impossible to automatically map database table relations to Java object relations. Because of this, the platform's model classes do not directly reference each other but contain identifiers of related objects. For example, the Job class contains a field with a parent project identifier instead of a reference to the Project class. In practice, this constraint turned out to not be a problem, and it simplified building the HTTP interface (as objects could be easily mapped to JSON without serializing the whole object graph).

The following code is a part of the project's persistence layer and shows what the jOOQ code that retrieves a list of jobs for a given project looks like:

```
dslContext
        .select()
        .from(Tables.JOBS)
        .where(Tables.JOBS.PROJECT_ID.eq(projectId)));
```

The H2 database was used during the development and tests of the platform. It is a small file-based database written in Java. Using H2 makes it possible to start the platform without configuring complicated database management systems. In addition, having the whole database in a single file makes backups and migrations easy.

5. Experimental verification

In this section, actual outcomes of testing the platform efficiency are presented. The platform was tested in a cloud environment, and it was applied to solve three quite-different problems: simulation of work using the *sleep* function, solving the LABS (Low Autocorrelation Binary Sequence) problem using a genetic algorithm, and a distributed search in Wikipedia.

5.1. Experimental setup

All tests were performed using Amazon EC2 virtual machines to make sure that they are reproducible. The platform server and volunteer browsers were each run on separate T2.micro instances, and the program that coordinated the tests used a single M4.large instance.

T2.micro is the least expensive instance type available on Amazon EC2, and it is also available for free within the so-called "Free Tier". Its minimal cost makes it feasible to perform many tests using multiple virtual machines. T2.micro has only one virtual CPU and 1 GB of RAM. This was enough to perform the tests, and it showed that both the server and volunteer client do not require excessive hardware.

T2 machines are "Burstable Performance Instances," which means that the maximal computing power can be used for around 30 minutes after starting a new instance, and then the computing power drops significantly. To return to full computing power, one has to decrease the load for some time to collect so-called "CPU credits." The tests were designed so that the full computing power was always available – they were shorter than 30 minutes, and virtual machines were created from scratch for each round of the tests.

The application that coordinated the tests used an M4.large machine, which has a constant performance over time so that it could coordinate several rounds of tests without being restarted.

Tests were run using 32 volunteer machines at most, which was possible after asking Amazon support to increase the limit of T2.micro instances running at the same time from 20 to 33.

Ubuntu Server 14.04 LTS was installed on each virtual machine. The test coordinator application used Amazon EC2 SDK for Java to manage the virtual machines and the sshj library to communicate with them.

The platform server virtual machine was configured with Java 8, Apache Maven, git, and H2 database installed.

Testing a volunteer code required running it within a system without any graphic environment. PhantomJS is a headless browser that could potentially be used. However, during the first tests, it turned out that its implementation of Web Worker technology is different than in common browsers, which made it impossible to finish the tests. To overcome this issue and make the environment more realistic, tests were run using a Mozilla Firefox browser. To run it from the command line without a graphic environment was possible thanks to the Xvfb application, which simulated a graphic environment in the memory. The browser was run with a fresh user profile in each test round to make sure that there are no traces of the previous test rounds remaining.

The steps used to perform the tests were as follows:

1. Start as many virtual machines as there are volunteers needed in tests.
2. Start the server virtual machine.
3. Start the volunteer computing platform on the server virtual machine.

4. For each tested number of volunteers:
   (a) Turn off all browsers on volunteer virtual machines.
   (b) Wait until server reports having no connected volunteers.
   (c) Start web browsers on volunteer virtual machines and open volunteer web page.
   (d) Wait until server reports reaching the expected number of connected volunteers.
   (e) Create set of tasks on platform and wait until they are completed.
   (f) Download task results from platform server.
   (g) Save computation time and additional information into file.
   (h) Turn off some volunteer virtual machines, so that the number of volunteers is equal to the number expected in the next round of tests.
5. Turn off all created virtual machines.

5.2. Sleep function

During this test, tasks consisted of calling the *sleep* function (which performed active waiting for a number of milliseconds) passed as an argument. This allowed us to measure the time overhead added by the platform when delegating tasks to volunteers without the taking real computing power of the volunteer machines into consideration.

5.2.1. Efficiency

The efficiency test was performed for 1, 2, 8, 12, 16, 24, and 32 volunteers. Total "waiting" time to be performed was always constant and equal to 640 000 milliseconds. To check how the number of tasks and time of a single task influence the time overhead added by the platform, total waiting time was divided into different numbers of tasks: 32, 64, 128, 256, 512, and 1024. In the first case, each task awaited for 20 000 milliseconds, and in the last case, each task took only 625 milliseconds. Figures 4a, 4b, and 4c show the time, speedup, and efficiency results, respectively, for different configurations.

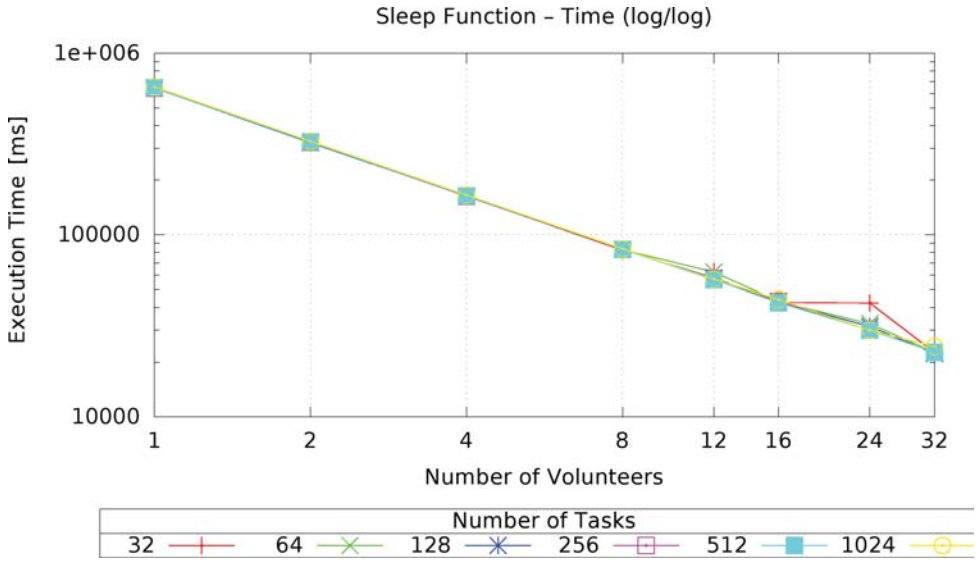
Speedup increases as the number of volunteers increases. For the most optimal configurations with 128 to 512 tasks, efficiency does not fall below 0.8.

It can be seen that the results for 12 and 24 volunteers are significantly worse than for the others. The reason is that none of the tested numbers of tasks is divisible by 12 (or 24) without a remainder. Therefore, the computing power of several volunteers is not fully utilized in such cases. For example, when the number of tasks is equal to 32 and there are 24 volunteers, the first 24 tasks will be finished at almost the same time, and then 8 volunteers will execute the 8 remaining tasks while 16 volunteers will have no tasks to perform.

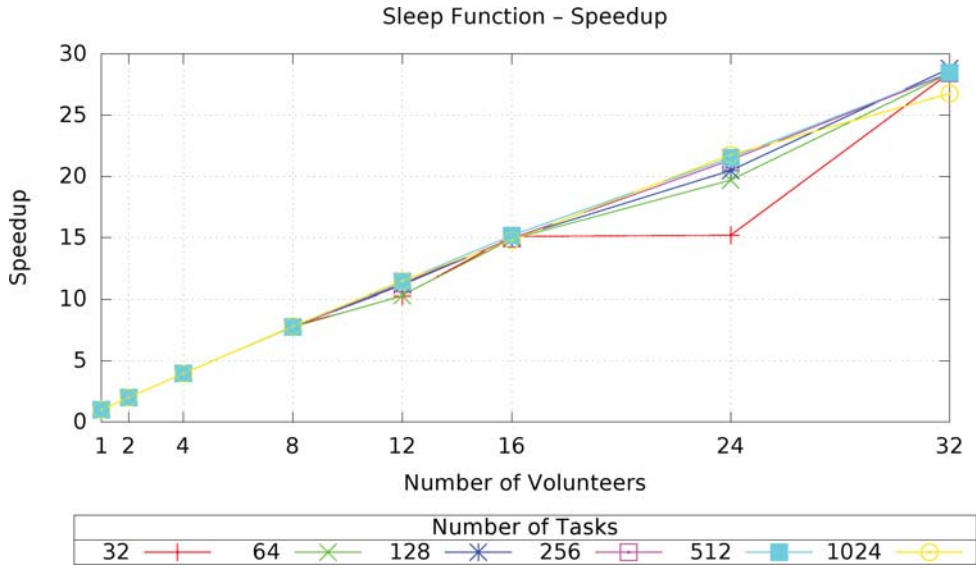
The negative effect of these situations diminishes as the number of tasks increases, which shows that dividing a problem into smaller parts makes it easier to efficiently divide work between volunteers. On the other hand, the time overhead of task scheduling is relatively high when the problem is divided into parts that are too small. In our study, for 32 volunteers, the speedup for 1024 tasks (625 milliseconds each) is lower than for a smaller number of tasks.

The conclusion is that one must choose the appropriate time of each task when dividing the computations to be performed on the platform into separate task. Long-running tasks make it harder to efficiently use the donated computing power and increase the chances of volunteers disconnecting when performing a task. Too short tasks can make the related overhead of managing them too large.

a)



b)



c)

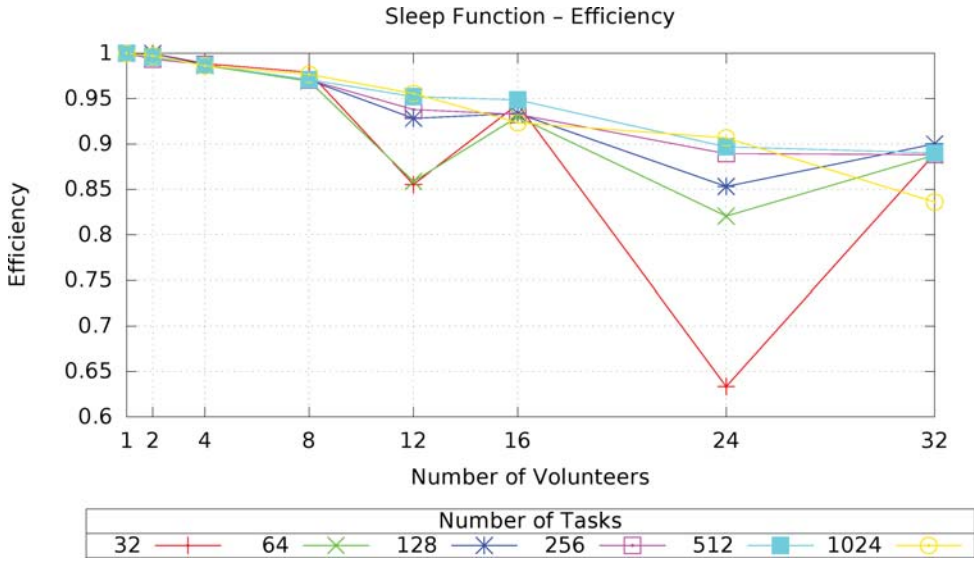


Fig. 4. Efficiency results in *sleep* function test: a) time, b) speedup, c) efficiency.

5.2.2. Test for handling disconnections

Using a web browser as a client increases the risk of volunteer devices being disconnected in the middle of performing computations. The goal of this test was to check the impact of such disconnections on the time required to finish the tasks and check whether the platform can be configured to mitigate that impact.

During this test, volunteers also performed an active waiting with the "sleep" function, and each task took 60 seconds. There were 12 tasks and 32 connected volunteers. In the background, a timer thread restarted a random volunteer browser when the configured interval of time passed.

As the number of tasks was lower than the number of volunteers, some volunteers were not performing any tasks by default. This can be changed using the "concurrent executions" configuration property. By default, this is equal to 1, which means that only one volunteer can simultaneously perform a given task at each given moment. Increasing the value means that the next volunteer can start computing the task even when one volunteer is already performing it. Theoretically, this wastes computing resources by performing redundant computations, but it can be helpful when some volunteers disconnect when performing tasks or return results very slowly. As the result is available as soon as any volunteer finishes the task, increasing the "concurrent executions" increases the chances of getting the result without the need of restarting parts of the computation. This can be especially useful when some of the volunteers are idle or perform lower-priority tasks that can be set aside to make sure that more important tasks are finished on time.

Figures 5 and 6 show the average time of the test for random volunteer disconnection intervals equal to 5 and 2 seconds. The test was performed for 'concurrent executions' equal to 1, 2, and 3. As random volunteers were disconnected and the total time varied significantly between the test rounds, standard deviation occurred as shown in the charts.

In both cases it can be seen that, increasing the "concurrent executions" factor had a positive impact on the computation time, and the impact was greater when volunteers disconnected more often. In case when a random volunteer disconnected every 2 seconds, increasing "concurrent executions" from 1 to 3 decreased the average computation time from about 260 to about 120 seconds. The test shows that redundant computations can be an effective tool for ensuring short computation time in difficult environment of volunteer computing, in which computing devices can be unreliable.

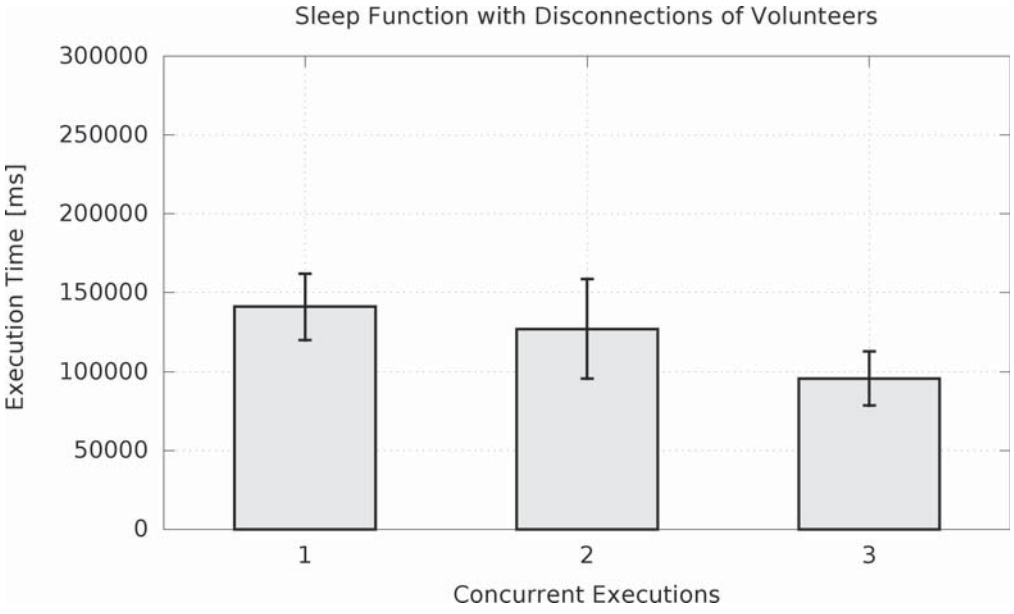


Fig. 5. *sleep* Function – disconnecting volunteer every 5 seconds.

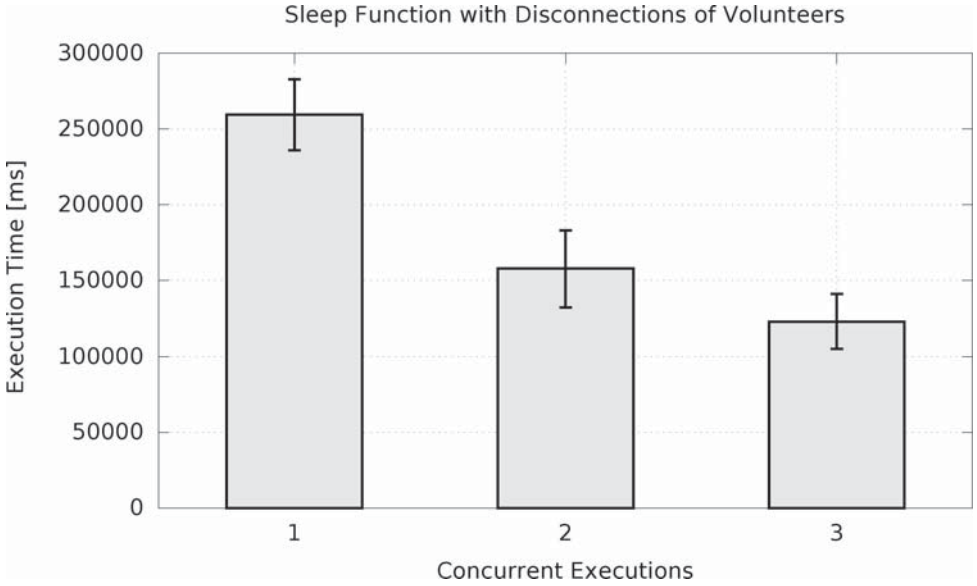


Fig. 6. *sleep* Function – disconnecting volunteer every 2 seconds.

5.3. Genetic algorithm – solving the LABS problem

To test if the platform performs well when solving a complex real-world problem, it was used to apply common genetic algorithm to LABS (*Low Autocorrelation Binary Sequences*) – a hard discrete optimization problem. Both the problem and algorithm represent a wide range of computational research related to solving hard optimization problems with biologically inspired metaheuristics [15, 37]. The LABS problem is a very hard combinatorial optimization problem with many applications in areas such as telecommunications, physics, and chemistry. A detailed description of LABS and methods of solving it by using the evolutionary approach can be found in [27, 31, 34]. There have been several attempts to solve such problem by using various techniques such as evolutionary multi-agent systems [30], memetic algorithms [31], or GPGPU [45]. In our study a separate application called Charles was created to coordinate the execution of the genetic algorithm using the HTTP API of the platform.

The genetic algorithm used the island model: individuals were divided into multiple populations that were improved separately, with some individuals migrating between populations in the migration step performed at regular intervals. While the population generation and migration were performed locally by Charles, the tasks of improving the populations were delegated to volunteers, as shown in Fig. 7.

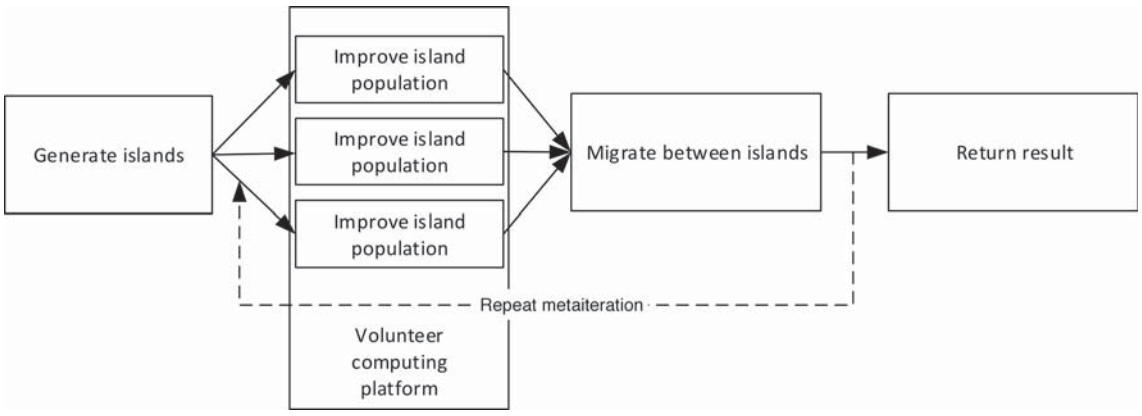


Fig. 7. Simplified schema of genetic algorithms in Charles.

As neither the LABS problem itself nor the genetic algorithms are the main subject of this study, the test code did not implement every known way to make the solution optimal, and the parameters of the genetic algorithm were not tuned. However, the algorithm was tested to see if it works correctly by executing it on the problem sizes of 40, 50, 60, and 70. Optimal solutions were found for the sequences of 40 and 50 elements, the results were compared to those already known [27, 34].

The platform efficiency test was executed with the following parameters:

- Number of populations: 32.
- Individuals in population: 50.
- Number of migrations: 4.
- Executions of evolutionary operators between migrations: 200.
- Length of sequence: 50.
- Number of volunteers: 2, 4, 8, 12, 16, 24, 32.

A single task for a volunteer was to execute evolutionary operators (selection, migration, crossover) 200 times on a single population. Charts of time, speedup, and efficiency are shown in Figs. 8a, 8b, and 8c.

It must be emphasized that the migration of individuals between islands requires a "synchronization" of computations. The next round of population improvements cannot be performed by volunteers until all of the tasks for the current round are finished and the migration is performed. Because of this, if any volunteer device finishes its task earlier, it will not execute any task until the other tasks are finished (which makes this way of performing computations difficult to scale). This issue could be mitigated by implementing some kind of asynchronous migrations that would not require all of the populations to be present at the same time. It could also be mitigated by executing the genetic algorithm multiple times in parallel (probably with different parameters), so that the volunteers always have tasks to perform.

As expected, computation time decreases as the number of volunteers increase. Efficiency does not fall below 0.6, and in cases when the number of populations is divisible by the number of volunteers, it does not fall below 0.75. For 32 connected volunteers, the computation time was almost 25 times shorter than for 1 volunteer.

5.4. Searching through Wikipedia

During this test, volunteer devices downloaded Wikipedia articles and searched for a given keyword inside their contents. The set of downloaded articles consisted of 2665 articles related to programming and IT, and the task was to find all lines containing word "Java" using regular expressions. The problem is similar to many real-world problems such as web crawling or communication with HTTP interfaces. In this test, volunteers donated not only the computing power but also the Internet connection.

Implementation

Articles to be analyzed were divided into tasks in such a way that each task contained a similar number of articles (the difference in the number of articles between tasks was no greater than 1). However, the fact that the articles differ in length had to be taken into consideration. Therefore, the time needed to download and parse each of them was different. In effect, the difficulty of the tasks also differed significantly.

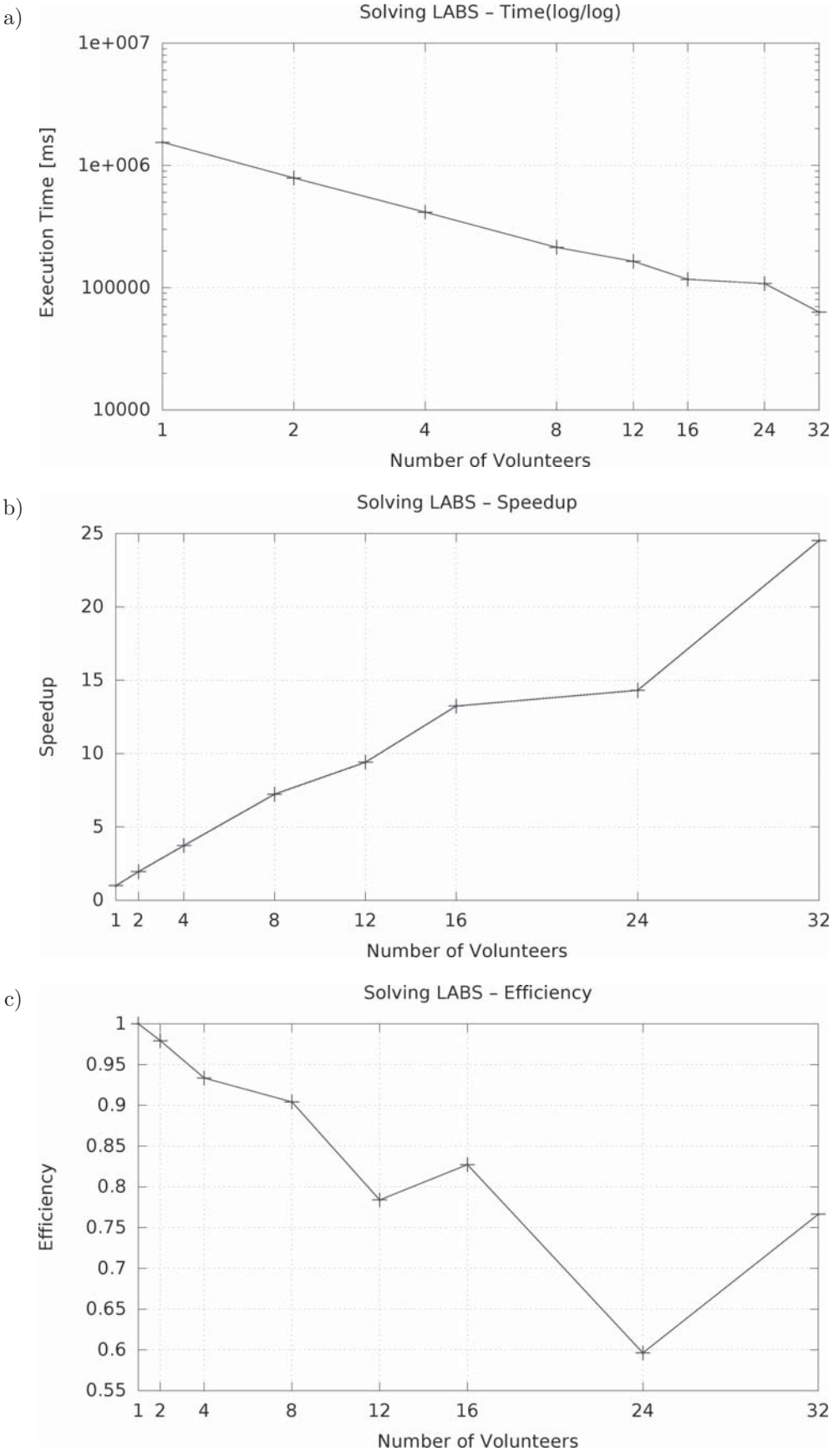


Fig. 8. Efficiency results in *LABS* function test. Solving LABS with genetic algorithm: a) time, b) speedup, c) efficiency.

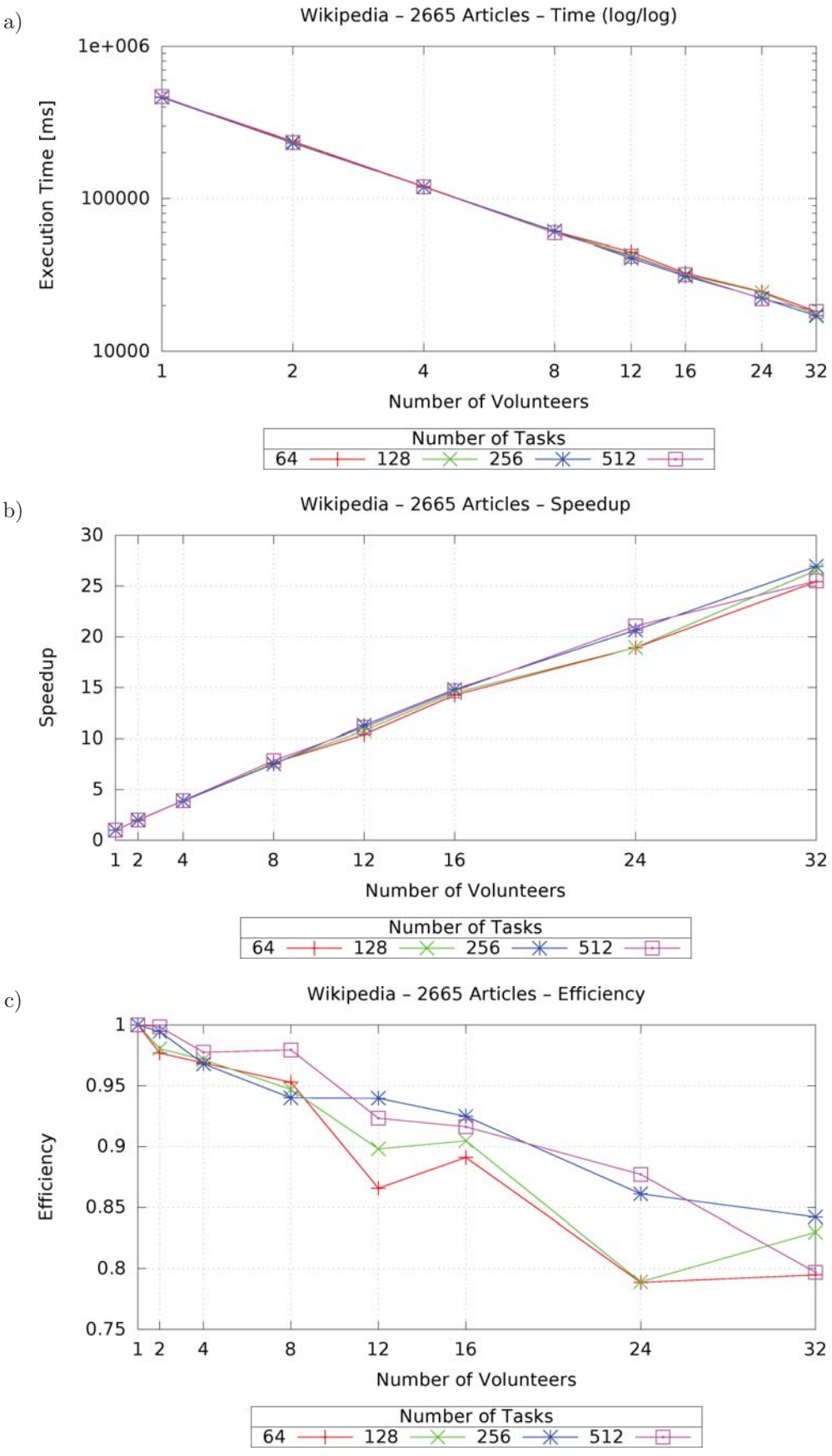


Fig. 9. Efficiency results in *Wikipedia search* function test. Analyzing 2665 Wikipedia articles: a) time, b) speedup, c) efficiency.

The "same-origin policy" prevents sending HTTP requests to another domain from a JavaScript code. Usually, JSONP (Java Script Object Notation with Padding) is used to work around this constraint. However, the tasks are executed in a web worker context, and using common JavaScript libraries that handle JSONP requests would be difficult. Instead of JSON (Java Script Object Notation), the *importScripts()* function provided by the web worker was used to download data from Wikipedia. The disadvantage of using this function is that it works synchronously – the function returns when the server returns the whole response. Synchronous calls are usually avoided in JavaScript programming, as asynchronous input/output operations can be much more efficient. In this test, volunteer devices downloaded and analyzed Wikipedia articles sequentially, which should be enough to measure the scalability of the platform. To perform the actual test task in the most efficient way, one should find the technique that not only works well with the "same-origin policy" but also allows to download and analyze the articles in parallel.

The articles were downloaded in the JSON format and parsed. After that, regular expression `/.*\bjava\b.*/ig` was used to find lines containing "java". A list of the lines was returned as the result.

Analyzing 2665 articles

2665 Wikipedia articles were divided into tasks and sent to the platform. There were 1908 lines containing word "java". The number of volunteers tested was as follows: 1, 2, 4, 8, 12, 16, 24, and 32. The articles were divided into 64, 128, 256, or 512 tasks. Figures 9a, 9b, and 9c show the time, speedup, and efficiency charts.

For lower number of tasks, it was observed that the results were significantly worse when the number of tasks was not divisible by the number of volunteers (tests for 12 and 24 volunteers), which is similar to the effect observed when testing the sleep function (5.2). In these cases, dividing the articles into 512 tasks was the best choice, as these many tasks make it easier to effectively schedule them. On the other hand, as the shorter tasks were introduced, the more overhead was allocated to communication and management. Therefore, the best choice was to divide the articles into 256 tasks for 32 volunteers.

With one connected volunteer, the test took about 460 seconds. For 32 volunteers, this was about 17 seconds (which represents an almost – 27 times speedup). In the ideal case of linear speedup increase, a test for 32 volunteers would take 460/32 = 14.375 s. Therefore, the difference between the actual result and the ideal case was only 3 seconds. Figure 9 shows that speedup increased quite steadily with the number of volunteers, and the results were similar for different ways of dividing articles into tasks. In the most optimal cases, the efficiency did not drop below 0.8.

6. Conclusions

Volunteer computing is an attractive idea of bringing together a community of donors, and constructing a supercomputing infrastructure that is capable of doing practically HPC simulations or computing without an integrated infrastructure, power sources, cooling appliances, administration staff, etc. Though some of such projects are still available, many of them went into oblivion over the last few years, and the prevailing ones have begun to be outdated. Therefore, there is a need for new volunteer computing frameworks based on the currently available technologies.

In this paper, thorough description of the proposed Edward volunteer computing platform was presented along with clear evidence of its usefulness which was demonstrated during the tests conducted using the Amazon EC2 cloud. In those tests, which involved up to 32 volunteers and different types of tasks, the platform demonstrated a relatively good logarithmic-like scalability. Surely, a further scaling striving to obtain an improved linear speedup is feasible. Therefore, performing tests with more resources and in different settings is planned in the future. The lightweight quality of our framework, its complete independence from the operating system, and apparent effi-

ciency all make it (in the opinion of authors) an appropriate tool of choice for dealing with problems requiring HPC power without utilizing an actual supercomputing infrastructure.

In the near future, a broader set of experiments is planned, emerging out of artificial environments like LANs (Local Area Networks) and commercially available clouds. These experiments will aim at employing real world resources as well as will explore the possibility of engaging real-life users to participate in the realization of the platform tasks.

Acknowledgments

The research presented in this paper was partially supported by the AGH University of Science and Technology statutory project.

References

[1] BOINC. Detailed Statistics of all Projects. http://boincstats.com/en/stats/1/project/detail.
[2] BOINC User Manual: Android FAQ. http://boinc.berkeley.edu/wiki/Android_FAQ.
[3] BOINC User Manual: GPU computing. http://boinc.berkeley.edu/wiki/GPU_computing.
[4] Publications by BOINC projects. http://boinc.berkeley.edu/wiki/Publications_by_ BOINC_projects.
[5] World Community Grid Forums. Linux is here!!! 2005. http://www.worldcommunitygrid.org/forums/wcg/viewthread?thread=4224#33880.
[6] World Community Grid Forums: BOINC Migration Announcement, 2007. http://www.worldcommunitygrid.org/forums/wcg/viewthread?thread=15715.
[7] D.P. Anderson. BOINC: A system for public-resource computing and storage. In: *5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004. https://boinc.berkeley.edu/grid_paper_04.pdf.
[8] D.P. Anderson. Emulating volunteer computing scheduling policies. In: *2011 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 1839–1846, 2011.
[9] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, **45**(11): 56–61, 2002. http://setiathome.berkeley.edu/sah_papers/cacm.php.
[10] D.P. Anderson, E. Korpela, R. Walton. High-performance task distribution for volunteer computing. In: *Proceedings of the First International Conference on e-Science and Grid Computing*, pp. 196–203. E-SCIENCE '05, IEEE Computer Society, 2005. http://boinc.berkeley.edu/boinc_papers/server_perf/server_perf.pdf.
[11] J.D. Baldassari. *Design and Evaluation of a Public Resource Computing Framework*. Master's thesis, Worcester Polytechnic Institute, 2006. https://web.wpi.edu/Pubs/ETD/Available/etd-042006-225855/unrestricted/baldassari1.pdf
[12] A.L. Beberg, D.L. Ensign, G. Jayachandran, S. Khaliq, V.S. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. In: *IEEE International Symposium on Parallel Distributed Processing*, pp. 1–8, 2009. http://www.hicomb.org/HiCOMB2009/papers/HICOMB2009-13.pdf
[13] BME Közigazgatási Informatikai Központ. GridBee Web Computing Framework – Official Website. http://webcomputing.iit.bme.hu/.
[14] M. Broniszewski, M. Poczwardowski, M. Zalewski. Comcute – Official Website. http://comcute.eti.pg.gda.pl/.
[15] A. Byrski, R. Dreżewski, L. Siwik, M. Kisiel-Dorohinicki. Evolutionary multi-agent systems. *The Knowledge Engineering Review*, **30**(2): 171–186, 2015.
[16] P. Chorazyk, M. Godzik, K. Pietak, W. Turek, M. Kisiel-Dorohinicki, A. Byrski. Lightweight volunteer computing platform using web workers. *Procedia Computer Science*, **108**: 948–957, 2017. Special Issue: International Conference on Computational Science, ICCS 2017, 12–14 June 2017, Zurich, Switzerland. http://www.sciencedirect.com/science/article/pii/S1877050917306348.
[17] D. Clery. IBM offers free number crunching for humanitarian research projects. *Science*, **308**(5723): 773, 2005. http://andrewlawler.com/website/wpcontent/uploads/Science2005LawlerAstronomers_Want_to_Be_Heard_Before_NASA_Acts775.pdf.
[18] P. Czarnul, J. Kuchta, M. Matuszek. Parallel computations in the volunteer – based Comcute system. In: *Conference on Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, **8384**: 261–271, 2014. https://www.researchgate.net/publication/260480164_Parallel_Computations_in_the_Volunteer_based_Comcute_System.
[19] B. Czerwinski, R. Debski, K. Pietak. Distributed agent-based platform for large scale evolutionary computations. In: *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 462–466, June 2011.

[20] R. Dębski, T. Krupa, P. Majewski. ComcuteJS: A web browser based platform for large-scale computations. *Computer Science*, **14**(1): 2013. http://journals.agh.edu.pl/csci/article/view/113.
[21] distributed.net. distributed.net History and Timeline. http://www.distributed.net/History.
[22] distributed.net. Project RC5. http://www.distributed.net/RC5.
[23] distributed.net. RC5-56/Disposition of Prize Money. http://stats.distributed.net/misc/money.php?project_id=3.
[24] distributed.net. RC5-72/CPU Participation. http://stats.distributed.net/misc/platformlist.php?project_id=8&view=tco.
[25] G. Fedak. Contributions to Desktop Grid Computing. Distributed, Parallel, and Cluster Computing [cs.DC]. Ecole Normale Supérieure de Lyon, 2015. https://hal.inria.fr/tel-01158462/file/hdr.pdf.
[26] G. Fedak, C. Cerin. *Desktop Grid Computing*. Chapman & Hall/CRC, 2012.
[27] J.E. Gallardo. C. Cotta, A.J. Fernández. Finding low autocorrelation binary sequences with memetic algorithms. *Appl. Soft Comput.*, **9**(4): 1252–1262, September 2009. http://www.lcc.uma.es/~ccottap/papers/labsASC.pdf.
[28] IBM. IBM, United Devices and Accelrys Aid U.S. Department of Defense in Search for Smallpox Cure. IBM News releases, 2003. https://www-03.ibm.com/press/us/en/pressrelease/335.wss.
[29] IBM. IBM Introduces "World Community Grid". IBM News releases, 2004. https://www-03.ibm.com/press/us/en/pressrelease/7404.wss.
[30] M. Kolybacz, M. Kowol, L. Lesniak, A. Byrski, M. Kisiel-Dorohinicki. Efficiency of memetic and evolutionary computing in combinatorial optimisation. In: *European Council for Modeling and Simulation, ECMS*, pp. 525–531, 2013.
[31] M. Kowol, K. Pietak, M. Kisiel-Dorohinicki, A. Byrski. Agent-based evolutionary and memetic black-box discrete optimization. *Procedia Computer Science*, **108**: 907–916, 2017. Part of the International Conference on Computational Science, ICCS 2017, 12–14 June 2017, Zurich, Switzerland. http://www.sciencedirect.com/science/article/pii/S1877050917307573.
[32] T. MacWilliam, C. Cecka. CrowdCL: Web-based volunteer computing with WebCL. In: *High Performance Extreme Computing Conference (HPEC)*, IEEE, pp. 1–6, September 2013. http://tommymacwilliam.com/docs/publications/hpec13.pdf.
[33] Mersenne Research, Inc. GIMPS history. http://www.mersenne.org/various/history.php.
[34] B.S. Naick, P.R. Kumar. Detection of low auto correlation binary sequences using meta heuristic approach. *International Journal of Computer Applications*, **106**(10): 32–37, November 2014.
[35] Pande Lab, Stanford University. Folding@home: Papers. https://folding.stanford.edu/home/papers/.
[36] K. Pearson [Ed.]. *Active Distributed Computing Projects*. http://www.distributedcomputing.info/projects.html.
[37] K. Piętak, M. Kisiel-Dorohinicki. Agent-based framework facilitating component-based implementation of distributed computational intelligence systems. *Transactions on Computational Collective Intelligence X*, pp. 31–44. Part of the Lecture Notes in Computer Science book series (LNCS, volume 7776), Springer, Berlin, Heidelberg, 2013. http://dx.doi.org/10.1007/978-3-642-38496-7_3.
[38] L.F.G. Sarmenta. Bayanihan: Web-based volunteer computing using Java. In: *Proceedings of the Second International Conference on Worldwide Computing and Its Applications*, pp. 444–461, 1998. http://groups.csail.mit.edu/cag/bayanihan/papers/wwca98/html/.
[39] L.F.G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, **18**: 561–572, 2002. http://people.csail.mit.edu/lfgs/papers/ccgrid-fgcs.pdf.
[40] H. Schnell, A. Szarvas, G. Molnár, I. Szeberényi. GridBee web computing framework. In: *The 7th BOINC Workshop*, 2011. http://boinc.berkeley.edu/trac/raw-attachment/wiki/WorkShop11/GridBee.pdf.
[41] TOP500.org. Top500 List – November 2015. http://www.top500.org/list/2015/11/.
[42] University of California. Choosing BOINC projects. https://boinc.berkeley.edu/projects.php.
[43] University of California. SETI@home Classic: In Memoriam, 2005. http://setiathome.berkeley.edu/classic.php.
[44] Wikipedia. List of distributed computing projects. https://en.wikipedia.org/wiki/List_of_distributed_computing_projects.
[45] D. Żurek, K. Piętak, M. Pietroń, M. Kisiel-Dorohinicki. Toward hybrid platform for evolutionary computations of hard discrete problems. *Procedia Computer Science*, **108**: 877–886, 2017. Part of the International Conference on Computational Science, ICCS 2017, 12–14 June 2017, Zurich, Switzerland. http://www.sciencedirect.com/science/article/pii/S1877050917307949.