Extensible volunteer computing platform

Grzegorz Jankowski, Roman Dębski, Aleksander Byrski
AGH University of Science and Technology Al. Mickiewicza 30, 30-059 Kraków, Poland
E-mail: jankowskigreg@gmail.com, roman.j.debski@gmail.com, olekb@agh.edu.pl

Abstract—**Existing Volunteer Computing environments are usually dedicated to one project/application and, what is very often unwelcome nowadays, require some software to be installed on each volunteer PC. Moreover, the environments are not completely platform-independent so it is not possible to use a fast growing mobiles computation potential. The idea of utilizing a web browser as an environment for executing a volunteer computational task addresses the last issue. The paper presents an extensible, component oriented volunteer computing platform that may be easily adapted to different problems. After reviewing existing solutions, the architecture and behavior of the platform is discussed, and simulation results are given.**

Keywords— **volunteer computing; distributed computing; computing framework**

INTRODUCTION

There are problems that require great computation power to be solved. However, purchase and maintenance of such devices is costly. On the other hand such problems can be decomposed and distributed to machines of less computation power. Example problems which are computing with this approach are: knapsack problem, traveling salesman problem and others belonging to NPC or NPH classes. In volunteer computing paradigm, computation problem is decomposed and volunteer participants compute the parts on their own computers.

There are existing software solutions that supports volunteer computing. Although popular, they have some drawbacks e.g. they are software or hardware dependent and they can not be used with mobile devices which popularity and computation power grows fast. Moreover, their components needs to be installed on participants' computers. Nowadays computer users, especially those non experienced, are not willing to install anything on their machines. On the other hand almost everyone has an Internet browser installed and know how to use it.

Taking all those aspects into consideration, it was decided to propose and implement a new tool supporting volunteer computing. It is a portable and universal platform which can be used either from an installed application or from an Internet browser. Additional motivation for this paper was to design the architecture of this solution to be extensible and flexible depending on the particular computation problem.

Platform architecture is designed as service oriented architecture (SOA). In order to implement this architecture, services development and hosting technology was needed. Moreover, technology should cooperate with different software and hardware providers because users' computers might be configured variously.

It was decided to implement the platform in the Microsoft. NET technology. It is a popular and robust software development environment that offers interesting opportunities for services development. Windows Communication Foundation technology as a part of the .NET environment, provides runtime environment and ensures communication between them.

In this study, theoretical issues related to the field of computing in distributed systems are presented first. The next chapter describes the platform overview, its architecture and implementation. Lastly, the course of experiments with the platform and the most important conclusions are presented.

PLATFORMS FOR LARGE-SCALE COMPUTATIONS

Many large-scale computations are performed using supercomputers. However, this approach has one big disadvantage – it has always been expensive. That is why computer clusters (as a cost-effective[1] 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], [14], *Volunteer Computing* [5], [1], *Cloud Computing* [10], [8], [13], [11], [2] or *Augmented Cloud Computing* [7], [4] can be utilized.

The platform presented in this paper utilizes the idea of volunteer computing based on web browsers. The main advantage of this approach is the hardware and software independency. A brief description of this concept is presented in the next two paragraphs.

A. Volunteer Computing.

Volunteer Computing[2] is a type of distributed computing where main computational resources come from the personal or office computers. The resources are shared by volunteers who want to support the specific computation.

Both in the past and at present there are many projects based on the paradigm of volunteer computing.The most famous are *SETI@Home*[3] (started in 1999) and the *Great Internet Mersenne Prime Search*[4] (started in 1996).

The most recognized frameworks for Volunteer Computing projects are Berkeley Open Infrastructure for Network Computing[5] (BOINC) (open source, base of SETI@Home), Xgrid[6] (a proprietary software prepared by and for Apple) or Grid MP[7] (a commercial product).

B. Web Browser Based Volunteer Computing.

Unfortunately, the existing Volunteer Computing environments are usually dedicated to one project/application[8]

[1] mainly because it is based on commodity hardware
[2] or a similar idea – *Sideband Computing* [15]
[3] http://setiathome.berkeley.edu/
[4] http://mersenne.org/
[5] http://boinc.berkeley.edu/
[6] http://www.apple.com/pl/server/macosx/technology/xgrid.html
[7] http://www.univa.com/
[8] And so, it cannot be considered even as a quasi-general purpose platform.

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

and, what is very often unwelcome nowadays, require some software to be installed on each volunteer's PC. Moreover, the environments are not completely platform-independent so it is not possible to use a fast growing mobiles computation potential. The idea of utilizing a web browser as an environment for executing a volunteer's computational task addresses the last issue. The idea was already implemented in the past using Java Applets as computational workers (e.g. [12], [6]) but in this paper an approach with JavaScript is utilized.

THE PROPOSED PLATFORM

One of the paper aims was to create a platform supporting distributed computations in volunteer computing paradigm. The platform should provide an infrastructure that can be used by volunteers regardless of their software and hardware configuration.

In the platform surrounding there is an external system that is performing a calculation algorithm. This system delegates such calculations to the platform by sending packages in an well-defined format. A volunteer user who wants to contribute to the calculation connects to the platform by a client program. The client program (e.g. web browser) gets the calculations to perform from the platform and sends the results back to the platform. Finally, the external computing system fetches computation results from the platform.

Simplified conceptual diagram of the platform interaction with surroundings is shown in the diagram 1.

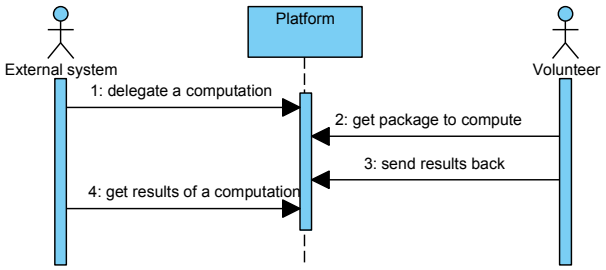

Fig. 1. Platform interaction with surroundings

C. Platform design and architecture

This section describes concepts and architecture assumptions of the platform as well as its implementation. The overall platform architecture is shown in the diagram2.

The external computation system that acts in the platform surrounding communicates with the platform by sending a computation requests or receiving its results in a well-defined package format. The package has a name (identification), data and an algorithm that is used to carry out the calculation. On the other side of the platform there are client programs run by volunteer users that contributes to the calculations. Clients download the packages, perform a calculations and send it back to the platform.

The platform consists of the three kinds of services: persistence services, package services, topology services. All the platform services acts in the pull model (download data from the other services).

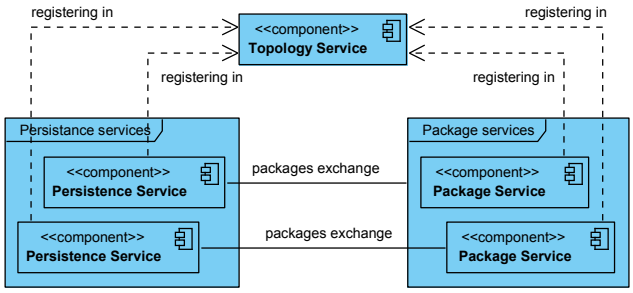

Fig. 2. The platform architecture

C.1 Interaction with surroundings

The platform provides services to its surroundings. The details of this collaboration are shown on the diagram 3.

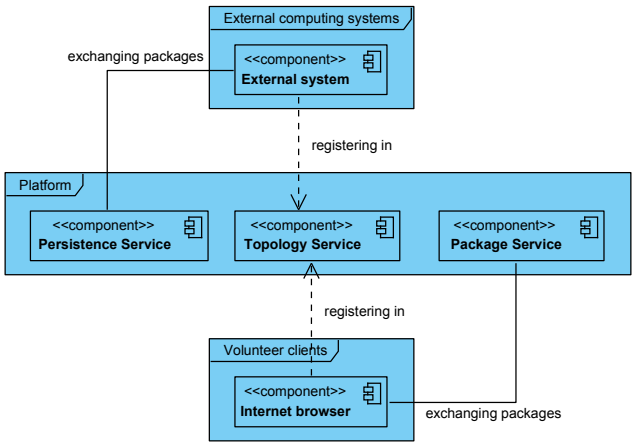

Fig. 3. Platform services interaction

A necessary condition to cooperate with the platform, both for the external computation systems, and volunteer clients is to register in the platform. It is done via topology service contract. Afterwards, an external computing system interacts with a persistence service, and clients with a package service.

C.2 Platform common structures

The section describes the platform common structures that are used by platform services. They are shown on the diagram 4.

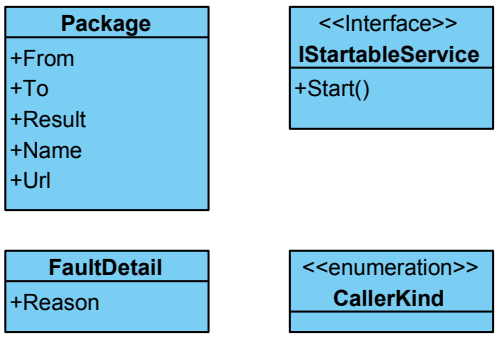

Fig. 4. Classes common for the platform services

Package class defines the data packet. It has a name, the number of the beginning and the end of the data interval, and the url to the library of the algorithm by which to perform the calculation.

FaultDetails class is responsible for providing information about errors.Information about the occurrence and cause of error between the web services platform are sent in the Reason property.

IStartableService interface is implemented by all the platform services. It allows to run the particular service.

CallerKindEnum enumeration indicates the topology service, what kind of service (or client) is registering.

C.3 Persistence service

Persistence service manages calculations delegated to the platform by an external computation system. The service provides the functionality to external systems as well as to the other platform services.

The service has two responsibilities in the interaction with external computation system.

Send a computation to perform to the platform - An external computation system sends the calculation in packages to the platform via the package service. Responsibility of the service is to store the packages for further processing.

Get a result of the computation from the platform - The service is responsible for sending results of the computation back to the external computation system.

The persistence service provides also two responsibilities for the other platform services.

Get a computation to perform - At the request of a package service it provides a package of calculations to perform. Package is shared only for a specific time and marked as shared. After the time expires, it will be marked back as available for download.

Send the result of the computation - Service receives the package with the calculation result and stores it until claimed by the external system. If the packages with the calculation results are sent back after the expiry of the time defined, it will be ignored. This behavior is intended to prevent from waiting for a package that will never be calculated (e.g. due to failure of another service or client).

Class diagram is shown on the diagram 5:

PersistenceService class implements service contract. It has a PackageProvider which responsibility is to provide packages for computations. FileRepository class manages packages serialized as files.

C.4 Package service

The service manages computation packages. It fetches them from a persistence service and provides to the volunteer clients. The package service does not have its own persistence. This implies that clients have just a certain time to perform computation and send the results back to the package service. Then the package service sends it back to the persistence service.

The service has two responsibilities for the volunteer clients.

Get package to compute - Volunteer clients who want to support a computation get packages via the service.

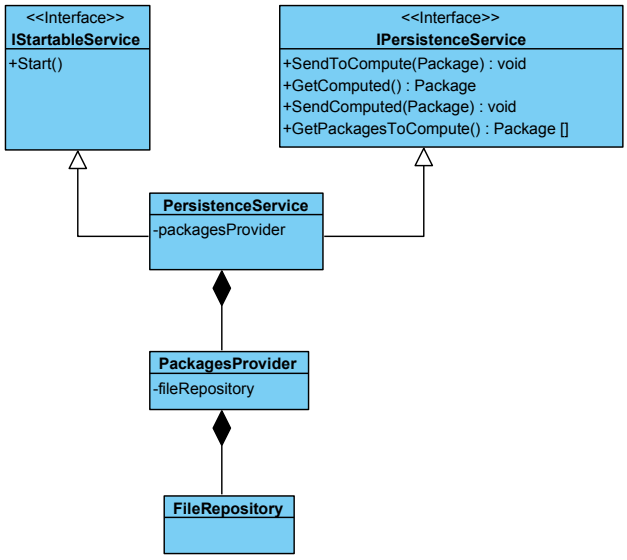

Fig. 5. Persistence service class diagram

Send computed package - After making computations, client sends the package with results back to the platform using the service.

Class diagram is shown on the diagram 6:

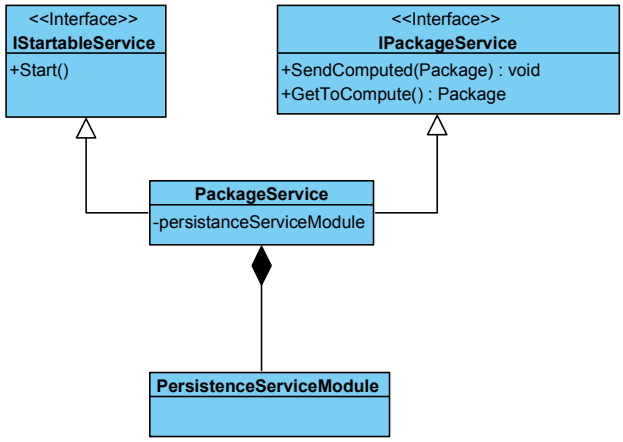

Fig. 6. Package service class diagram

PackageService class implements the service contract. It is so simple that the other classes were unnecessary.

C.5 Topology service

The platform is designed to work in a distributed environment. The topology service is responsible for an intelligent combining platform services with each other as well as clients and external computation systems in order to maximize the platform efficiency.

The service has a single responsibility.

Register - Every other platform service and the platform clients, at the time of its connection to the platform must register via the topology service. As a result the topology service knows about services and clients localization in the network and can manage the platform (e.g. load balancing). When registration is invoked, the topology service returns an address of the service which is the most appropriate for a

caller at the moment:

- For volunteer client: address of appropriate package service,
- For package service: address of appropriate persistence service,
- For persistence service: nothing is returned.

Class diagram is shown on the diagram 7:

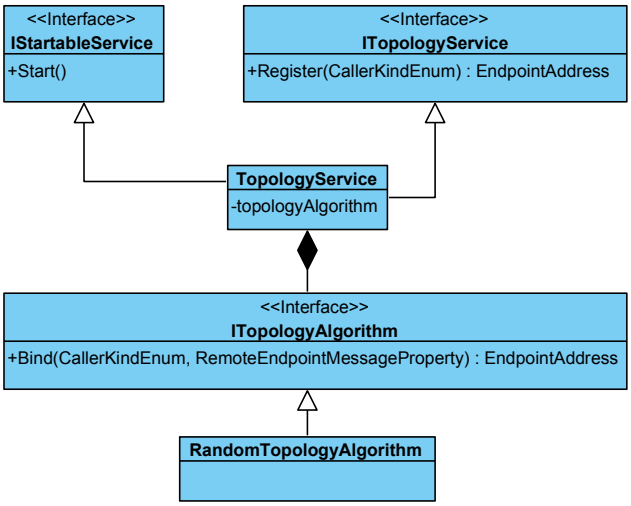

Fig. 7. Topology service class diagram

TopologyService class implements the service contract. Responsibility of the topology service to combine the platform components and the platform with its clients is delegated to a dedicated algorithm. This algorithm has to implement the interface ITopologyAlgorithm. An example algorithm implementation is RandomTopologyAlgorithm, which is combining randomly.

D. Platform deployment

The platform is designed to work in the distributed environment. However, it is also possible to install the platform entirely on one machine. In the further sections, the platform requirements and deployment process will be introduced.

The minimal platform configuration requires one topology service, one package service and one persistence service.

D.1 Requirements

The platform was designed to work on Microsoft Windows. The operating system should have Internet Information Services (IIS) version 6 or later installed. Moreover, .NET Framework 4.0 and Windows Communication Foundation (WCF) 3.5 are needed.

D.2 Installation

Installation of the platform is based on the installation of its services to IIS server through the supplied package (called deployment package).

The installation process should always be started from the topology service installation. It is the most important service from the platform initialization point of view because all the other platform services must register in it. The topology service address should be known within the platform and should be the single platform endpoint visible by the external systems and volunteer users.

EXPERIMENTAL RESULTS

This section will describe the experiments with the platform. The current implementation is only a proof of concept prototype so the empirical results will be provided showing platform potential and limitations.

E. Description and course of the experiment

The experiment consisted of finding prime numbers in a given numerical range.

A simple application imitating external computing system was created for the testing purpose. The application divided a numeric range 1 - 10 million into intervals of the equal count. The intervals boundaries were placed in the packages which were sent to the platform. The algorithm to find prime numbers in Java Script was also implemented. The algorithm was indicated to be executed on the packages data. The algorithm had the computational complexity of $O(n^2)$. The platform was responsible for the packages distribution to client machines. In the experiment participated from 1 to 16 computers. Each of them was running one Google Chrome web browser (version 19). Every computer participating in the experiment was identical to the hardware:

- CPU: Intel Core 2 Duo E6550 2,33 GHz
- RAM: 2GB

The platform which is the subject of research, has been placed, in full, on a single computer with hardware parameters:

- CPU: Dual - Core T2130 1,86 GHz
- RAM: 2GB

The platform was configured to have a single type of every service, which means one topology service, one package service and one persistence service.

F. Experiment results

The platform behavior was studied, depending on the number of compute nodes (computers with web browsers) and the degree of the problem decomposition on the intervals (number of packages to calculate sent to the platform). The time of calculations were measured in minutes started from the first package download from the platform by the client to the last calculated package returned to the platform. For each experiment configuration, three measurements were carried out, and the final result was the average of them. An important observation was, furthermore, that for any particular configuration the three measurements taken were very close to each other, differing at most by 1 minute.

G. Description of results and conclusions

The description of the experimental results, is divided into two parts: the platform support to the calculations in the volunteer computing paradigm and the platform limitations.

G.1 Volunteer computing calculations support

The idea behind the platform is to support the calculations in volunteer computing paradigm. The first observation is taken for the configuration where the proportions in a number of packages and the number of browsers are constant. In these configurations, each browser calculated exactly one data package.

With the increasement in the task decomposition on more packages and web browsers, a decrease in task processing time was observed. This is shown in figure 8. A calculation scenario for 1 package calculated on 1 computer (non-distributed computational task) the processing time was almost 3 hours. Distribution of the task resulted in significant decrease in computation time, up to only 19 minutes for 16 packages and 16 browsers. It was proved that the platform significantly supported the calculation by distributing packages.

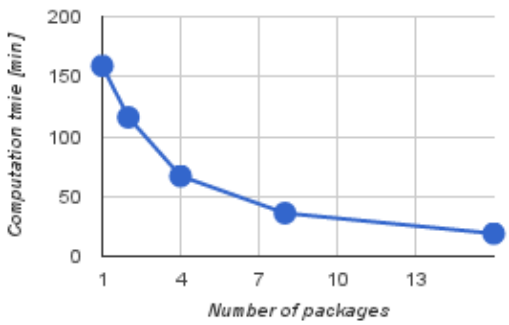

Fig. 8. The processing time for different amounts of web browsers, where each process one packet

The calculation time as an effect of the task decomposition degree was analyzed for a fixed number of 16 web browsers.

With increasement in the number of packages, calculation time initially decreased until it reached its minimum for 256 packages. For more packages, calculation time slightly increased and remained at a constant level. This is shown in figure 9. There are two facts relevant to explain this behavior: first, the computational complexity of the algorithm used was $O(n^2)$, so reducing the data set on which it operated gave significant gains. Second is to consider the overheads associated with communication within the platform and with client machines.

To achieve the minimal computation time - 10 minutes for 256 packages, the profits associated with an increased decomposition of the problem, outweighed losses resulting from the communication overhead. Not only the further increase in the number of packages did not reduce the computation time but even increased it for a one minute, because of the cost of communication. The analyzed results, allow to draw conclusions about the platform communication overhead. The interesting case, showing the platform limitations is described in section -G.2. It turned out that the communication overhead is at some point so significant that the platform can not keep up with it.

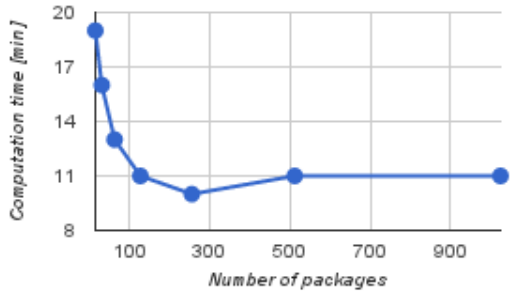

Fig. 9. Processing time for 16 web browsers

G.2 Platform limitations

In the experiment, there was an attempt to show the limit of the packages number when not only the calculation time is not decreasing, but is actually increasing due to communication overheads within the platform.The experiment was performed for 16 web browsers. As described earlier, the optimal number of packages for the investigated problem calculated on 16 compute nodes, was 256. For more packages calculation time slightly increased. However, for 2048 packages, the experiment failed. For this number of packages only 9-10 web browsers succeeded in registering in the platform. The rest have failed to register and start to process, due to the intense platform communication with the other client machines. Such a large number of packages, meant that in a small unit of time, the platform faced too many requests and it was not able to handle them all. This resulted in time limits exceed on requests and their rejection as a result. Prospects for the platform development in the direction of reducing the communication are presented in the summary of the article.

CONCLUSION

The aim of this article was to introduce a platform supporting calculations in volunteer computing paradigm.

The introduced solution met the assumptions. This was confirmed by experiments carried out (see. -E). It was shown that the platform supported the computation task performed in volunteer computing paradigm, by reducing the time of its calculation. The example calculation performed on a single computer lasted 159 minutes. After the calculation distribution by the plaatform to the 16 computers the calculation time decreased to 19 minutes. Therefore more than eight-fold decrease in the calculation time was achieved. The current implementation is most appropriate for coarse granular problems, i.e. in which the computation-to-communication ratio is relatively high.

The platform has a number of possible development direction. Two main trends are the expansion of the experimental research directions on its capabilities and the platform development.

In the experiment the platform consisted of a single topology service, one package service and one persistence service. The platform is designed to allow multiple services of the same type. Thus, another area of research may be different internal platform configurations. For example, with

more package services probably more compute nodes would get connected to the platform (see -G.2).

In the experiment carried out the platform is set up and run in its entirety on a single computer. The platform is a distributed solution itself. The individual components (services) can be placed on different nodes. By distributing the platform implementation the communication impact can be observed not only between the platform and the computational nodes, but also within the platform.

The main experiment to perform in the future, is to examine the platform performance on the Internet. Calculations in volunteer computing paradigm are usually conducted in the global network where many potential users are willing to support the project. In this experiment the impact of Internet communication and possibly other problems, such as firewalls might be examined.

Finally, it is worth to compare the performance of the platform with some existing environments to see advantages and disadvantages of the introduced approach.

The second proposed course of further work is the platform development.

An important component that might be improved is a topology service. Currently, the algorithm that binds platform services to its clients does it in the random way (see -C.5). If experiments are performed according to the scenarios proposed above, in which the configuration of the platform is more complex (there are more services), the algorithm will play a key role in the load balancing. It is therefore necessary to implement a family of algorithms and test their impact on the platform performance.

Limitations of the platform due to the excessive communication (see -G.2) can be tried to resolve in the development matter. In the current version of the platform when a package service receives a request from the client willing to calculate the package, it sends the request to a persistence service. Only after receiving a package from a persistence service, package service sends it to the client. The proposed solution is to introduce a buffer in the package service that will fetch more packages from persistence service than it needs at the moment. As a result, package service would provide packages for clients without the each time additional communication with the persistence service.

ACKNOWLEDGMENT

The research presented here was partially supported by the grant "Biologically inspired mechanisms in planning and management of dynamic environments" funded by the Polish National Science Centre, No. N N516 500039.

REFERENCES

[1] David P. Anderson, Eric Korpela, and Rom Walton. High-performance task distribution for volunteer computing. In *Proceedings of the First International Conference on e-Science and Grid Computing*, 2005.

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

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

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

[5] Aleksander Byrski, Michal Felus, Jakub Gawlik, Rafal Jasica, Pawel Kobak, Grzegorz Jankowski, Edward Nawarecki, Michal Wroczynski, Przemyslaw Majewski, Tomasz Krupa, and Jacek Strychalski. Volunteer computing simulation using repast and mason. *Computer Science*, 14(1), 2013.

[6] Bartosz Czerwiński, Roman Dębski, and Kamil Piętak. 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.

[7] Roman Dębski, Aleksander Byrski, and Marek Kisiel-Dorohinicki. Towards an agent-based augmented cloud. *Journal of Telecommunications and Information Technology*, (1):16–22, 2012.

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

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

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

[11] Jeffrey Napper and Paolo Bientinesi. Can cloud computing reach the top500. In *UCHPC-MAW'09*, 2009.

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

[13] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. In *ISPAN 2009*, pages 4–16. IEEE Computer Society, 2009.

[14] Stephen J. Wright. Solving optimization problems on computational grids, 2000.

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

ABOUT THE AUTHORS

Grzegorz Jankowski obtained his M.Sc. in 2012 at AGH University of Science and Technology in Cracow and right afterwards started Ph.D. studies. He is interested in large-scale computing, in particular in volunteer model.

Roman Dębski is an independent consultant with over 14 years of experience in IT. He holds MSc in Computer Science (AGH University of Science and Technology) and in Mechanical Engineering and a PhD in Computational Mechanics (both obtained at Cracow University of Technology). His current research focuses on the cloud computing.

Aleksander Byrski obtained his Ph.D. in 2007 at AGH University of Science and Technology in Cracow. He works as an assistant professor at the Department of Computer Science of AGH-UST. His research focuses on multi-agent systems, biologically-inspired computing and other soft computing methods.