Securing the Grid using Virtualization
The ViSaG Model

Pierre Kuonen, Valentin Clément, Frédéric Bapst
University of Applied Sciences Western Switzerland, HES-SO//Fribourg
Fribourg, Switzerland
Emails: {pierre.kuonen, frederic.bapst}@hefr.ch, clementval@gmail.com

***Abstract*—Security in large distributed computing infrastructures, peer-to-peer, or clouds, remains an important issue and probably a strong obstacle for a lot of potential users of these types of computing infrastructures. In this paper, we propose an architecture for large scale distributed infrastructures guaranteeing confidentiality and integrity of both the computation and the host computer. Our approach is based on the use of virtualization and we introduce the notion of *confidence link* to safely execute programs. We implemented and tested this approach using the POP-C++ tool, which is a comprehensive object-oriented system to develop applications in large decentralized distributed computing infrastructures.**

Keywords-virtualization; security in large distributed system; grid middleware.

I. INTRODUCTION

Today, more and more applications require having punctually access to significant computing power. The purchasing of High Performance Computing (HPC) hardware is really profitable only in case of frequent usage. There are several alternatives to purchasing HPC hardware. The two most popular are Clouds [11] and Grids [12]. Even if these two approaches are not totally identical, they share at least one difficulty, which is the fact that the user has to trust the resources provider. In the case of Grid infrastructures, this problem is complemented by the fact that the resource provider also has to trust the user to be sure that running user's tasks will not harm his own resources. This paper will focus on how, by using virtualization, we can guarantee confidentiality and integrity of computing and resource for the user and the resources provider in decentralized distributed computing environments, such as Grid systems.

The main questions we intend to answer are:

- How to ensure the integrity of the user's data and the user's calculations?
- How to ensure the integrity of the host machine?
- How to ensure the confidentiality of the communications?
- How to safely use machines belonging to different private networks in the presence of firewalls?
- How to ensure that different users sharing the same computing resource cannot interfere between each other?

Last, but not least, we want to provide these features while minimizing the loss of performances.

We propose an abstract vision of a secure decentralized distributed computing environment. This vision is based on the notion of *confidence links*. It has been implemented in the ViSaG project (ViSaG stands for: Virtual Safe Grid), which is presented in this paper.

The rest of this paper is organized as follows: Section II details the main security issues we want to address with the ViSaG project. Section III presents the ViSaG model and Section IV presents the POP-C++ model, which has been used to implement the ViSaG model. Section V details the implementation of the ViSaG model using the POP-C++ middleware and Section VI presents the results we have obtained. Finally, Section VII concludes this paper.

II. CURRENT SECURITY ISSUES IN GRID COMPUTING

As mentioned in the introduction, there are several security issues in current Grid middleware systems that must be addressed. These issues are detailed below.

A. How to ensure the integrity of the user's data and the user's calculations

When a user submits a computation on a remote machine he must be sure that the owner of the remote resource cannot interfere with his computation, or, at least, if there is interference the user must be aware of it.

B. How to ensure the integrity of the host machine

Consider a user of the Grid willing to provide his computing resources to the infrastructure. As this user does not have a strong control on who will execute a job on his resources, he wants that the middleware guarantees him that the executed jobs cannot access unauthorized resources, cannot harm his resources, and cannot use more resources than he agreed to allocate to them.

C. How to ensure the confidentiality of the communications

We want to secure communications between nodes. First, we want to prevent communications from being seen by any other person/system and also we do not want that anyone could intercept and modify the transmitted data.

D. How to safely use machines belonging to different private networks (presence of firewalls)

One of the most difficult security problems when deploying decentralized Grid middleware is due to the presence of private networks protected by firewalls. Indeed, most of the time available resources in an institution, which could be part of a Grid, are resources located on private networks protected by firewalls. The question is: how to make these resources available without creating dangerous security holes in the firewalls.

E. How to ensure that different users using the same computing resource cannot interfere between each other

We also have to ensure that a user of a remote resource cannot harm processes of other users on the same remote resource.

Usage of virtual machines in conjunction with Grids to address security issues has been already proposed in several papers, one of the first being [3]. Santhanam et al. [5] propose four scenarios to deploy virtual machines on Grid. None of these four scenarios exactly corresponds to our approach, even if the fourth is the closest. Smith et al. [6] propose a Grid environment enabling users to safely install and use custom software on demand using an image creation station to create user-specific virtual machines. Keahey et al. [4] focus on creating, configuring and managing execution environments. The authors show how dynamic virtual environments can be modeled as Grid services, thus allowing a client to create, configure and manage remote execution environments. In all these papers, the problem of deploying virtual machines in a Grid is addressed in a general way, although Santhanam et al. [5] have used Condor to test their scenarios. Our approach is different because the model we propose is closely related to an execution scheme based on the paradigm of distributed object-oriented programming. The proposed solution is specifically designed to solve the problems associated with this model such as the creation and destruction of the remote object (process) and passing remote objects as parameters of remote methods.

III. THE VISAG MODEL

Unlike most existing Grid middleware, the approach proposed in the ViSaG project is based on the fundamental assumption that a Grid infrastructure is a fully decentralized system, which, in a sense, is close the peer-to-peer (P2P) network concept. At the hardware level, a computing Grid is composed of an unknown, but large, number of computer owning computing resources. None of the computers in the Grid has a global view of the overall infrastructure. Each computer only knows a very limited number of neighbors to which it is directly connected by two-way confidence links. A confidence link is a bidirectional channel that allows two computers located at both ends of the link to communicate safely at any time. How the confidence links are established is not part of the ViSaG model, but is a hypothesis which defines our vision of a computing Grid. However, we can give as an example of a confidence link, an SSH channel between two computers whose system managers, or users, have manually exchanged their public keys. The set of all computers together with all the confidence links form a connected graph, we call it a *trusted network*, whose nodes are computers and edges are the confidence links. In the remainder of this document, when no confusion is possible, we often use the terms nodes and links, respectively, to designate computers and confidence links. Although, usually, computers are volatile resources, we will not address this aspect in this paper, where we made the assumption that, during the execution of a given program, computers participating to this execution do not fail. Finally, we assume that the confidence links are reliable.

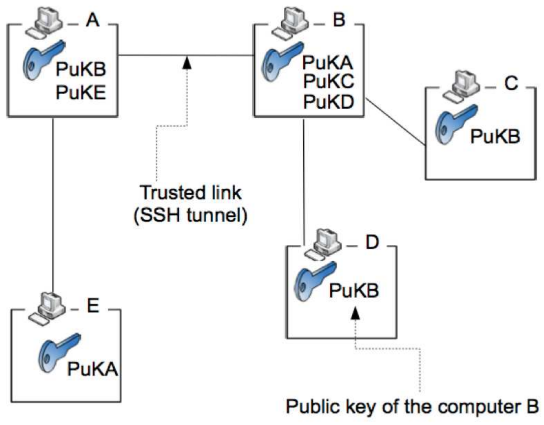

Figure 1. A trusted network using SSH tunnels.

Figure 1 illustrates a computing Grid, as defined above, where confidence links have been realized using SSH (Secure Shell [10]) tunnels.

In the ViSaG execution model, computing resources are requested on the fly during execution of the application. Obtainment of requested resources is achieved through the usage of a *resource discovery algorithm* which runs on every node and only uses confidence links. Usually, this algorithm is a variant of the flooding algorithms Details of this algorithm are not part of the model, but is an implementation issue. When a node, we call it the *originator*, needs a new computing resource, it issues a request, which will be handled by the resource discovery algorithm.

When the requested computing resource, provided by a node we will call the *target*, has been found, the originator of the request must contact the target to launch the computation and possibly communicate with it during the computation. For the originator, one possibility would be to communicate with the target by following confidence links. This option is, obviously, very inefficient because it exaggeratedly loads all intermediary nodes which have to route all messages. This is especially true when, during computation, nodes must exchange large data as is often the case in HPC applications. A better solution would be for the originator, to contact the target directly. Unfortunately, it is likely that the originator does not have a direct link (a confidence link) with the target and, in addition, the target does not necessarily desire to create a direct confidence link with the originator. Nevertheless, as the request reached him following confidence links, the target could accept to launch a virtual machine to provide the necessary computing resources for the originator. The virtual machine will act as a sand box for the execution of the remote process. If the virtual machine is not permeable, this will guarantee that the executing node (the target) cannot be damaged by the execution of the remote process and that the computation made by the process cannot be biased by the node which hosts the computation (the target). To summarize, we can make the following statement:

The security of this execution model is only limited by the security offered by the virtual machine and the security offered by the confidence links.

This is the very basic idea of the ViSaG model. The implementation of such a model raises numerous problems that we are going to address in the next sections.

IV. THE POP-C++ EXECUTION MODEL

A Grid computing infrastructure not only consists in hardware but also requires the presence of a middleware which provides services and tools to develop and to run applications on the Grid. Therefore, before presenting how the ViSaG model has been implemented we need to know which Grid middleware our implementation is based on. The ViSaG model, as presented in the previous section, has been implemented in the POP-C++ Grid middleware [1]. In order to achieve this task we had to adapt to the execution model of POP-C++, which is briefly presented below. For more information of the POP-C++ tool please visit the POP-C++ web site: http://gridgroup.hefr.ch/popc.

The POP-C++ tool implements the *POP programming model* first introduced by Dr. Tuan Anh Nguyen in his PhD thesis [2]. The POP programming model is based on the very simple idea that objects are suitable structures to distribute data and executable codes over heterogeneous distributed hardware and to make them interact between each other.

The object oriented paradigm has unified the concept of module and type to create the new concept of class. The next step introduced by the POP model is to unify the concept of class with the concept of task (or process). This is realized by adding to traditional “sequential” classes a new type of classes: *the parallel class*. By instantiating parallel classes we are able to create a new category of objects we call *parallel objects*. Parallel objects are objects that can be remotely executed. They coexist and cooperate with traditional sequential objects during the application execution.

POP-C++ is a comprehensive object-oriented framework implementing the POP model as a minimal extension of the C++ programming language. It consists of a programming suite (language, compiler) and a run-time providing the necessary services to run POP-C++ applications.

In the POP-C++ execution model, when a new parallel object is created, the node which required the creation of the parallel object contacts the POP middleware running locally to ask for a new computing resource for this parallel object. To find this new computing resource, the POP middleware launches the resource discovery service available in the POP-C++ middleware. This service will contact all the neighbors of the node thanks to its confidence links, to ask for computing resources. Then the request is propagated through the network by following confidence links. When the request reaches a node which is able to provide the requested computing resource, it answers the originator of the request by following back the confidence links. The originator of the request chooses, between all the positive answers it received, the resource it wants to use to create the parallel object and remotely launch the execution of the parallel object inside a virtual machine. In order to be able to use the procedure presented above with the POP-C++ runtime, we had to design a dedicated architecture for the nodes of the Grid. This architecture is presented in the next section.

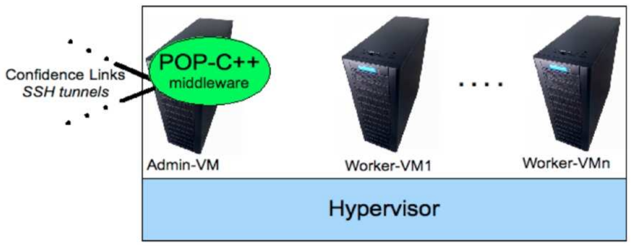

Figure 2. Architecture of a node in the ViSaG model implementation

V. IMPLEMENTATION

A. Node architecture

In the presented implementation, a node is a computer running a virtualization platform, or hypervisor. On this platform, two or more virtual machines are deployed. The first virtual machine, called the *administrator* virtual machine (or in short: *Admin-VM*) is used to run the POP-C++ runtime. The other virtual machines are the *worker* virtual machines (or in short: *Worker-VM*). They are used by the Admin-VM to run parallel objects. The Admin-VM is connected to its direct neighbors in the Grid by the confidence links. The latter are implemented using SSH tunnels. Figure 2 illustrates this architecture.

One of the first questions we have to answer is how many Worker-VMs do we launch on a specific node. In other words, do we launch all parallel objects in the same Worker-VM, or do we launch one Worker-VM for each parallel object allocated to this node? In order to guarantee isolation between applications (see sub-section II.E) we decided to allocate Worker-VM on an application basis: for a given node, parallel objects belonging to the same application (the same POP-C++ program running instance) are executed in the same Worker-VM. This choice implies that we are able to identify applications. For this purpose, we have to generate a unique application identifier, called *AppUID*, for each POP-C++ program launched in the Grid. As we are in a fully decentralized environment, to guarantee unicity of the identifier we have based it on the IP address of the node where the program is launched, the Unix process ID as well as on the clock. This AppUID is added to all requests to allow identifying parallel objects belonging to the same POP-C++ program.

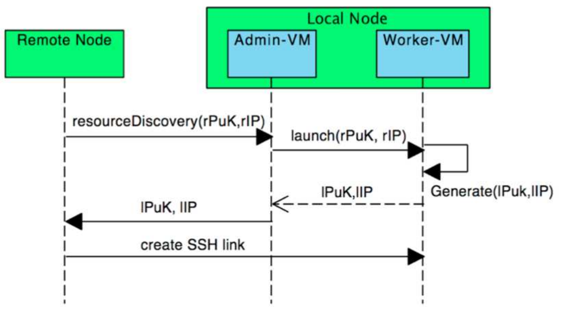

Figure 3. Creation of a parallel object.

When the Admin-VM launches a Worker-VM to provide a computing resource for the execution of a parallel object, it must ensure that the node that originates the request is able to safely contact this Worker-VM, i.e., is able to establish an SSH tunnel. This is realized through a key exchange process which is detailed below.

B. Key exchange process

There are two main situations where the POP-C++ middleware needs to exchange keys between virtual machines. The first one is, as mentioned above, when a new Worker-VM is launched, and the second is when the reference of a parallel object is sent to another parallel object. Indeed, as POP-C++ is based on the C++ programming language, it is possible to pass the reference of a parallel object as parameter of a method of another parallel object. As a consequence, these two parallel objects, possibly running on different nodes, must be able to communicate.

Let us first consider the situation where a new Worker-VM is launched by the Admin-VM. This operation is the consequence of a resource discovery request sent by a node that asked for the creation of a new parallel object. This request contains, among other information, the public key (rPuK) and the IP address (rIP) of the node that sent the request. The Admin-VM launches the Worker-VM and passes it the rPuK and the rIP address. The newly launched Worker-VM generates a new pair of private/public keys and sends its lPuK and lIP to the originator of the request along the confidence links. At this stage, both the originator of the request and the newly launched Worker-VM, have both PuKs and therefore are able to establish an SSH tunnel between them. This keys exchange process is illustrated on Figure 3.

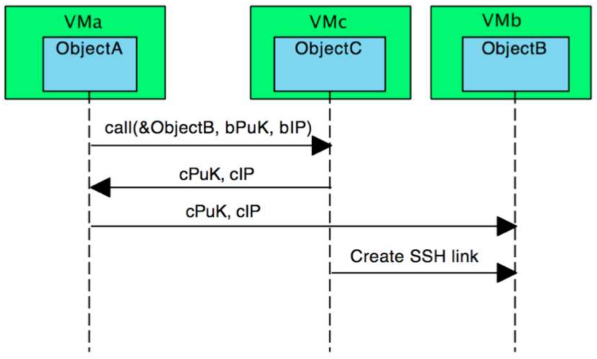

Figure 4. Passing a parallel object reference.

The second situation we have to consider is when a parallel object running on the virtual machine VMa sends the reference of a parallel object running on a virtual machine VMb to a third parallel object running on a virtual machine VMc. This situation is illustrated on Figure 4. In such a situation, the POP-C++ middleware must ensure that VMb and VMc can establish an SSH tunnel.

When this situation occurs, we necessarily have an object running on VMa calling a method of an object running on VMc. Thus, the first thing to do is to add the PuK and the IP address of VMb in the message sent by VMa to VMc. This does not increase the number of messages but just slightly increases the size of the message sent to execute a remote method call. Next, along the confidence links, VMc sends its PuK to VMb. Now VMb and VMa can establish an SSH tunnel.

We claim that the proposed infrastructure solves four of the issues mentioned in Section II, namely, issues A, B, C and E.

The integrity of the user’s data and the user’s calculation (issue A) as well as the integrity of the host machine (issue B) are guaranteed by the isolation the virtual machine provides between the host machines and the remotely executed process. The confidentiality of the communications (issue C) is guaranteed by the SSH tunnels. Finally, as each application is executed in a different virtual machine, we guarantee that different users using the same computing resources cannot interfere between each other (issue E).

The last issue to solve (issue D) is to be able to safely use machines belonging to different private networks in presence of firewalls. Indeed the nodes belonging to a same Grid are not necessarily in the same administrative domain and can be

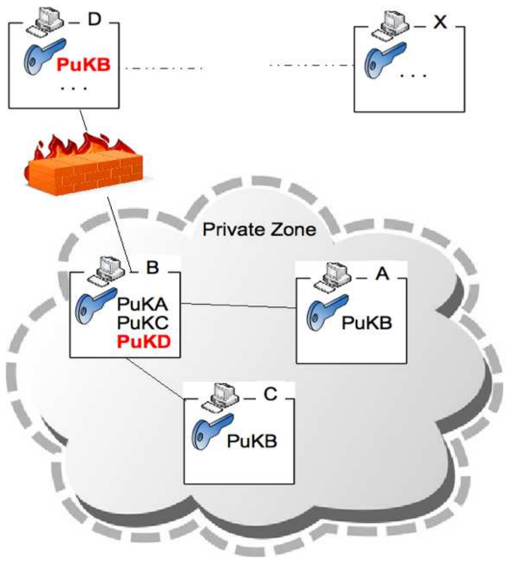

Figure 5. Example of a Grid including a private network protected by a firewall

separated by firewalls managed by different authorities. Our goal is to enable these nodes to belong to the same Grid and to be able to safely run parallel objects without opening security holes in firewalls.

Figure 5 shows a situation where three nodes (A, B and C) are in a private network separated from the rest of the Grid by a firewall. If at least one node of the Grid belonging to the private network (node B on Figure 5) creates at least one confidence link with one node located on the other side of the firewall (node D on Figure 5); then, it is possible to make this private network part of a Grid located outside of the private network. To achieve this goal we have to follow the following procedure. Suppose that a node X, located somewhere in the Grid but outside the private network, launches a request for resources. By following confidence links, this query can reach nodes located in the private network (thanks to the confidence link B-D). If a node inside the private network, let's say node A, agrees to carry out this execution, it must establish a communication with the node X. To achieve this, the Admin-VM of node A will launch a virtual machine (a Worker-VM) and will configure it with the public key of X contained in the request launched by node X. The Worker-VM creates a pair of public/private keys and transmits its public key to its Admin-VM. The latter transmits, by following the confidence links, this public key to the Worker-VM of the node X. Then, node X can establish an SSH tunnel with the Worker-VM started on node A. Now, node X and node A which are not in the same private network can safely directly communicate.

To be able to realize this communication, the firewall must be configured in the following way:

- It must allow the permanent SSH tunnel between nodes B and D.
- It must allow temporary establishment of an SSH tunnel between any node outside the local network and any Worker-VM launched by any Admin-VM inside the local network.

We claim that this configuration of a firewall is perfectly acceptable and does not create security holes if the administrator of the private network follows the following recommendations. First, the node D, which in a sense acts as a bridge toward outside the private network, must be under the control of the administrator of the private network. The Admin-VM running on this node must only run the minimal services required by the POP-C++ middleware. In our case, the SSH services with node B and the few fixed neighbors it will manually establish a confidence link with. Second, when installing the POP-C++ middleware in the private network, the administrator must take the following precaution. As the POP-C++ middleware creates and launches virtual machines, a good policy is to reserve a set of well-defined IP addresses only for this purpose and then to open, in the firewall, the SSH service only for this set of IP addresses. This will guarantee that the nodes external to the private network can only access, through SSH, Worker-VMs handled by the POP-C++ middleware. Of course, we make the assumption that the POP-C++ middleware itself is not malicious.

The middleware must ensure that when the execution of a POP-C++ program terminates, all Worker-VMs allocated to this program are deleted (or reset), to ensure that all links established during the program execution are destroyed.

VI. TESTS

To demonstrate the feasibility of the ViSaG model, we have developed a prototype integrated with the POP-C++ middleware. This prototype uses the VMware ESXi hypervisor [8] to manage virtual machines.

The virtual machine management layer can start, stop, revert, and clone virtual machines. It also allows to exchange SSH PKI and to get the IP address of a virtual machine. All these operations are performed thanks to the *libvirt* library [9] and the proprietary *VMWare VIX API*. As much as possible, we used libvirt to be compatible with different hypervisor virtualization platforms. Unfortunately, not all desired features were available, so we had to partially rely on the proprietary *VIX API* for a few key features such as cloning virtual machines and information gathering.

The SSH tunneling management is independent of any API because it uses the installed version of SSH to initiate and manage SSH tunnel. In our infrastructure, the installed version was OpenSSH running on Ubuntu 10.04 operating system.

To test our model and our prototype, we have deployed a Grid on two different sites. The first site was the “Ecole d’ingénieurs et d’architectes” in the city of Fribourg in Switzerland and the second was “Haute école du paysage, d'ingénierie et d'architecture” in the city of Geneva in Switzerland. These two sites were connected only by Internet and therefore, the security was a key point. More important, the two sites have totally different administrative network management, as required to make the test significant.

We have been able to run several distributed applications written with POP-C++ between the two sites in a transparent way for the users. The performance loss was acceptable; the main slowdown is due to the startup of the virtual machines.

VII. CONCLUSION AND PERSPECTIVES

This paper addressed the security issues in the context of a fully decentralized Grid infrastructure. Grid consumers, system managers, local users of a shared resource, network administrators, etc., are different actors involved in distributed computing, and as such they need security guarantees to accept taking part in a Grid infrastructure. Our solution takes advantage of virtualization as an isolation means, and on public key cryptography.

The existing POP-C++ Grid middleware is taken as the illustration of the decentralized Grid paradigm.

POP-C++ offers "parallel objects" as a programming model that essentially hides the complexity of the Grid aspects (local *vs* remote access, heterogeneous machines, resource discovery, etc.). On top of this architecture, and with no further constraint on the developer, our new implementation adds the wrapping of the parallel objects within virtual machines, as well as secure communications via SSH tunneling. Combining those two features brings a convincing answer to the security issues in decentralized Grids. Two levels of activities in the Grid are distinguished:

- Setup: to join a grid returns to configure and start a dedicated virtual machine (VM-Admin), which manages the POP-C++ services. The setup phase establishes connections to other nodes of the Grid; those confidence links ensure the connectivity of the Grid. The Admin-VM never executes user code itself, but has control over a pool of virtual machines for the user jobs. The setup is considered as a local event (it does not need to stop the Grid), and it typically involves a manual intervention of a user responsible of the Grid installation.

- Grid computing: when a POP-C++ user program is launched, the Admin-VMs communicate to distribute the jobs on the available resources; when it accepts a job, an Admin-VM wraps that job in a virtual machine (Worker-VM) that will be devoted to that running instance of the Grid program. The necessary encrypted connections with other Worker-VMs are automatically established, as our system takes care of conveying the needed public keys from node to node.

Thus launching a program on the Grid causes the start of several virtual machines that will be dedicated to this computation, with the appropriate communication topology. When the distributed program terminates, no trace of its execution remain (the involved virtual machines are reset before being recycled).

Our prototype has been implemented with ESXi virtual machines, but the code relies on libvirt, so that porting to another virtualization technology is greatly simplified.

The value of virtualization as a companion of Grid technology has been shown for many years. In a centralized Grid architecture, using virtual machines instead of physical systems can for instance greatly simplify Grid reconfiguration or load balancing. In the decentralized approach that we advocate, virtual machines are used as an isolation wrapper for pieces of distributed computing, a means to guarantee an appropriate security level.

In the course of our work, we identified several issues that need to be further investigated:

- It would be interesting to bring the current version based on ESXi on another virtualization software (hypervisor). The ideal candidate should provide the same level of isolation, but lightweight VMs management operations (start, stop, resume, revert, clone, etc.).

- A potential issue is about ensuring that the different VMs can benefit from system updates.

- Concerning the VMs installation, it would be worth to define precisely what capabilities have to be included in the OS equipment. In fact this leds to a concept of a "harmless Worker-VM", i.e., a virtual machines that somehow are restricted to compute and communicate with other harmless Worker-VMs only, and that are unable to cause any damage in the hosting environment (in particular no other network traffic).

- In our system, one hypothesis about security is that the POP-C++ installation is safe; we should study how this can be guaranteed and verified by the different Grid nodes.

ACKNOWLEDGMENT

The ViSaG project has been funded by the University of Applied Sciences Western Switzerland (HES-SO), project No.24247.

REFERENCES

[1] T. A. Nguyen and P. Kuonen, “Programming the Grid with POP-C++”, in Future Generation Computer Systems (FGCS), N.H. Elsevier, vol. 23, iss. 1, Jan. 2007, pp. 23-30.

[2] T. A. Nguyen, An object-oriented model for adaptive high-performance computing on the computational Grid. PhD Thesis no 3079, EPFL, Switzerland, 2004.

[3] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, “A Case for Grid Computing on Virtual Machines”, in International Conference on Distributed Computing Systems, May 2003, pp. 550-555.

[4] K. Keahey, K. Doering, and I. Foster, “From Sandbox to Playground: Dynamic Virtual Environments in the Grid”, in Proceedings of the 5th IEEE/ACM international Workshop on Grid Computing, Nov. 2004, pp. 34-42.

[5] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny, “Deploying Virtual Machines as Sandboxes for the Grid”, in Conference on Real, Large Distributed Systems - Volume 2, Dec. 2005, San Francisco, CA, pp. 7-12.

[6] M. Smith, M. Schmidt, N. Fallenbeck, T. Dörnemann, C. Schridde and B. Freisleben, “Secure on-demand grid computing” in Future Generation Computer Systems, Volume 25 Issue 3, March, 2009, Pages 315-325.

[7] M. Smith et al., “Secure on-demand grid computing”, in Future Generation Computer Systems archive, vol. 25, no. 3 March 2009, pp. 315-325.

[8] http://www.vmware.com [retrieved: March, 2014].

[9] http://libvirt.org/ [retrieved: March, 2014].

[10] http://en.wikipedia.org/wiki/Secure_Shell [retrieved: March, 2014].

[11] Michael Miller, “Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online”, Que Publishing Company 2008, ISBN:0789738031

[12] Ian Foster, Carl Kesselman, “The Grid 2: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA 2003, ISBN:1558609334