Future Generation Computer Systems 29 (2013) 1442–1451

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards a volunteer cloud system

Attila Marosi, József Kovács *, Peter Kacsuk

MTA SZTAKI, Kende u. 13-17, 1111 Budapest, Hungary

ARTICLE INFO

Article history:
Received 16 January 2012
Received in revised form
8 March 2012
Accepted 16 March 2012
Available online 27 March 2012

Keywords:
Volunteer computing
Cloud
Virtualization
Desktop grid

ABSTRACT

The paper completes the work started in the EU FP7 EDGI project for extending service grids with volunteer (global) and institutional (local) desktop grids. The Generic BOINC Application Client (GBAC) concept described in the paper enables the transparent and automatic forwarding of parameter sweep application (parametric) jobs from service grid VOs (Virtual Organizations) into connected desktop grids without any porting effort. GBAC that introduces virtualization for the volunteer BOINC (Berkeley Open Infrastructure for Network Computing) clients can also be considered as a first step towards establishing volunteer cloud systems since it provides solutions for several problems of creating such a volunteer cloud system.

1. Introduction

The EU FP7 EDGI [1] project investigates how to combine desktop grid (DG) systems like BOINC [2] and XtremWeb [3] with clouds in order to improve response time of volunteer DG systems. Another important research direction is virtualization in order to facilitate application porting for the volunteer DG systems. The on-demand extension of volunteer DG resources with cloud resources was reported in [4]. In the current paper we would like to report results of the second research direction and show how virtualization can be used to eliminate application porting efforts for BOINC systems. In EDGI we support service grid (gLite, ARC, UNICORE, etc.) virtual organizations (VOs) to extend their limited number of resources with volunteer or local DG resources. The service grids typically run parameter sweep (PS) applications. If we want to support them with DG systems it is vital to enable the execution of the PS applications on the connected DG systems. Without virtualization every PS application required a porting effort and hence the service grid (SG) VO users were not interested in the DG extension of their VOs.

Currently BOINC requires a large porting effort to adapt an application to the BOINC middleware. Although tools like DC-API [5] and GenWrapper [6] significantly reduced the porting effort it still requires some BOINC skill. More than that, in a volunteer environment the ported application should be compiled for every possible kind of client operating system and version. This requires a huge and tedious further effort that distracts application developers from adapting their applications to BOINC. Finally, BOINC can run only those applications that are registered in the BOINC server and hence the flexibility of service grid systems (like Globus, gLite, ARC and UNICORE) where scientists can submit any kind of PS application is completely missing in BOINC.

In the framework of the EU FP7 EDGeS [7] and EDGI [1] projects of more than 30 applications have been ported to BOINC. Although this is a significant number of applications there are still many other parameter sweep applications, typically simulations that could advantageously be run on BOINC but would require significant porting effort. In order to avoid future porting efforts a new framework, called GBAC (Generic BOINC Application Client) has been developed in the EDGI project.

The main idea of GBAC is that instead of porting and registering applications individually a generic virtualized application (GBAC) is ported and registered in BOINC. Whenever a PS application is to be executed the number of resources can be extended on-demand as in clouds by submitting a large number of tasks generated from the parameter sweep application to the BOINC server as inputs for the GBAC application. For every PS task a GBAC work unit will be generated and executed by one of the BOINC clients.

Notice that in this approach only the GBAC application is registered in the BOINC server and this application can accept any PS application as a parameter. In this way any PS application running in a service grid can be transparently forwarded to and executed in a BOINC system that extends the service grid VO. As a result EDGI has achieved its original goal: service grids can be extended with large DG systems and the service grid users can transparently run PS applications in the integrated SG-DG system without any application porting effort.

* Corresponding author. Tel.: +36 13297864; fax: +36 13297864.
E-mail addresses: atisu@sztaki.hu (A. Marosi), smith@sztaki.hu (J. Kovács), kacsuk@sztaki.hu (P. Kacsuk).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.03.013

This solution shows many similarities with SaaS clouds in the sense that the SG systems are extended with new resources on demand. The main difference is that these additional resources are collected from a volunteer system and hence the users do not have to pay for the use of these resources. The volunteer resources are rewarded by a virtual credit system and the collected credits can be used in EDGI to access real cloud resources when response time is critical for the PS application. This feature of EDGI will be described in detail in a forthcoming paper.

The availability of huge amounts of volunteer resources and the introduction of virtualization on these desktop grid resources resulted in a natural continuation towards volunteer clouds within the EDGI project. Volunteer cloud can be a new step in the evolution of the cloud paradigm, where the cloud resources are provided by the volunteers. Of course, volatility and availability issues in a desktop grid environment cause significant difficulty, however several research activities are on the way to provide solutions. Currently, research directions towards reliable services on unreliable resources show promising results, where redundancy is one of the key factors in the proposed solutions. In this paper, we introduce GBAC also as a step towards introducing the volunteer cloud paradigm.

In the current paper we show the details of the GBAC solution. Section 2 describes the main challenges for applications in volunteer computing. Section 3 explains the problems that arise when a volunteer cloud system should be created. Section 4 overviews and compares the development APIs and tools for volunteer computing. Section 5 gives the detailed description of the GBAC concept and implementation both at the server and client side. Finally, Section 6 reports some related work in the area of virtualization in volunteer DG systems.

2. Challenges for applications in volunteer computing

In volunteer computing there are several key areas where application developers and project maintainers are facing problems. First, in volunteer computing, applications can arrive from various scientific areas, therefore application executables/implementations can be of wide range. The supported types of applications are master–worker, parameter sweep and bag of tasks type of applications where communication is not needed among the worker nodes. In all cases the BOINC server distributes work units to the clients that process the work units independently from each other.

Based on the implementation method one group of applications can be called native. Native applications are the ones prepared to run on desktop grid/volunteer computing resources by invoking some API calls to realize the necessary (but minimal) integration. The other form of implementing applications for BOINC is wrapping. In case of wrapped applications the source code of the application is not modified at all and hence this approach of porting applications is particularly important for the legacy applications.

The porting activity can take a significant amount of time for some applications. There are, however, several libraries for this purpose. For legacy applications where source code is not available the developer can use some wrapping mechanisms. This is, however, still not automatic and requires human effort. Another important task that has to be done manually is to implement the checkpoint/restart functions for the applications. Since on desktop or volunteer resources execution is preempted from time to time, applications must be prepared for saving and restoring their internal status. For native applications, API calls are provided to notify the application, while in the case of wrapped applications notification is not provided.

In volunteer computing the typical number of client resources is in the range of tens or hundreds of thousands, therefore it is natural that they show a high diversity related to their platforms (Windows 32/64 bit, Linux 32/64 bit, Mac OS X, etc.). Each of the different platforms requires its optimized version of the application executable. For native applications, optimization can be a tedious (compilation) task while for wrapped applications it is impossible in most of the cases.

In volunteer computing volunteers (clients) basically trust in the volunteer project. To keep this trust, applications have to be validated (tested substantially) in order to make sure they will not harm the owner's machine even in case of malfunctions or invalid inputs. This testing has to be done for each platform and unfortunately no automatic method is known currently to perform these validation activities to provide secure applications. To perform secure execution, volunteer software (like BOINC) provides volunteers with the possibility to define resource limits for the given project/application. Currently, a mechanism is implemented in the client software to monitor the resource usage of the executed application and in case of over usage get rid of it. However, the best solution would be the total isolation of the running application in order to make sure it cannot cross the boundaries of its predefined environment.

Deployment of desktop grid applications is done by the volunteer software by simply copying the necessary files as a non-privileged user to the volunteer machine. This implies that pre-installing other dependencies (libraries, etc.) is not possible or hardly solvable. In a virtualized environment it would be much easier to prepare all the necessary dependencies for the application.

The aforementioned problems highlight those important areas where the virtualization provided by our GBAC can help in executing applications in volunteer computing environments. The next sections will provide the details for the reader.

3. Volunteer cloud

Virtualization is the mechanism when a logical representation is created on top of the real, physical software or hardware component(s). Virtualization, for example, enables decoupling software services and their resources, i.e., separating the actual resources (CPU, storage, and network) from the physical hardware. Virtualization provides more flexibility for maintenance and improves the utilization rate of the physical resources.

Introducing virtualization techniques substantiated a new paradigm, called Cloud. Cloud Computing builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Computing resources – based on the virtualization techniques – are logically provided for the services and dynamically assigned to the physical resources on the fly. Any running service therefore can be served by changing the amount of resources (CPU, storage and network) in time. In this new environment accounting is based on the pay-as-you-go approach since users can define the required amount of resources and have to pay only for the actually used cycles. A cloud infrastructure is typically deployed on a large number of dedicated machines, where the required portion is dynamically assigned for services and users on demand. However, it may happen especially in small size academic clouds that the system runs out of resources during an intensive utilization period. There are alternatives for handling this situation, but one of them is to include volunteer resources.

Contrary to the usual service grid infrastructures where complex middleware is used to manage the connected dedicated resources, desktop grids also use a lightweight middleware to attract volunteer resources. The most well-known volunteer desktop grid is SETI@Home [8] that is collected from about 3 million CPUs from worldwide. This desktop grid is based on the

BOINC platform [2] which is the most popular volunteer computing platform currently.

BOINC was developed with volunteer computing in mind, thus its architecture is aiming at utilizing the spare cycles of home PCs. It uses a server–client approach where a centralized component is queried for units of work (work units) by its clients. BOINC server deployments are independent from each other and are referred to as projects. Each project aims to achieve a different scientific goal ranging from drug discovery to distributed rendering. Each project typically runs one application. Resources can be donated by downloading a lightweight client and connecting it to the public URL of a specific project. The resource donating persons, referred as donors, are rewarded with virtual credits for the work done by their computers. Credit for a work unit is awarded based on the amount of CPU time spent for computation. Malicious results from unreliable resources are filtered out by using, for example, redundancy and/or validation. Usually, a BOINC project aims at solving a grand-challenge scientific problem that requires the project owner to provide an attractive appearance. This can include dissemination of scientific results, maintaining and serving donors, implementing fancy screen savers, and so on. The more attractive the project is for the citizens, the more resources the project can collect. Huge volunteer projects like SETI@Home can collect millions of machines that are, however, unreliable and only partially available.

Volunteer cloud is the term used for a cloud type infrastructure based on volunteer resources. Thanks to the combination of BOINC and virtualization, now it is possible to launch virtual machines on volunteer resources. BOINC primarily supports VirtualBox [9], which means any client machine having preinstalled VirtualBox can participate in executing virtual machines.

In this volatile and unreliable environment deploying an Infrastructure-as-a-Service cloud infrastructure faces several challenges. The first challenge is motivating donors not just to donate their CPU and GPU resources but to provide more access to their computers. Any service running on a volunteer host would require network access and disk access as well as CPU cycles. Traditionally, motivation is solved in volunteer computing by a credit-reward system where donors receive virtual credits for their total contributed CPU/GPU cycles for each successfully computed task. For volunteer clouds a composite reward system would be required which takes into account the different donated resource types (e.g. network, disk, CPU and GPU cycles) and also the reliability of the hosts. Donors should be motivated to donate their resources in a more reliable way, e.g., a CPU intensive job may checkpoint itself periodically, so it will be able to finish even if it can execute only in many shorter "bursts". Unfortunately it is not acceptable for many services to be run in this way.

Compared to a "typical" volunteer computing resource usage, participating in a volunteer cloud would mean increased utilization of network and disk. One critical component in this is downloading virtual machine images. A service combined with an operating system and different libraries are contained inside each virtual appliance and these need to be deployed on the volunteer resource before use. The images' sizes can vary from a couple of hundred megabytes to tens of gigabytes making management and deployment a major challenge. Usually, volunteer resources have a more limited network connection so the bandwidth requirement for image downloads should be reduced.

In volunteer computing most donated resources are home computers sitting behind routers providing Network Address Translation (NAT) for the local network and firewalls. Accessing these resources or virtual machines on these resources from the public internet is not possible. VMs run on (virtual) private networks thus routing incoming network connections to them is problematic. Appliances should contain networking facilities and donors should have the possibilities to allow/disallow network access for appliances they run. Also some donors don't want to or don't have the technical skill of configuring a network for appliances so there should be an automatic mechanism for this. One possible solution is to use Universal Plug and Play (UPnP) to configure the firewall and NAT.

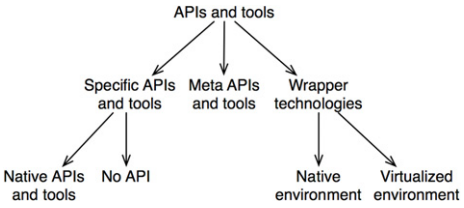

Fig. 1. Classification of available APIs and tools.

Volatility and availability problem of volunteer resources is another major challenge in volunteer computing. Since any volunteer host may be shut down any time by either the donor or as result of technical issues (e.g., power or hardware failure) fault tolerance of appliances becomes a key challenge. In volunteer computing this is mitigated in two ways. First, redundancy is used, meaning more than one resource handles the same task and if one fails the other(s) are still able to produce the result. Second, applications periodically checkpoint themselves, in case of a restart the application can continue from the last checkpoint. Overall, checkpointing and redundancy has to be carefully designed and optimized for network bandwidth. Moreover, forecasting the availability of the volunteer hosts is also a key factor to increase the efficiency of the service migration.

To summarize, building a volunteer cloud requires additional research on the related areas. Please note that this paper will not detail a full solution for the challenges mentioned above, but will try to give some insight into the current achievements towards deploying volunteer clouds, i.e., how to run virtual machines on volunteer resources and how this eliminates the application porting effort required in volunteer BOINC systems without virtualization.

4. Development APIs and tools for volunteer computing

Before they can be deployed in a volunteer BOINC system, applications usually require specific modifications due to the idiosyncrasies of the middleware. In this section we will categorize the available APIs and tools helping application porting and show how support for virtualization (of the execution environment on the clients) is possible, which represents a basis for volunteer clouds.

The available application porting APIs and tools can be categorized in three main classes as shown in Fig. 1. The first category consists of middleware specific APIs and tools. Every middleware usually provides a set of tools and APIs for application developers. These are available for specific programming languages that the application developer has to use. For example the native BOINC API supports developing applications in C, C + +, FORTRAN and Python with partial support for Java. The native API is usually used when developing applications from scratch or when the complexity of existing applications makes porting infeasible. This also implies that the source code of the application is available and the programming language in which the application is written is supported. A special case is when the middleware does not provide (require) any APIs for application development.

In the second category the "Meta APIs" (e.g., DC-API [5] and PyMW [10]) are applied. These support developing applications for multiple middleware. They hide the specifics of each supported

middleware and provide a single common set of features. For example, DCI-API supports both, BOINC and Condor as well as other middleware and hence the same application instrumented with DC-API can run on both without any changes to the code. DC-API supports developing applications in C, C + +, Java and Python. In DC-API each application consists of two parts: a master and a client part. The master part is responsible for generating work units (e.g., by dividing a problem set into smaller pieces), submitting them to the specific middleware and processing the returned results, while the client part processes the pieces of the problem set. PyMW is a similar general purpose master–worker parallel computation framework but it supports only Python and targets the non-expert users. To summarize, the benefit of Meta APIs is that the same application code can be compiled and deployed on different distributed computing platforms without any modifications to the source code when it is moved to another middleware. The drawback is that recompiling might be required and fine grain tuning of the application for a given middleware is usually not possible.

In the third class the different wrapper technologies are applied. These provide a shell around the application which handles all specifics required by the middleware. Wrappers are typically used for rapid prototyping or when the source code of the application is not available (legacy application) or when API-based porting is infeasible for some other reason (e.g., complexity). In case of BOINC there are different wrappers available: a standard BOINC wrapper that is able to execute legacy applications in a sequential order; a generic wrapper called GenWrapper [6] that uses a POSIX like shell scripting language for describing control flow, and thus suited for more complex tasks that consist of many legacy binaries. Wrappers ease the burden of porting but the final result is less seamless adaptation, e.g., BOINC only supports application level checkpointing, thus each application has to implement its own mechanism. For legacy applications such application level checkpointing is not possible in the general case. It is only possible if the application consists of several binaries that are executed sequentially. In this case a checkpoint can be made after finishing the execution of each binary. The main benefit of using wrappers is that less effort is required to port existing applications to a new middleware but in return they do not necessarily allow the applications to exploit all the advantageous features of the middleware.

In the following subsections we discuss Table 1 where we defined four groups of criteria for comparing tools and APIs: (1) application development, (2) execution environment, (3) security issues and (4) deployment. Using these criteria we compare the native BOINC API, the DC-API (Meta API) and three available wrappers: the standard BOINC wrapper, GenWrapper and finally our virtualization based approach. VBoxWrapper and GBAC are both virtualized environment providing wrappers, thus we do not list them both, but we detail the difference between them in Section 5.

4.1. Application development

The first criterion (see 1.a in Table 1) of the application development specifics group deals with the supported programming languages of the API (or tool). The ideal case is when there is no restriction for languages. In case of the APIs the criterion is obvious, the more languages are supported the better. For wrappers there is no such restriction and applications can be developed using any languages (the source code is not required), thus the supported programming languages criteria has a different intent there, the language is usually used to describe the control flow for the execution of the legacy binaries and to do pre- and post-processing for the legacy binaries. BOINC provides a C/C++ API with FORTRAN bindings and a Python API, but traditionally applications are developed using the former one. DC-API provides a C/C++ API as well with Java bindings using the Java Native Interface (JNI). The BOINC wrapper uses an XML like syntax for describing control-flow: executables can be defined, which will be executed in sequential order, but no pre- or post-processing functions can be described. This wrapper is aimed at legacy applications which consist of (preferably) a single binary and well defined inputs and outputs. In contrast GenWrapper is aimed at legacy applications that require more complex control-flow or even at implementing a workflow with multiple (legacy) binaries. It uses a POSIX like shell scripting language and provides a set of standard Linux commands (e.g., sed, awk, tar, cut, unzip, gzip). In the case of GBAC virtualization lifts any limitations for development tools or APIs. A virtual appliance can contain any scripting language interpreter and there is no need to use any middleware specific or Meta API for development, since the application will be run in (a) a standardized environment or (b) in the standardized environment adapted to the requirements of the application.

The second criterion (see 1.b in Table 1) lists the supported middleware. The native APIs obviously support only a single middleware while allowing fine grain tuning of the applications, In case of the BOINC API the developed application can be also run without the BOINC environment in a "standalone" mode, which allows easier debugging (and possibly executing the application on another middleware). DC-API supports Condor and XtremWeb beside BOINC, thus applications developed only need to be recompiled to be used on a different middleware. However this comes with a cost, fine-grain tuning of the applications is not always possible, e.g., setting the fine details of the redundancy parameter of BOINC was only recently introduced to DC-API. Although wrappers hide the specifics of the middleware they are developed for a specific or multiple middleware. The BOINC wrapper uses the native BOINC API and thus is suited for BOINC only. GenWrapper and GBAC utilize DC-API and thus can be deployed on all middleware DC-API supports.

The third and fourth criterion (see 1.c and 1.d in Table 1) detail whether the tool or API is suited for legacy and native (non-legacy) applications. A native application is either (a) a new application written from scratch using some API that supports the middleware or (b) an existing application with available source code that is feasible to port. Porting is only possible if the chosen API supports the programming language which was used to develop the application. In some cases using native or Meta APIs and thus developing native applications is not feasible: (a) there might be already an implementation of the application for a different middleware, (b) the complexity of the application makes middleware specific modifications hard or infeasible, or (c) the source code is not available. A possible solution is that an application specific wrapper can be written for each application. In case of BOINC these wrappers could be written both using the native API or a Meta API (e.g., DC-API). However writing such specific wrappers seems needless in most cases, and generic wrappers are easy to use or adapt. The BOINC Wrapper is suited for simple legacy applications (e.g., single binary), while GenWrapper is aimed at more complex use cases. However it is still hard or even impossible to run legacy applications which have complex dependencies (e.g., require special libraries to be installed or a specific version of an operating system) via these wrappers. On the other hand native applications can be executed via wrappers as well, however there is very little reason doing so for the BOINC Wrapper and GenWrapper. In contrast GBAC and its virtualized environment provide additional benefits (e.g., isolation, resource usage limits and standardized environment) which make executing native applications feasible and complex legacy applications possible.

Table 1
Comparison of the native BOINC API, DC-API, BOINC wrapper and GenWrapper.

	BOINC API	DC-API	BOINC Wrapper	GenWrapper	GBAC
1. Application development					
a. Supported programming languages	C/C++/ FORTRAN/ Python	C/C++/Java/ Python	Control-flow description in XML	POSIX shell scripting	*None required*
b. Supported middleware	BOINC/ Standalone	BOINC/Condor/ XtremWeb/ Standalone	BOINC/ Standalone	BOINC/Condor/ XtremWeb/ Standalone	*BOINC/Condor/ XtremWeb/ Standalone*
c. Legacy application support	No	No	Yes	Yes	*Yes*
d. Native application support	Yes	Yes	Partial	Partial	Partial
e. Application level checkpointing	Yes	Yes	Partial	Partial	Partial
2. Execution environment					
a. Type	Native	Native	Native	Native	*Virtualized*
b. Support for complex application dependencies	No	No	Partial	Partial	*Yes*
c. Process/system level checkpointing	No	No	No	No	*Yes (system-level)*
3. Security					
a. Isolation (sandbox)	Partial (middleware, OS)	Partial (middleware, OS)	Partial (middleware, OS)	Partial (middleware, OS)	*Yes (hypervisor)*
b. Resource usage limit enforcement	Partial (middleware)	Partial (middleware)	Partial (middleware)	Partial (middleware)	*Yes (hypervisor)*
4. Deployment a. Requires server-side application deployment	Yes	Yes	Yes	Yes	*No*
b. Same application binaries can be used for all platforms	No	No	No	No	*Yes*
c. Supported major platforms	Windows/Linux/ Mac OS X	Windows/Linux/ Mac OS X	Windows/Linux/ Mac OS X	Windows/Linux/ Mac OS X	*Windows/Linux/ Mac OS X*
d. Requires client-side third party software	No	No	No	No	Yes (VirtualBox predeployed)

Another challenge is caused by the volatility of resources on which volunteer computing relies on. These resources can be turned off any time and the tasks they are executing are interrupted before they finish. If they are not prepared, the computation (work) they performed might be lost. The easiest way to overcome this is by introducing periodic or triggered checkpointing, which can be implemented at different levels. The last criterion of the group (see 1.e in Table 1) details whether the lowest form (application level) is supported. BOINC natively supports only application level checkpointing which means that all native applications have to implement their own logic to do that. The application should (a) periodically query the middleware whether there is time to checkpoint (triggered) or (b) call its checkpointing logic whenever it sees fit (periodic). In case of DC-API a similar approach is used but for middleware where there is no possibility for querying the middleware (e.g., XtremWeb) its internal logic decides. Legacy applications might have some internal checkpointing logic implemented but our experience shows that they usually do not. Wrappers cannot checkpoint the application itself; the only possibility is when the application executes multiple binaries sequentially. In this case a checkpoint can be introduced after each execution. This is how the BOINC Wrapper and GenWrapper supports application level checkpointing, although the former does it automatically while in the latter the control-flow script has to implement it (there are reusable samples available to do it). In the GBAC application, level checkpointing works similarly, although virtualization makes application level checkpointing obsolete.

4.2. Execution environment

This group aims to characterize the execution environment the tools provide and the developed applications used. The first criterion defines the type of the execution environment (see 3.a in Table 1), that can be either native or virtualized. Volunteer resources can be considered as resources with diverse hardware capabilities (e.g., memory, disk and CPU) on which a diverse software stack (operating system, installed libraries) is deployed. The native environment does not change this and the developer must prepare the application or service in a way that it (a) has no or minimal external dependencies, (b) it has binaries for most of the major operating systems and architectures (cross-platform). Normally these steps are not considered as part of the task of any API or wrapper technology, it is largely dependent on the given application and it is achieved usually at compile-linking time. However this is still a challenge posed to the execution environment. The only solution for native environments (BOINC API, DC-API, BOINC Wrapper and GenWrapper) is to bundle external dependencies and possibly compile statically executables if possible (see (a)), and to repeat this for each major platform supported (see (b)).

On the other hand virtualized environments can be considered as standardized environments with a given architecture, operating system and software stack. Developers can more easily develop their applications and services for this environment or in case of legacy applications the environment can be customized with the dependencies and requirements of the application. The second

criterion (see 2.b in Table 1) details exactly this, whether there is any support for dealing with complex dependencies and requirements of applications. The native BOINC API does not support this since native applications are developed usually from scratch and the developers can design their applications in a way that external dependencies are minimized and are cross-platform. In the case of legacy applications, this is not possible and the wrappers that provide a native environment have limited support for this: it is possible to include external libraries with the application or service, but only to some extent.

The final criterion in the group (see 2.c in Table 1) details whether there is any support for higher-level checkpointing provided by the API or tool. This can be either at process level (e.g., Condor supports this) or at the system level. In the case of BOINC only application level checkpointing is available without any external support. This is also true for DC-API (regardless of the supported middleware). Neither the BOINC Wrapper nor GenWrapper have support for higher level checkpointing, while GBAC provides system level checkpointing through the use of virtualization (see details in Section 5).

4.3. Security

In this group we detail two security aspects. The first aspect is isolation (see 3.a in Table 1). In volunteer computing the donors trust that the application they are executing on their resources will not perform any malicious activity (e.g., use network without permission, install backdoor, access personal data). To ensure this there are several mechanisms provided by the operating system on which the client can rely on (e.g., less privileged accounts, sandboxes). This criterion checks whether there are any isolation enforcements applied for an application using a specific solution. For example, BOINC runs under two less privileged accounts; the first one which has the more privileges is for the BOINC client software while the application is executed under an even more restricted account. This is always applied regardless of what API the application was developed with or which wrapper it uses. No previously used approach offered more isolation than the other, all of them provided some partial solution. However, GBAC significantly improves isolation through the use of a hypervisor.

The second aspect of security is resource usage limit enforcement (see 3.b in Table 1). This criterion has synergy with the previous one since the middleware is responsible to ensure that no running application uses more resources (CPU, disk and memory) than it is allowed to and terminates if it is required. The BOINC client continuously monitors the running applications and acts as needed but it could be possible that a malicious application escapes this supervision (e.g., by forking itself a large number of times). This is impossible in case of GBAC where the applied hypervisor can prevent such malicious activities.

4.4. Deployment

In this group we detail the different aspects of deployment on the middleware for applications and services developed using the listed tools and APIs. The first criterion (see 4.a in Table 1) describes whether it is required to deploy the application or service on the BOINC server before any client can download and execute it. The deployment requirement stems from the architecture of BOINC and thus it is either lifted by the tool (or API) or left intact. The native BOINC API is only an API for application development and as such cannot affect the requirements of the middleware.

The BOINC Wrapper and GenWrapper are both "single purpose" applications meaning that for each application specific parts have to be written (XML description or control-flow description) and deployed independently. On the other hand GBAC serves as a "multi-purpose" application.

All legacy applications are executed via GBAC and thus there is no need to deploy them. The legacy application binaries and inputs are input files for the GBAC application.

The second criterion (see 4.b in Table 1) describes whether application binaries for a single platform (e.g., Linux) are enough to support all major platforms. This is a direct result of the type of the execution environment since if there is a single standardized environment only a single virtualized platform needs to be supported. This means that only GBAC is able to fulfill this criterion.

The third criterion (see 4.c in Table 1) shows which platform the tool or API supports natively. We included this one to show that all tools and APIs support all major platforms.

Finally the last criterion (see 4.d in Table 1) describes whether there is any 3rd party software or library required by the tool or API at the client machine. We do not consider middleware related components (e.g., the BOINC Client) as requirements. Only GBAC has such a requirement, namely VirtualBox should be deployed on the client and should be accessible by the BOINC Client software and by the applications it is executing.

5. GBAC: Generic BOINC Application Client

Using virtualization within desktop grids is not a new field. We already investigated this topic in 2007 and published a technical report detailing the results in 2008 [11] and a research paper in 2010 [12]. In this research we (i) defined a criteria system for comparing different desktop virtualization solutions for desktop grids; (ii) evaluated the available tools (Bochs, QEMU, KQEMU, VMWare Player and VirtualBox); (iii) defined a generic architecture which allows building virtualized environments for task execution on desktop grid and volunteer resources; and (iv) did a reference implementation with focus on integration with BOINC and XtremWeb. We chose QEMU and KQEMU as the virtualization software for our implementation since at that time it was the solution best fitting to our criteria system. The second best solution was VirtualBox. Later we faced two issues which made us revise our initial approach.

(1) Our original proposition was an integrated solution where the virtualization based execution service is integrated with the given (desktop) grid middleware client. Others chose this approach as well [13]. The problem is that this would require extensive modifications in the client desktop grid software. In case of BOINC there are currently 6.7 million hosts running the client software [14] and only after an update would these hosts be able to run virtualization enabled tasks. This is not feasible and we think that a solution is required that is not tightly coupled with the client software and does not impose any modifications to the client. This can be achieved the easiest by including all virtualization related parts in a "traditional" application which would act as a wrapper.

(2) QEMU dropped support for KQEMU and is left without acceleration support on Windows. QEMU itself (without KQEMU) is too slow, as our evaluation has shown [11], to be considered as a viable alternative. Contrary, VirtualBox has progressed a lot in the recent years. It was improved in many ways and currently based on our criteria system [11] we consider it the best virtualization solution for desktop grids. Several other implementations aiming at utilizing virtualization for desktop grids have emerged and these also build upon VirtualBox. For example, CernVM [15] and VBoxWrapper [16] are available for BOINC. The common aspect in these solutions is that both are explicitly developed for BOINC and are not intended to become a generic framework.

Based on these two considerations we decided that instead of relying on QEMU in the future (without KQEMU support)

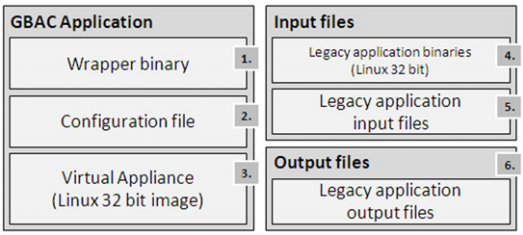

Fig. 2. GBAC: application, inputs and outputs.

we would switch to VirtualBox (see (1)); and instead of tight integration we have chosen a wrapper like approach (see (2)). We put the QEMU based implementation aside and chose VBoxWrapper as basis for a new implementation. The difference between our implementation and the basic VBoxWrapper one is that we intend to provide a generic framework and not a BOINC specific one. Currently our implementation supports BOINC, Condor and XtremWeb middleware beside standalone execution. Our approach differs in the following: (a) we are providing a generic framework that can be used with multiple middleware, although for demonstration purposes in this work we use BOINC; (b) we want to support multiple virtualized environments through multiple layered virtual appliances (see Section 5.3) and (c) we consider GBAC as one of the foundations for volunteer clouds rather than only a wrapper able to execute applications within a virtualized environment.

The Generic BOINC Application Client (GBAC) is a virtualization based wrapper. Contrary to its name it aims to be a generic framework providing virtualized environments for various distributed computing infrastructures (DCIs). GBAC is implemented using the DC-API Meta API and does not rely on any middleware specific functionalities, thus it is possible to use it on any DCIs that are supported by DC-API. In the following we refer to the BOINC version of GBAC for demonstrating its concepts and internals.

GBAC wrapper consists of the following components as shown in Fig. 2: First, the wrapper binary (see 1. in Fig. 2) itself is a BOINC enabled DC-API application that contains all BOINC related parts and handles communication with the BOINC client. Its task is to set up the client execution environment and manage the virtual machine on the client machine. Second, a supplied XML based configuration file (see 2. in Fig. 2) is used to set the different parameters of the virtual machine: (i) the operating system type (e.g., Linux 64bit); (ii) the size of the allocated memory for the virtual machine; (iii) whether the machine should have network access; (iv) which virtual appliance to use; and (v) whether to enable a shared directory between the host and the guest (the virtual machine). The third component is a virtual appliance (see 3. in Fig. 2) that contains the operating systems and libraries for the virtual machine. This image contains a 32bit Linux installation with some GBAC related components that will be detailed later in Section 5.2.

The wrapper sets up the client execution environment first by creating a shared directory for the virtual machine. It puts all input files in this directory. GBAC does not separate the binaries of the legacy application (typically a parameter sweep application as described in the introduction) and its input files (see 4. and 5. in Fig. 2). This means that all legacy application binaries and their input files are normal input files for GBAC and are not part of the GBAC BOINC application. As a next step the virtual machine is started using VirtualBox. GBAC does not contain VirtualBox; it is a prerequisite that every host has VirtualBox preinstalled before GBAC can be used. Next, the legacy application is executed in the virtual machine and the results are copied to the shared directory.

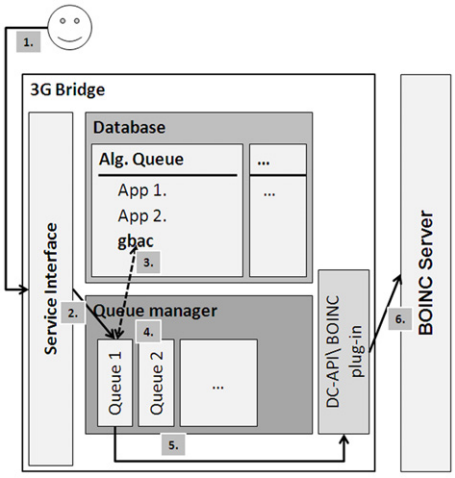

Fig. 3. GBAC: server side related parts.

Once the application finishes the virtual machine shuts down and finally GBAC copies the results from the shared directory to the work directory of the BOINC client and terminates.

In the following subsections we detail the functionality of GBAC by describing a job submission generated from a parameter sweep application that is not registered at the BOINC server and hence without GBAC BOINC would not be able to handle it. In this description of BOINC we assume that the PS application is initiated either in a service grid VO (gLite, ARC or UNICORE) that is extended with a BOINC DG system or by the WS-PGRADE portal [17] that can submit jobs directly to BOINC systems. In both cases 3G Bridge [18] plays a crucial role connecting the SG systems with BOINC and enabling the job submission from WS-PGRADE to BOINC.

5.1. Server side

The 3G Bridge provides a job handler service interface for accepting job submissions (see 1. in Fig. 3). It assigns jobs to different job queues and stores them in its job database. Different DCI plug-ins can be used to submit (forward) jobs to different DCIs. 3G Bridge also provides extended services like a web service interface to add and query jobs.

3G Bridge keeps a list of the supported algorithms (applications in case of desktop grids) for each configured DCI. When a job is received (see 2. in Fig. 3), 3G Bridge checks if the job fits to a registered algorithm (see 3. in Fig. 3). If the algorithm is not registered it means that the application is not deployed so the job cannot be executed. When GBAC is registered in the algorithm Queue of 3G Bridge the job for the "unknown" PS application is redirected to it with one constraint, namely the task should contain not only the input files (which is normal for desktop grid job submission) but the application binaries as well or else the execution will fail. It is possible to include all application related files in a special named bundle (zip or tar.gz). This is only for user convenience and it eases job submission. Once the job is internally redirected to the GBAC queue, 3G Bridge submits it to the desktop grid via its configured DC-API plug-in (see 5. and 6. in Fig. 3) that generates a GBAC work unit for the connected BOINC server (see 6. in Fig. 3). After this BOINC will register the new work unit for the deployed GBAC application and when a client with deployed VirtualBox asks for tasks BOINC will assign the work unit to the client. The current client side implementation of GBAC is based on VirtualBox and the BOINC client is able to detect and report to the BOINC server whether VirtualBox is installed on the client. The BOINC server will assign GBAC tasks only for those hosts that have VirtualBox preinstalled.

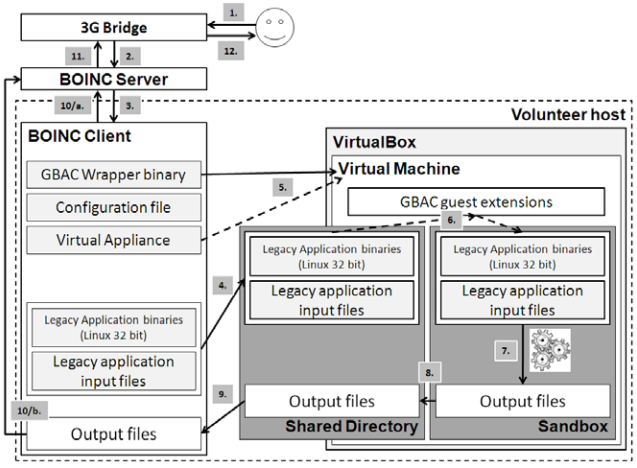

Fig. 4. GBAC: execution of a sample task.

5.2. Client side

First a legacy application with its inputs is submitted via the 3G Bridge to BOINC (see 1. and 2. in Fig. 4) and as written in Section 5.1 it is transformed into a GBAC application work unit containing the binaries and input files of the legacy application as inputs of the GBAC application. When a BOINC client connects, from a host where VirtualBox is installed, to the BOINC server and asks for tasks, it receives the work unit containing the GBAC application with its inputs. The BOINC client first downloads the GBAC binary, its configuration file and the virtual appliance along with the legacy application binaries and the input files (see 3. in Fig. 4).

After the download finishes the client starts the GBAC application. The first task of GBAC is to bootstrap the execution environment: it creates a directory that will be shared between the host and guest and all legacy application binaries and input files are copied here (see 4. in Fig. 4). Also a special file is put into this directory that contains additional parameters of the application (e.g., command line parameters and environment variables). After this the virtual machine is started (see 5. in Fig. 4) using the VirtualBox command line interface ("VBoxManage"). All parameters for the virtual machine are set in a configuration file for GBAC. This configuration file is currently part of the application and is common for every GBAC application work unit, but it can also be supplied individually for each task allowing more customization for the virtual machine. The configuration file also contains a reference to the virtual appliance to be used. Currently a single Linux appliance has been developed for GBAC but using more appliances is not prohibited by the GBAC concept. The appliance contains the GBAC guest extensions (see 6. in Fig. 4) and these are started right after the boot process finished. First, a component checks if there are any application bundles found in the shared directory. If yes, it extracts the contents to a separate sandbox within the virtual machine. If not, then all files are simply copied to the sandbox. After this the legacy application is started in the sandbox (see 7. in Fig. 4). Volunteer resources are highly volatile and any application running on these resources in the framework of a volunteer computing project should be prepared for interruption or shut down. This problem is usually solved by including an application specific (application-level) checkpointing mechanism in each application (see Section 4.1). The advantage of GBAC is clear here: application-level checkpointing is not needed since GBAC uses the system-level mechanism provided by the virtual machine monitor. GBAC can be suspended or stopped at any time regardless of what application it is executing and the suspended GBAC application can be resumed either on the current client or on a new client. While the virtual machine is running, GBAC continuously monitors its status through the virtual machine monitor.

After the execution of the legacy application has finished, all new and changed files are copied to the shared directory by the guest extensions (see 8. in Fig. 4). Finally the virtual machine is instructed to shut itself down. Next GBAC notices that the virtual machine has terminated and it will copy all changed files from the shared directory to the working directory and terminate itself (see 9. in Fig. 4). From here the BOINC client takes over, it will contact the BOINC server (see 10/a. in Fig. 4), upload the output files (see 10/b. in Fig. 4) and report the completion of the task. Next 3G Bridge fetches the results from BOINC (see 11. in Fig. 4) and will return it to the submitter (see 12. in Fig. 4).

BOINC employs reporting techniques which allow (i) the native applications to report their status (e.g., completion ratio) and (ii) measure the CPU time used for computation of the current task. These data are visualized in the BOINC client so that volunteers know what the status of the current task is and what to expect. Also for each task an upper limit for used resources (FLOPs, disk and memory) is set by the entity that created the task (in this case by 3G Bridge). In case of applications running in a virtualized environment (which acts basically like a sandbox) the reporting techniques do not work. The current implementation of GBAC does not allow reporting completion status (see (i)), instead the ratio is calculated from the ratio of used and estimated FLOPs. CPU time is not measured directly either (see (ii)) rather the total time used by the virtual machine is reported. We think this is better since the overhead caused by the virtual machine should also be included in the total (and thus the volunteer should be rewarded for that). GBAC also considers the overhead (CPU and memory) introduced by the virtualization. Currently when a task is submitted via 3G Bridge to GBAC the upper resource limit for the task is automatically increased by a predefined percentage. In case the submitted task takes more memory than is predefined, the OS (in virtual machine) can use its swap partition (if configured) or the application will simply abort with a "run out of memory" error.

5.3. Virtual appliance management

GBAC provides a standardized virtual environment for tasks of applications and services. Currently we provide a single base virtual appliance that contains a 32 bit Linux distribution (Debian 6.0) with its default components. For each task a new virtual machine instance is started with the appliance by GBAC and shut down after the task is finished. There are two concerns with this: (i) although tasks are executed inside a sandbox within the virtual machine, if a task is somehow able to break out and do modifications to the file system of the virtual machine (e.g., the kernel is deleted) that might render it unusable permanently; (ii) after each execution a cleanup procedure is required to remove the remnants of the previous task.

To overcome these problems we chose to compose the virtual appliance for GBAC of multiple overlay images (appliances) as shown in Fig. 5. First our base virtual appliance is immutable, meaning that no modifications can be done to it (see (i)). This is enforced not at the file system level, rather it is guaranteed by the hypervisor. It redirects all disk I/O to a separate overlay image which is specific to each running virtual machine instance. This also solves the problem of cleaning up the sandbox (see (ii)) after each execution, since the instance image can be simply thrown away. Second, we introduce optional application specific overlay

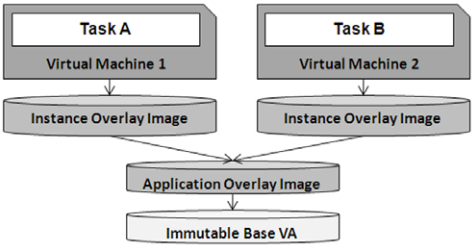

Fig. 5. A virtual appliance for GBAC is composed of a base image, an optional application overlay image and an instance image.

images which can contain all application specific dependencies that are supplementary for the base image. In this case the immutable image is a composite of the base and the application image and all file system modifications initiated in the virtual machine are redirected to the instance image.

6. Evaluation and related work

Since GBAC is a functional extension of the desktop grid middleware (e.g., BOINC) focusing on providing a virtualized environment, it does not make much sense to speak about performance evaluation of GBAC. Furthermore, the virtualization itself where performance can be an issue is realized by the VirtualBox tool. However, there are several functions where performance can be an important factor in this overall BOINC environment and some optimization can help a lot. For example the distribution of the virtual appliance among the client machines or startup of the virtual appliance on the client machine is crucial concerning the overall performance of the system. Reducing the size of the virtual appliance can easily solve these problems, or for distributing virtual appliances existing p2p data storage systems can be used, like Attic [19].

The BOINC VBoxWrapper is a virtualized environment providing wrapper for BOINC created by the developers of BOINC. It interfaces between the BOINC client and VirtualBox. It supports "single purpose" (traditional wrapped) and "multi purpose" (GBAC like) applications as well. It is a BOINC specific wrapper as it relies on middleware specific functionalities (e.g., file transfer between the host and the shared directory, application execution) and does not aim at being a generic one. Nevertheless our GBAC wrapper implementation is based on an early version of VBoxWrapper and we are convinced that it is one of the most matured solutions for virtualization of BOINC clients.

The CernVM solution is also based on the BOINC VBoxWrapper tool however, the main difference is in the virtual appliance and work distribution. Usually, in a BOINC environment the downloaded work unit (including the image as well) contains inputs and results are uploaded to the BOINC server when the job is finished. The CernVM solution follows a new approach where the virtual appliance contains a complete job scheduler that performs job fetching from an external server by itself. So, this solution does not utilize BOINC work unit scheduling and the application validation framework rather uses the BOINC server only for distributing the CernVM images.

Ferreira et al. [13] designed a common set of VM manipulation functions to hide the hypervisor specific details. Five methods have been defined by the API covering information query, VM startup, file copy, checkpoint and finally command execution inside the virtual machine. In this experience they have implemented the API for local execution, VMWare and VirtualBox. Finally, they have integrated this library as an application into the BOINC wrapper and performed tests to measure the overhead. Its most powerful feature is the easy switching between different virtualization environments, however, this work is somehow a duplication of the virtualization API and its libraries provided by libvirt.org.

7. Conclusions

The GBAC concept enables the intensive and wide-spread use of volunteer BOINC Desktop Grid systems in the future. Currently, without virtualization the use of BOINC is very limited. Typically one BOINC project executes one grand-challenge application. There are only very few umbrella BOINC projects (IBM Community Grid [20], SZTAKI Desktop Grid [21], IberCivis [22], EDGeS@home [23]) that are intended to support more than one application and even these projects support fewer than 10 applications. There are 60–80 active BOINC projects in the world [14] so the use of BOINC concerning the number of supported applications is really limited.

The GBAC approach enables the execution of any parameter sweep application on a BOINC project that registers the GBAC application. In this way volunteer BOINC systems can be used for a much wider set of applications than before without any porting effort. This is particularly important in two scenarios.

The first scenario comes from the EDGI project where Service Grid Virtual Organizations can be extended with volunteer and local DG systems. If these Desktop Grid systems register the GBAC application then any parameter sweep application running on Service Grids can be automatically transferred to and executed in the connected Desktop Grids. No application porting effort is required. This opens a new horizon for extending Service Grids with a large set of volunteer and local Desktop Grid resources. Since EDGI strongly collaborates with EMI [24] and EGI [25], the typical service grid VOs in Europe will be extended with volunteer and university DG systems in the future. The introduction of the GBAC concept in those Desktop Grids will significantly increase the popularity of this option of extending the limited size of service grid Virtual Organizations with on-demand volunteer cloud system resources.

In the second scenario universities would like to set up volunteer DG systems for their researchers and students as it is under way in Hungary within the framework of the Web2Grid project. In this program the e-science research infrastructure of many of the Hungarian universities will be extended with a volunteer or local BOINC DG system depending on the requirements and choice of the universities. In order to ensure the flexibility of these university DG systems the GBAC approach will be applied. At the discussion forum of the International Desktop Grid Federation [26] case studies and opinions are published discussing the possible ways of creating such university DG systems not only from Hungary but also from the UK.

The current GBAC solution supports only Linux PS applications since the GBAC virtual appliance is based on Linux. It means that Linux-based PS applications can run on Linux and MS Windows volunteer clients without porting the Linux applications to Windows. Naturally, GBAC can also be equipped with MS Windows virtual appliance in the future. In that case Windows applications will be able to run on Linux and MS Windows clients without porting the Windows applications to Linux. However, in this case some licensing issues might be raised.

The work described in this paper is discussed with the BOINC team in order to avoid redundant and parallel research and development. Once the GBAC solution is matured enough to run in production both in the EDGI and Web2Grid infrastructure our BOINC related VBoxWrapper modifications will be returned to the BOINC team in forms of patches to be included in the stock version of BOINC.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement no 261556 (EDGI). The research has been partially supported by the Hungarian National Office for Research and Technology (NKTH) under grant No TECH_08-A2/2-2008-0097 (WEB2GRID).

References

[1] The EDGI EU FP7 project. http://edgi-project.eu.
[2] David P. Anderson, BOINC: a system for public-resource computing and storage, in: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID'04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 4–10. http://dx.doi.org/10.1109/GRID.2004.14.
[3] G. Fedak, C. Germain, V. Neri, F. Cappello, XtremWeb: a generic global computing system, cluster computing and the grid, in: 2001, Proceedings, First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, pp. 582–587. http://dx.doi.org/10.1109/CCGRID.2001.923246.
[4] P. Kacsuk, J. Kovacs, Z. Farkas, A. Marosi, Z. Balaton, Towards a powerful European DCI based on desktop grids, J. Grid Comput. 9 (2011) 219–239.
[5] Attila Csaba Marosi, Gabor Gombas, Zoltan Balaton, Peter Kacsuk, Enabling Java applications for BOINC with DC–API, in: Distributed and Parallel Systems, Proceedings of the 7th International Conference on Distributed and Parallel Systems, 2009, pp. 3–12.
[6] Attila Csaba Marosi, Zoltan Balaton, Peter Kacsuk, GenWrapper: a generic wrapper for running legacy applications on desktop grids, in: 3rd Workshop on Desktop Grids and Volunteer Computing Systems, PCGrid 2009, Rome, Italy, May, 2009.
[7] Etienne Urbah, Peter Kacsuk, Zoltan Farkas, et al., EDGeS: bridging EGEE to BOINC and XtremWeb, J. Grid Comput. 7 (3) (2009) 335–354. http://dx.doi.org/10.1007/s10723-009-9137-0. (Print) 1572-9814 (Online). ISSN: 1570-7873.
[8] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, Seti@home: an experiment in public-resource computing, Commun. ACM 45 (2002) 56–61.
[9] Jon Watson, VirtualBox: bits and bytes masquerading as machines, Linux J. 2008 (166) (2008) 1.
[10] E.M. Heien, Y. Takata, K. Hagihara, A. Kornafeld, PyMW—a python module for desktop grid and volunteer computing, in: IEEE International Symposium on Parallel & Distributed Processing, 2009, IPDPS 2009, 23–29 May 2009, pp. 1–7. http://dx.doi.org/10.1109/IPDPS.2009.5161132.
[11] A. Csaba Marosi, P. Kacsuk, G. Fedak, O. Lodygensky, Using virtual machines in desktop grid clients for application sandboxing, Technical Report TR-0140, Institute on Architectural Issues: Scalability, Dependability, Adaptability, CoreGRID—Network of Excellence, August 2008.
[12] A.C. Marosi, P. Kacsuk, G. Fedak, O. Lodygensky, Sandboxing for desktop grids using virtualization, in: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2010, vol., No., 17–19 February 2010, pp. 559–566.
[13] D. Ferreira, F. Araujo, P. Domingues, libboincexec: a generic virtualization approach for the BOINC middleware, in: IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, IPDPSW, 2011, vol., No., 16–20 May 2011, pp. 1903–1908. http://dx.doi.org/10.1109/IPDPS.2011.349.
[14] http://boincstats.com.
[15] Carlos Aguado Sanchez, et al., Volunteer clouds and citizen cyberscience for LHC physics, J. Phys. Conf. Ser. 331 (2011) 062022. http://dx.doi.org/10.1088/1742-6596/331/6/062022.
[16] http://boinc.berkeley.edu/trac/wiki/VboxApps.
[17] P. Kacsuk, P-GRADE portal family for Grid infrastructures, Concurr. Comput.: Pract. Exper. J. 23 (3) (2011) 235–245.
[18] Z. Farkas, P. Kacsuk, Z. Balaton, G. Gombás, Interoperability of BOINC and EGEE, Future Gener. Comput. Syst. 26 (8) (2010) 1092–1103. ISSN: 0167-739X http://www.sciencedirect.com/science/article/pii/S0167739X10000890 http://dx.doi.org/10.1016/j.future.2010.05.009.
[19] Ian Kelley, Ian Taylor, Bridging the data management gap between service and desktop grids, in: Peter Kacsuk, Robert Lovas, Zsolt Nemeth (Eds.), Distributed and Parallel Systems in Focus: Desktop Grid Computing, Springer, 2008.
[20] http://www.worldcommunitygrid.org/.
[21] http://szdg.lpds.sztaki.hu/szdg/.
[22] http://www.ibercivis.es/.
[23] http://home.edges-grid.eu/home/.
[24] http://www.eu-emi.eu/.
[25] http://www.egi.eu/.
[26] http://desktopgridfederation.org/.

Attila Marosi is a member of the Laboratory of Parallel and Distributed Systems at the Computer and Automation Research Institute of the Hungarian Academy of Sciences since 2001. He received his M.Sc. from the Budapest University of Technology and Economics in 2006. He started Ph.D. studies at the Budapest University of Technology and Economics in 2009. His research interests include desktop grid and cloud computing. He participated in many national (HAGRID, Web2Grid) and international (EGEE, CoreGrid, EDGeS, EDGI) research and development projects. He is the coauthor of more than 20 scientific papers in journals and conference papers related to grid computing.

József Kovács was born in 1975 in Budapest, Hungary. He is a Senior Research Scientist at the Laboratory of Parallel and Distributed Systems (LPDS) of the Computer and Automation Research Institute of the Hungarian Academy of Sciences. He obtained his B.Sc. (1997), M.Sc. (2001) and Ph.D. (2008) in the field of Informatics. From 1998, he was continuously involved in national, international and European (Esprit, FP5, FP6, FP7) R&D projects. From 2002 his research focused on parallel checkpointing techniques in grids and later on the field of Desktop Grid computing. From 2006 he has been the leader of the Desktop Grid team of LPDS. He is the author and co-author of more than 40 scientific papers and also a regular reviewer of several journals.

Peter Kacsuk is the Director of the Laboratory of the Parallel and Distributed Systems in the Computer and Automation Research Institute of the Hungarian Academy of Sciences. He received his M.Sc. and university doctorate degrees from the Technical University of Budapest in 1976 and 1984, respectively. He received the Kandidat degree (equivalent to Ph.D.) from the Hungarian Academy in 1989. He habilitated at the University of Vienna in 1997. He received his professor title from the Hungarian President in 1999 and the Doctor of Academy degree (DSc) from the Hungarian Academy of Sciences in 2001. He served as Full Professor at the University of Miskolc and at the Eötvös Lóránd University of Science Budapest. He has been a part-time Full Professor at the Cavendish School of Computer Science of the University of Westminster. He has published two books, two lecture notes and more than 200 scientific papers on parallel computer architectures, parallel software engineering and Grid computing. He is co-editor-in-chief of the Journal of Grid Computing published by Springer.