Future Generation Computer Systems 28 (2012) 136–146

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

Architecture for the next generation system management tools☆

Jérôme Gallard [a,*], Adrien Lèbre [c], Christine Morin [a], Thomas Naughton [b], Stephen L. Scott [b], Geoffroy Vallée [b]

[a] INRIA Rennes – Bretagne Atlantique, France
[b] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA
[c] Ecole des Mines de Nantes, France

ARTICLE INFO

Article history:
Received 20 January 2011
Received in revised form
10 June 2011
Accepted 29 June 2011
Available online 8 July 2011

Keywords:
HPC system resource management
Flexibility
Virtualization
Emulation
Distributed systems
Virtual platform (VP)
Virtual system environment (VSE)

ABSTRACT

To get more results or greater accuracy, computational scientists execute their applications on distributed computing platforms such as clusters, grids, and clouds. These platforms are different in terms of hardware and software resources as well as locality: some span across multiple sites and multiple administrative domains, whereas others are limited to a single site/domain. As a consequence, in order to scale their applications up, the scientists have to manage technical details for each target platform. From our point of view, this complexity should be hidden from the scientists, who, in most cases, would prefer to focus on their research rather than spending time dealing with platform configuration concerns.

In this article, we advocate for a system management framework that aims to automatically set up the whole run-time environment according to the applications' needs. The main difference with regards to usual approaches is that they generally only focus on the software layer whereas we address both the hardware and the software expectations through a unique system. For each application, scientists describe their requirements through the definition of a *virtual platform (VP)* and a *virtual system environment (VSE)*. Relying on the VP/VSE definitions, the framework is in charge of (i) the configuration of the physical infrastructure to satisfy the VP requirements, (ii) the set-up of the VP, and (iii) the customization of the execution environment (VSE) upon the former VP. We propose a new formalism that the system can rely upon to successfully perform each of these three steps without burdening the user with the specifics of the configuration for the physical resources, and system management tools. This formalism leverages Goldberg's theory for recursive virtual machines (Goldberg, 1973 [6]) by introducing new concepts based on system virtualization (*identity, partitioning, aggregation*) and emulation (*simple, abstraction*). This enables the definition of complex VP/VSE configurations without making assumptions about the hardware and the software resources. For each requirement, the system executes the corresponding operation with the appropriate management tool.

As a proof of concept, we implemented a first prototype that currently interacts with several system management tools (e.g., OSCAR, the Grid'5000 toolkit, and XtreemOS) and that can be easily extended to integrate new resource brokers or cloud systems such as Nimbus, OpenNebula, or Eucalyptus, for instance.

1. Introduction

Nowadays, computational scientists execute parallel or distributed applications and try to scale up to get more results or greater accuracy. Accordingly, they use more and more distributed resources, using available high-performance computing (HPC) systems such as clusters, grids, or even clouds, such as the latest Amazon EC2 computing instances [1].

Although many system management tools have been studied and developed for these platforms, they have limitations: (i) most of them have been designed to address specific characteristics of their target platform; (ii) the resource management is mainly done by partitioning, i.e., batch scheduled systems are typically used; and (iii) applications must be adapted to the infrastructure in most cases.

The first two points create some limitations for both the management and the usage of the different platforms, whereas the third point can lead to important and expensive efforts for adapting and porting applications to the target platform (portability criteria).

☆ This work is done in the context of the INRIA SER-OS associated team — http://www.irisa.fr/myriads/ser-os/.
* Corresponding author.
E-mail addresses: Jerome.Gallard@inria.fr (J. Gallard), adrien.lebre@emn.fr (A. Lèbre), Christine.Morin@inria.fr (C. Morin), naughtont@ornl.gov (T. Naughton), scottsl@ornl.gov (S.L. Scott), valleegr@ornl.gov (G. Vallée).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.06.003

From our point of view, the best way to address the challenges associated with the use and the management of distributed resources is to focus on the *scientists' efficiency*: based on what the scientists want to execute, the system should provide adequate mechanisms to (i) abstract the complexity of distributed architectures (in terms of heterogeneity of computational resources); and (ii) provide the requested environment (considering both hardware and software expectations).

Leveraging the latest virtualization capabilities and advanced resource management systems, we propose to investigate how these challenges can be tackled in large-scale distributed systems in order to automatically and transparently (i.e., implicitly[1]) configure the physical resource, and set up and customize the execution environments. We advocate that *flexibility* is the key element to meet this goal: *flexibility* of a computing system is the capacity for this system to be configured and reconfigured automatically according to the needs of applications.

Considering that several configuration/reconfiguration operations may have to be executed to correctly set up a scientist's requested environments, we propose to decompose a computing system into four layers: (0) *the physical resources*, (1) *the physical resource manager*, (2) *the execution environment layer*, and (3) *the application layer*. Layers 0 and 1 concern the hardware and its management, whereas layers 2 and 3 address software concerns. Fig. 1 illustrates a simple use case: a request for a 64-bit Windows system with 32 GB of memory (32 GB 64-bit) and particular libraries installed. According to the available resources, the system management tool should configure the infrastructure before satisfying this demand. Several cases can occur.

- A complete run-time environment similar to the requested one is available and it is delivered to the user.
- A 32 GB Windows node is available but without the expected libraries: the system should customize the software layer with respect to the user's expectation before delivering the environment.
- A 32 GB non-Windows node is available: the system should leverage system virtualization technologies to provide and customize the requested run-time.
- There is no 32 GB node: the system should leverage aggregation capabilities such as single system image (SSI) proposals (Kerrighed [2] or ScaleMP [3]), to first build such a 32 GB node, and second complete the configuration of the system.

In our example, the physical nodes are either 4 GB 32-bit Windows or 32 GB 64-bit Debian systems. The system can reconfigure either the Windows or the Linux nodes. We arbitrarily chose to illustrate the instantiation of the requested environment on the Windows infrastructure. Fig. 1(b) shows the operations that must be performed to configure layer 0 and to provide an adequate view of the resources at layer 3.

If we briefly compare such an approach with the available cloud computing "Infrastructure as a Service" (IaaS) platforms, we propose to go one step further in terms of flexibility by allowing the reconfiguration of the infrastructure itself in order to meet user expectations. Such operations are not possible on IaaS platforms where user environments are limited to the underlying virtualization system and to the physical limitations of each node. Another advantage of such a proposal consists in using system virtualization only when it is required. In other words, it gives the possibility for scientists to query physical nodes instead of virtual machines if some nodes meet the physical requirements. In such a case, our system would reconfigure nodes at a low level by leveraging system management tools [4,5].

[1] Note that throughout the remainder of this paper the term "transparent" is synonymous with "implicit".

The level of flexibility that such a system may provide is intrinsically linked to the system tools that it can leverage and the way in which it can combine them. In other words, each reconfiguration process is composed of several actions that the system should perform in a strict order (for instance, deploying hypervisors before launching virtual machines, …). To automatically infer combination rules, we propose to refine the Goldberg definitions [6] of emulation and virtualization, and extend the scope by introducing three new concepts: *abstraction*, *aggregation*, and *identity*. These will enhance the existing Goldberg definitions, so that users can formalize a larger set of computing environments.

Regarding software layers, we propose a solution based on the concept of package sets [7] that enables the on-demand creation, configuration, and management of execution environments.

We propose to unify the definitions of hardware and software expectations around one method relying on two building blocks: (i) *virtual platforms (VPs)* for layers 0 and 1, and (ii) a *virtual system environment (VSE)* for layers 2 and 3. A VP allows one to describe application needs in terms of hardware resources, which can be physical or virtual, whereas a VSE allows one to describe application needs in terms of software environment and software configuration. The management tool presented in this paper instantiates VPs on top of physical resources and VSEs on top of VPs, for a transparent execution of applications.

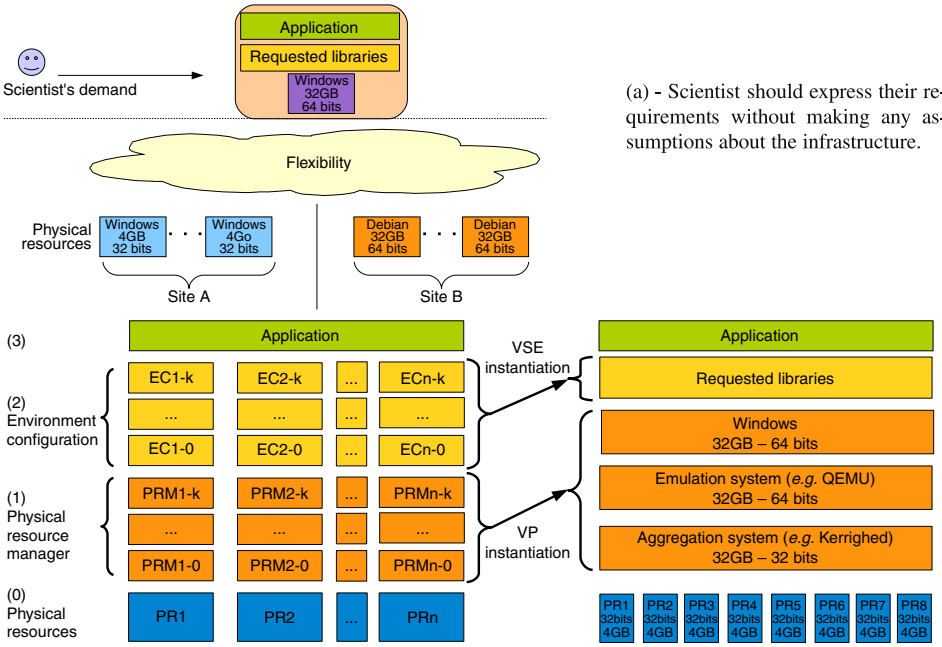
In addition to providing a more transparent and convenient way for scientists to manage their applications throughout different platforms, administrators can also leverage both VP and VSE descriptions to define site-specific constraints and provide more fine grained management policies.

The remainder of this paper is organized as follows. Section 2 introduces existing solutions for the management of clusters, grids, and clouds. Our proposal, which improves the flexibility for the usage of computing systems, is presented in Section 3. Section 4 presents the method whereby scientists may interact with the system for the specification of their needs in terms of execution environment and platform configuration. Section 5 presents the current implementation of our prototype, and Section 6 presents a general use case of our system. Finally, Section 7 concludes and presents some future work.

2. Management of distributed platforms

The most common way of using distributed architectures is via distributed resource managers, a system whereby a *static* set of resources (partition of the resources) is assigned to an application for a bounded amount of time [8]. Torque [9] and OAR [10] are well-known examples. Besides only enabling partitioning of resources, this approach does not address the adaptation concern of either run-time environments or applications with regards to the physical infrastructure. In the best case, these systems may rely on specific languages such as the *job submission description language (JSDL)* [11,12] or the *architecture description language (ADL)* [13,14] to describe the application's expectations. These generally include the description of the requirements in terms of both hardware and software resources that may be used by the management system to find the appropriate resources and start the application transparently when the job is scheduled. However, reconfiguration operations such as OS redeployment are not applied if any resource matches.

Many significant organizations focus on providing a highly configurable environment, which meets the application's needs. ALADDIN-G5K, also known as Grid'5000, is a French national grid platform where scientists can deploy their environment and have full control of the resources assigned to them [5]. This infrastructure is a good example of resource flexibility.



(a) - Scientist should express their requirements without making any assumptions about the infrastructure.

(b) - In the case where there is no resource that directly matches the request, several operations must be performed. In the current example, 4 steps are mandatory: deploy a system that supports the aggregation of resources, deploy a system that supports the use of a virtual machine, boot the requested Windows system and configure it.

Fig. 1. Computing system layers.

However, the environment should be configured manually for each experiment of each scientist. Although some work is currently in progress to enable the automation of the main operations, the system does not provide a generic way to express user requirements. Such a concern may be addressed by the use of systems such as Puppet [15], Chef [16], or Smartfrog [17] that support the configuration of physical machines according to a description file [18,19]. However, these solutions deal with administrator concerns: i.e., they have not been designed for fine grained use on a per application/per user basis. In other words, it is impossible to automatically generate complex VP/VSE configurations that may be spread across one or several sites.

Another well-known way to address the flexibility with regard to the underlying infrastructure consists in using system virtualization technologies [20,21]. This idea has been fundamental in the adoption of the IaaS cloud computing model, and several systems such as Nimbus [22], OpenNebula [23], and Eucalyptus [24] have been designed to deploy and manage virtual environments on distributed infrastructures. Although virtualization enables environment customization, users have to deal with the specifics of each system virtualization solution since most of the IaaS frameworks do not provide mechanisms to redeploy hypervisors in order to accommodate different virtual environments. This limitation becomes particularly significant when several IaaS platforms are federated into a common infrastructure and when users want to run particular environments. Claudia [25] is a service abstraction layer for the usage and management of distributed infrastructures, which provides a partial solution to the issue of cloud federation. However, this system does not tackle the issue related to the underlying system virtualization technologies, and each virtual environment can only be started on the appropriate (i.e., compatible) hypervisors. The *open virtualization format* (OVF) [26] standard is a platform-independent format for virtual machines (VMs) that aims to solve this current limitation. Although the OVF will improve flexibility by enabling the deployment and the execution of VMs upon different kinds of hypervisor, current and mid-term IaaS approaches [27] will face the partitioning concern where each virtual machine will have to be deployed on a unique physical node, large enough to host the requested environment.

More generally, our proposal does not focus only on IaaS cloud infrastructures. It addresses the emerging concepts of EaaS (Everything as a Service) and EaaR (Everything as a Resource), in which all tools (hardware/software) of a distributed system can be configured, used, and reconfigured on the fly in order to meet user expectations. In other words, our system does not focus only on the configuration and reconfiguration of VMs: it targets the configuration and the reconfiguration of the entire (physical) infrastructure, starting from the raw hardware up to the highest software components. At a coarse grain, it can be viewed as a meta-management system that may leverage all presented systems to deploy/configure and execute on the fly advanced VP/VSEs configurations.

3. A new approach for system management

Clusters, grids, and clouds are by nature different computing platforms: some span across multiple sites and multiple administrative domains, whereas others are restricted to a single site and a single administrative domain. We aim at federating tools for the management of such platforms in order to (i) enable more flexibility in the usage and the management of the physical resources, and (ii) automatically adapt the execution environment to the application. To accomplish this goal, we introduce new abstractions through the definition of a new formalism. This formalism enables the description of resources and execution environment requirements for a given application, without making any assumptions about the physical resources.

3.1. Computing environment layers

We assume that a computing system may be divided into four layers: (0) *the physical resources*, (1) *the physical resource manager*,

(2) *the execution environment layer*, and (3) *the application layer*. Assuming that layer 0 is provided by the physical infrastructure and that layer 3 represents the targeting environment requested by the end-user, the major objective of our system consists in using appropriate system tools to transparently configure and set up layers 1 and 2. Since several operations can be required to correctly set up each of these two layers, we need a formalism to describe each mandatory operation. In other words, we need a formalism that allows the user to describe the compositional structure of any computing environment. Therefore, we propose to refine the Goldberg proposal that relies on a similar decomposition of a computing environment into distinct layers.

Relying on this formalism, the system will be able to combine distinct system management tools to successively perform each step of the reconfiguration process with the ultimate goal being to transparently deliver the requested environment.

3.2. Goldberg's classification

Goldberg proposed a formalization of the virtualization concept, which relies on two functions, ϕ and f [6]. The function ϕ makes the association between processes running in the VM and resources exposed within the VM, whereas f makes the association between resources allocated to a VM and the bare hardware. Restated, ϕ maps processes to the virtual machine and f maps the virtual machine to the physical resources (or other virtual resources in the case of recursion, as discussed below). Functions ϕ and f are independent.

3.2.1. Definition of the f function

Let $V = \{v_0, v_1, \ldots, v_m\}$ be the set of virtual resources, and let $R = \{r_0, r_1, \ldots, r_n\}$ be the set of resources of the physical hardware; Goldberg defines $f : V \rightarrow R$ such that if $y \in V$ and $z \in R$ then $f(y) = z$, if z is the physical resource for the virtual resource y.

3.2.2. Recursion

Recursion could be reached by interpreting V and R as two adjacent levels of virtual resources. Then, the real physical machine is level 0 and virtual resources are level n. As a consequence, f does the mapping between level n and level $n+1$. If $f_1 : V_1 \rightarrow R$, $f_2 : V_2 \rightarrow V_1$, then, a level 2 virtual resource name y is mapped into $f_1(f_2(y))$ or $f_1 o f_2(y)$. Then, Goldberg generalized this case with n-recursion: $f_1 o f_2 o \cdots o f_n(y)$.

3.2.3. Definition of the ϕ function

Let $P = \{p_0, p_1, \ldots, p_j\}$ be the set of processes. Goldberg defines $\phi : P \rightarrow R$ such that if $x \in P, y \in R$ then $\phi(x) = y$, if y is the resource for the process x.

3.2.4. Execution of a virtual machine

Running a process on a VM means running a process on virtual resources. Thus, if processes $P = \{p_0, p_1, \ldots, p_j\}$ run on the VM composed of virtual resources $V = \{v_0, v_1, \ldots, v_m\}$, then $\phi : P \rightarrow V$. The virtual resources, in turn, are mapped into their equivalents: $fo\phi : P \rightarrow R$. From the previous statement, Goldberg defines the execution of a VM: $f_1 o f_2 o \cdots o f_n o \phi$.

3.2.5. Virtualization versus emulation

Performance was a key characteristic associated with virtualization as defined by Goldberg [28]. This is primarily where the distinction resides between virtualization and emulation. An inherent criterion of virtualization is that the majority of the execution must take place on the underlying hardware, which distinguishes VMs from simulators and emulators, the latter being more focused on running alternate execution platforms, potentially distinct from the native and non-existent or unavailable aspects that are emulated through software. It is interesting to note that modern hypervisors have employed some emulation-like mechanisms to support virtualization on hardware platforms that were not natively conducive to virtual machines, e.g., Xen's para-virtualization on x86 hardware.

3.3. Refinement of Goldberg's classifications

The theory proposed by Goldberg serves as the foundation for system-level virtualization, but we propose to refine it in order to define any computing system. With this refinement, it is possible to decouple the description of applications from the actual execution platforms, which ultimately abstract the computing platforms. It is then possible to develop new system tools which aim at adapting the computing platform to the application rather than requiring scientists to handle technical aspects related to the execution of their application on target HPC platforms.

3.3.1. Refinement of f

In this paragraph, we present a formalism for the description of combinations of resources and resource managers (layers 0 and 1). This formalism relies on a refinement of the f function of Goldberg, which was introduced in our previous work [29,30].

Refinement proposal. We propose to extend Goldberg's definitions related to virtualization and emulation, and then apply them to any kind of computing system. In addition, and for the rest of the paper, all mathematical sets we use follow the Zermelo–Fraenkel set theory, with the axiom of choice (ZFC).

Definition 3.1 (*A System*)**.** We define a system "S" with a resource "R" with which we apply the "f" function between level "$n+1$" and level "n" with $S = (R, f, n+1, n)$. For instance, with a system "S" with a resource "*computer*", we apply the function "*operating_system*" between level "$n+1$" and level "n":

$$S = (computer, operating\_system, n+1, n).$$

Definition 3.2 (*Granularity*)**.** The granularity of a system is defined by the fact that a system can be studied as a single entity or be broken down into subsets. For instance, it is possible to study the system "$S1$": $S1 = (computer, operating\_system, 1, 0)$, but it is also possible to study the system $S2 = (memory, operating\_system, 1, 0)$.

Definition 3.3 (*Set of Attributes*)**.** We define three sets of attributes related to resources. (•) *A capacity attribute* (*attributeC*) is a capacity of a given resource; for instance, a computer can have a 2 GB capacity attribute for the memory. We denote $C_{R_n} = \{attributeC_{1_n}, attributieC_{2_n}, \ldots, attributeC_{k_n}\}$ the set of capacity attributes for a resource "R" at level "n". In addition, $C_{R_n} \subset C$, with C being the set of all capacity attributes for a given resource. (•) *A functionality attribute* (*attributeQ*) refers to a functionality of the resource; for instance, a CPU can have a functionality attribute "*add*". We denote $Q_{R_n} = \{attributeQ_{1_n}, attributeQ_{2_n}, \ldots, attributeQ_{k_n}\}$ the set of functionality attributes for a resource "R" at level "n". In addition, $Q_{R_n} \subset Q$, with Q the set of all the functionality attributes a resource can have. (•) *A status attribute* (*attributeE*) refers to a status of the resource; for instance, a hard disk can have a status attribute "*ext2*". We denote $E_{R_n} = \{attributeE_{1_n}, \ldots, attributeE_{k_n}\}$ the set of status attributes for a resource "R" at level "n". In addition, $E_{R_n} \subset E$, with E being the set of all the status attributes of a given resource.

Definition 3.4 (*Function of Attributes*)**.** The "f" function characterizes the transformation of a resource between levels "n" and "$n+1$". We propose a refinement based on three functions: (i) c, a

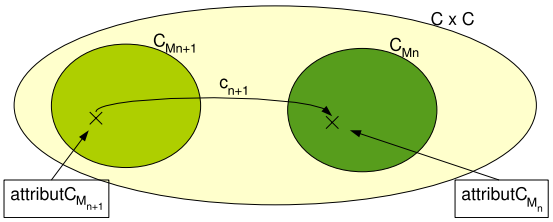


Fig. 2. Representation of the capacity attributes for a system "S": "S"= $(M, c_{n+1}, n+1, n)$.

function from the set of capacity attributes at level "$n+1$" to a set of capacity attributes at level "n" for a resource "R": $c_{R_{n+1}} : C_{R_{n+1}} \rightarrow C_{R_n}$; (ii) q, a function from the set of functionality attributes at level "$n+1$" to a set of functionality attributes at level "n" for a resource "R": $q_{R_{n+1}} : Q_{R_{n+1}} \rightarrow Q_{R_n}$; and (iii) e, a function from the set of status attributes at level "$n+1$" to a set of status attributes at level "n" for a resource "R": $e_{R_{n+1}} : E_{R_{n+1}} \rightarrow E_{R_n}$.

Fig. 2 illustrates how the refinement can be used with two sets of capacity attributes for a resource "M" between levels "$n+1$" and "n". For the rest of the paper, we use the following notation.

- Given set A and set B, we denote $A \subset B$ if $\forall x(x \in A \Rightarrow x \in B)$.
- Given set A and set B, we denote $A = B$ if $(A \subset B) \wedge (B \subset A)$.
- Given set A and set B, we denote $A \neq B$ if $\neg(A = B)$.

Relying on the previous definitions, we redefine virtualization and emulation by introducing four subsets: identity, partitioning, aggregation, and abstraction.

Virtualization. Based on the concept of attributes, we propose the refinement of the f function, which extends the definition of virtualization. This definition is directly derived from Goldberg's definition: the non-protected part of the code at level "$n+1$" is executed directly at level "n".

Definition 3.5 (*Virtualization*)**.**

$$\textit{Virtualization} \Leftrightarrow (Q_{R_{n+1}} = Q_{R_n}) \wedge (E_{R_{n+1}} = E_{R_n}).$$

Definition 3.6 (*Virtualization-Identity or Identity*)**.** The resources exposed at level "$n+1$" are the same as those available at level "n": *Identity* $\Leftrightarrow$ (*Virtualization*) $\wedge$ $(C_{R_{n+1}} = C_{R_n})$.

Definition 3.7 (*Virtualization-Partitioning or Partitioning*)**.** The capacity attributes at level "$n+1$" are a subset of the capacity attributes at level "n": *Partitioning* $\Leftrightarrow$ (*Virtualization*) $\wedge$ $(C_{R_{n+1}} \subset C_{R_n})$.

This means that capacity attributes at level "$n+1$" are a subset of the capacity attributes at level "n".

Definition 3.8 (*Virtualization–Aggregation or Aggregation*)**.** At least two elements belonging to the set of capacity attributes at level "$n+1$" are provided by two distinct sets of capacity attributes at level "n": *aggregation* $\Leftrightarrow$ (*Virtualization*) $\wedge$ $(C_{R_{n+1}} \subset \uplus_{i \geq 2} C_{Ri_n}) \wedge (\exists(x, y) \in C_{R_{n+1}}, c_{n+1}(x) \in C_{Ri_n} \wedge c_{n+1}(y) \in C_{Rj_n}, i \neq j)$.

Fig. 3 presents an example of aggregation.

3.3.1.1. Emulation. Emulation can be of two types: (i) emulation-simple (or emulation), and (ii) emulation-abstraction (or abstraction).

Definition 3.9 (*Emulation-Simple or Emulation*)**.** According to Goldberg's definition, emulation and virtualization are different because a microcode is required to execute the emulated code on the physical hardware. *Emulation* $\Leftrightarrow$ $\neg$(*Virtualization*). We define "*emulationQ*" to be the emulation function that represents

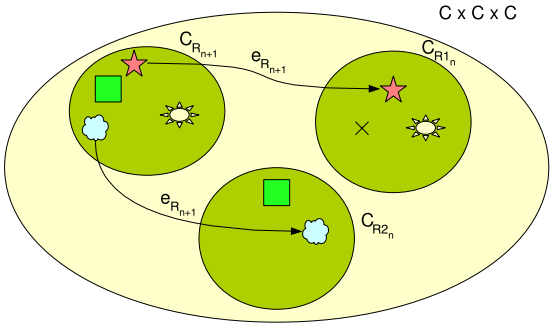


Fig. 3. Aggregation.

the microcode action for the functionality attributes: *emulationQ* : $Q_{R_n} \rightarrow Q_{R_n}$. We also define "*emulationE*" to be the emulation function that represents the microcode action for the status attributes: *emulationE* : $E_{R_n} \rightarrow E_{R_n}$. Fig. 4 presents an example of emulation.

Definition 3.10 (*Emulation-Abstraction or Abstraction*)**.** We denote *arity*(*fnct*), the arity of the function *fnct*. *Abstraction* $\Leftrightarrow$ (*Emulation*) $\wedge$ ((*arity*(*emulationQ*) $>$ 1) $\vee$ (*arity*(*emulationE*) $>$ 1)).

In that case, *emulationQ* (respectively *emulationE*) provides a logical simplification of level "n" at level "$n+1$".

Summary. Emulation allows one to provide at level "$n+1$" some logical characteristics not available at level "n" by means of a microcode. If these characteristics lead to a logical simplification of level "n", we call it abstraction. Virtualization allows one to provide access from level "$n+1$" to level "n". Virtualization is composed by identity (all the resources at level "$n+1$" are provided to the level "n"), partitioning (a subset of the resources at level "n" is provided to level "$n+1$"), and aggregation (two or more resources at level "n" are provided to level "$n+1$" as a "single" resource). This contribution allows one to describe any computing system independently of the physical resources (see Fig. 5). We based our management tool on these concepts of emulation and virtualization.

3.3.2. Refinement of ϕ

In this section, we focus on the ϕ function, also introduced by Goldberg, with the goal of building and adapting execution environments of scientific applications to a target platform (layers 2 and 3). We base our work on the package set concept [7].

3.3.3. Package sets

Definition 3.11 (*A Package*)**.** A package is an abstraction for the local management of software that aims at easing the installation, configuration, and removal of software. In addition, a collection of "operations" is available for the management of package sets, from the combination of package sets to the calculation of the intersection of package sets. These operations provide a very flexible method to manage execution environments.

Package set combination. It is possible to combine package sets together:

$PackageSet_A \cup PackageSet_B$.

Package set intersection. It is possible to define the intersection of package sets: $PackageSet_A \cap PackageSet_B$. This can be used to identify common software components between two execution environments.

Package set validation. It is possible to ensure that the package set can be correctly combined. This is based on a versioning and a dependency mechanism.

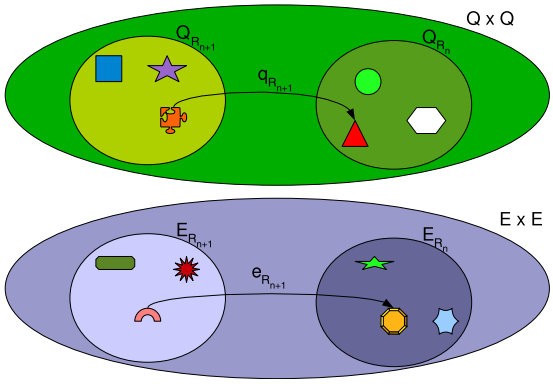


Fig. 4. Emulation.

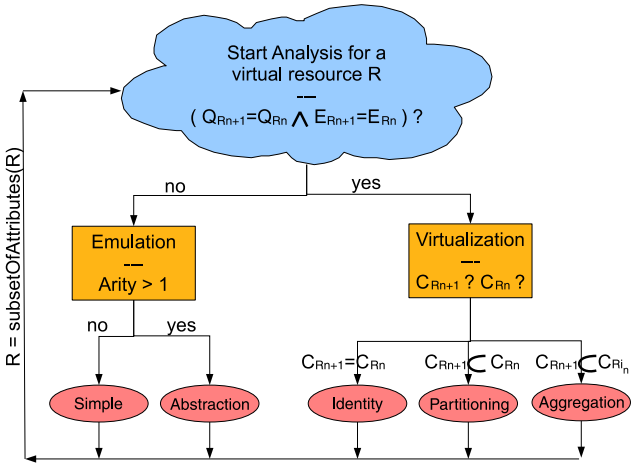


Fig. 5. Refinement of the theory of Goldberg.

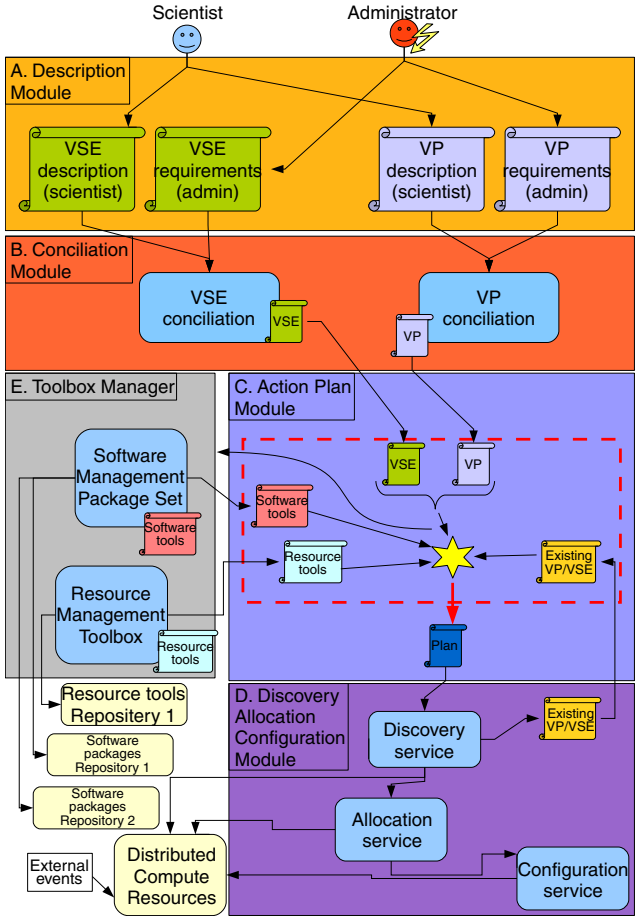


Fig. 6. Architecture overview.

Versioning. It is possible to specify the version of a specific package in a package set. Some standard operators are provided to deal with versioning: *equal to, superior to, inferior to, superior or equal to,* and *inferior or equal to*.

Package sets usage. Package sets define an execution environment in order to create a "golden image", which is agnostic of the target platform execution configuration. Then, the "golden image" can be deployed on the target platform.

4. Architecture overview

We propose a design organized in five major parts (see Fig. 6). The hardware and software requirements are collected by the *Description Module*. Any conflicts regarding these requirements are reconciled by the *Conciliation Module*. Such a module is required to manage restrictions that may be imposed by administrators of the different sites. This leads to an environment configuration step that is used by the *Action Plan Module* to infer the operations that should be performed to build the VP/VSE. According to available resources, the *Discovery, Allocation, and Configuration Module* applies the plan, as necessary, by leveraging tools provided by the *Toolbox Manager*. This last module provides the different tools that may be used to configure the hardware and software resources.

4.1. Description Module: the concept of VP and VSE

In this section, we present the new concept of a VP, and the concept of a VSE. The VSE concept was introduced in [7]. The VP and the VSE allow both the scientists and the system administrators to express their needs and constraints in terms of resources and execution environments, without making any assumptions regarding the physical resources.

4.1.1. Virtual platform (VP)

The VP is a description of the required resources for a given application. The VP description is made without any assumptions regarding the physical resources, which can be locally or geographically distributed. Our deployment tool instantiates the VP and combines all the necessary tools, as described in Section 3.3.1, to build the "custom" virtual platforms.

4.1.2. Virtual system environment (VSE)

A VSE allows one to describe an application's software needs, which can be deployed on any single-site single-administrative domain, as described in Section 3.3.2. The software requirements for a given application are typically constraints on the operating system (OS) and run-time environment (RTE). For instance, an MPI application can be designed to run on top of Red Hat Enterprise Linux 4.0 with LAM/MPI 7.1.3. If those constraints change (e.g., there is an update of the target platform software configuration), most likely the application will have to be modified, i.e., *ported*. Furthermore, it is important to decouple the definition of the application's needs in terms of the RTE and what components system administrators want to have in each environment used by applications. The science resides in the applications and not in the technical details about the requirements for their execution on distributed systems. The VSE is therefore a meta-description of the

run-time environment required by a given application, based on package sets.

4.2. Conciliation Module

The aim of our proposal is to improve the portability of applications across different platforms via the definition of a VP and a VSE. Although scientists can ask for a particular computing environment without making any assumptions about the physical resources, system administrators may have particular expectations too. Consequently, our proposal should be able to "merge" the needs of the scientists and the system administrators. As for the scientists, the proposal gives the capability to the administrators to define specific requirements for a set of VSEs and VPs according to the local policies and the usage of the infrastructure. For instance, a policy can consist of implicitly aggregating some resources to provide simpler VPs for a given group of scientists, or in installing a specific monitoring tool in all VSEs to make the administration process easier.

From the resource point of view, the Conciliation Module has to automatically merge the scientist's requirements and the administrator's constraints in order to provide a viable definition of the VP. From the execution environment perspective, the system should be able to build a single description of the environment taking into account requirements defined by both the scientists and the administrators. The resulting VP and VSE descriptions are sent to the Action Plan Module.

4.3. Action Plan Module

The Action Plan Module is in charge of defining the actions that have to be performed and their order to get the VP/VSE correctly configured.

The definition of the plan is done iteratively: at each iteration, the Action Plan Module asks the Discovery Module whether an adequate environment is already deployed and available on the infrastructure. If the Discovery Module finds a correct environment, the processing stops. If no suitable VP/VSE is available, the Action Plan Module follows a top-down approach to determine the complete plan according to the VPs/VSEs already deployed on the infrastructure. In other words, the Action Plan Module removes at each iteration the highest requirement from the requested VP/VSE stack before once again querying the Discovery Module whether a VP/VSE similar to the new one can be used and reconfigured to meet the final expectations. The operations that are required to finalize the reconfiguration of the VP/VSE are pushed in the plan at each iteration. In the worst case, the loop ends when none of the requirements could be satisfied (in that final case, the Action Plan Module queries raw physical resources).

To summarize, in the best case, the plan contains only the operations that are required to finalize the application configuration (layer 3; see Section 3.1), whereas, in the worst case, it defines the whole set of actions to correctly set up the VP/VSE (from layer 0 to layer 3; a complete example will be given in the next section).

4.4. Discovery, Allocation, and Configuration Module

The Discovery, Allocation, and Configuration Module is in charge of interacting with resource brokers available on the different infrastructures. In addition to providing the discovery capability, it is in charge of the execution of the action plan previously defined (i.e., the instantiation of the VP and VSE).

4.5. Toolbox Manager

The Toolbox Manager aggregates the collection of tools that enable the instantiation of the VPs and VSEs. The toolbox is composed of two classes: the tools dedicated to hardware management (i.e., VP configuration) and the ones that manage the software repositories (i.e., VSE configuration). In both cases, we use the concept of "repository": a repository contains the list of available software that can be used to create a given execution environment. For VPs, we propose a set of mechanisms that can be used to create a particular environment by leveraging capabilities such as resource partitioning, resource aggregation, and resource abstraction (as defined in Section 3.3.1). For VSEs, the repositories provide the different software that can be downloaded, installed, and configured on the VPs.

5. Current implementation

Our prototype is under development, merging and extending existing tools: we base our development on software such as the OSCAR system management tool [4,31], the XtreemOS grid operating system [32,33], the Saline VM manager [34], and several mechanisms of the Grid'5000 toolkit [5]. Experiments to validate the current prototype have been performed on the Grid'5000 infrastructure.

5.1. Description and conciliation modules

Currently, VPs are defined through XML files related to the XtreemOS and the Saline frameworks. VSEs are also defined via XML files describing the set of software packages. OSCAR is in charge of configuring the VSE environment. The reconciliation is currently experimental and relies on virtual organization (VO) policies provided by XtreemOS and defined via XML files. A VO allows the control of the access to the distributed resources: the VP/VSE required by a user belonging to a specific VO will be allocated or built on a set of physical resources belonging to this VO.

5.2. Action Plan Module

The Action Plan Module is currently a shell script that iteratively invokes the Discovery Module and the Toolbox Manager in order to build an adequate plan for the required VP/VSE. Our current prototype focuses on the following three attributes: the execution environment (*env*), the processor type (*cpu*), and the size of the required memory (*mem*).

The following sections present (i) the data structure used for the description of the initial VP/VSE request, (ii) the data structure used for the description of the VP/VSE already present on the infrastructure, (iii) the data structure used for the description of the tools, and (iv) the algorithm used by the action plan module.

5.2.1. The request

The request received from the conciliation module contains the necessary information for the configuration and reconfiguration of the VP/VSE. Table 1 presents two examples of requests with the identifiers *request*[0] and *request*[1]. A request is characterized by two fields: (i) the physical resources (level $n = 0$) and (ii) the requested VP/VSE.

For instance, *request*[0] is a request for a Solaris environment with a SPARC processor and 32 GB of memory. The needs for physical resources are not defined (represented by the "-" symbol). When no specific physical resources are mentioned, the system is allowed to use any physical resources it can.

Table 1
Data structure used for explaining a request.

ID	Physical resources ($n = 0$)	Requested VP/VSE
request[0]	env = -,cpu = -, mem = -	env = Solaris, cpu = SPARC, mem = 32 GB
request[1]	env = Windows,cpu = i386, mem = 2 GB	env = Windows, cpu = i386, mem = 2 GB

Table 2
Data structure for the description of a VP/VSE.

ID	Input	Output
vp-vse[0]	env = -, cpu = -, mem = -	env = Kubuntu, cpu = 1(x64), mem = 5 (GB)
vp-vse[1]	env = -, cpu = -, mem = -	env = *deployable*, cpu = 1(x64), mem = 5 (GB)
vp-vse[2]	env = -, cpu = -, mem = -	env = *deployable*, cpu = 1(x64), mem = 16 (GB)
vp-vse[3]	env = *deployable*, cpu = 2(i386), mem = 16 (GB)	env = Fedora, cpu = 2(i386), mem = 16 (GB)
vp-vse[4]	env = -, cpu = -, mem = -	env = *deployable*, cpu = 1(x64), mem = 16 (GB)

Request[1] is another initial request example. In that request the user asks for a VP/VSE with a Windows environment on top of an i386 CPU and 2 GB of memory. However, in that request, a specific physical resource is mentioned. That means no emulation tool can be used and the VP/VSE should be allocated directly on the bare hardware.

5.2.2. Discovery of the already available VP/VSE

Before building a specific VP/VSE to answer a request, the system searches for an already deployed VP/VSE in the infrastructure. Table 2 presents the data structure used to describe the VP/VSE already present in the infrastructure. This table presents five VPs/VSEs already set up on the infrastructure. Two fields can be highlighted: (i) the output and (ii) the input.

The output. The output field corresponds to the characteristics that the infrastructure is already able to provide. For instance, *vp-vse*[0] means that some resources of the infrastructure are able to provide a Kubuntu OS with an x64 CPU and 5 GB of memory.

The input. The input field corresponds to the characteristics on which the environment is running. For instance, *vp-vse*[0] provides a Kubuntu OS as output running directly on the bare hardware with no way to reconfigure it, whereas *vp-vse*[3] provides a Fedora OS set-up on the top of a *deployable* (reconfigurable) node.

5.2.3. List of available tools

If no VP/VSE already exists or is available on the physical infrastructure, the system tries to build it. In order to do that, the system asks the Toolbox Manager for a list of available tools which can be useful for building the requested VP/VSE.

Tools in this list are described according to the refinement of the Goldberg theory proposed in Section 3.3. The data structure used is the one presented in the Table 3. This data structure is very similar to the one used for the description of the VP and the VSE, which was presented in Table 2. A tool is characterized by (i) an output field and (ii) an input field.

For instance, *tool*[0] is a tool for generating virtualization-identity, which means that this tool ensures that from level n (input) to level $n+1$ (output), the capacity, functionality, and status attributes do not change. This tool is able to form a *deployable* resource at n with z x64 CPUs and y GB of memory to provide a Debian system with z x64 CPUs and y GB of memory.

Another example of a tool is *tool*[1], a tool for virtualization-partitioning. For instance, if a physical resource has 5 GB of memory and if the requested VP needs 2.5 GB of memory, the system will adjust the internal variable n to $n = 1/2$.

5.2.4. Algorithm

The algorithm we implement to perform the plan makes the link between (i) the initial VP/VSE requested, (ii) the already available VP/VSE on the infrastructure, and (iii) the list of available tools. In order to do that, the proposed algorithm makes all the valid combinations possible between the initial request, the available tools, and the VP/VSE available on the infrastructure. A valid combination is an adequate combination of the input and output of the tools on the physical infrastructure to satisfy the initial request.

5.3. Discovery, Allocation, and Configuration Module

The discovery service relies on the XtreemOS resource discovery system. Regarding the allocation and reconfiguration operations, XtreemOS and Saline are used to perform the actions related to the VP configuration (e.g., resource aggregation, VM deployment). The VSE configuration is performed by OSCAR (e.g., operating system configuration) and some XtreemOS modules (e.g., data federation). The current development deals with the federation of external resources that can be provided by public clouds [35].

5.4. Toolbox Manager

Saline exploits QEMU/KVM virtual machines to enable the deployment of virtual clusters. For that, Saline performs the network configuration of the VMs and manages the VMs' life-cycle at the grid level. In addition, Saline makes efficient periodic snapshots of the virtual cluster and can, if needed, restart the virtual cluster on another site of the grid.

XtreemOS provides three important functionalities to our system: (i) the support for federation of physical infrastructures through the concept of VO, (ii) the support for data federation through the distributed grid file system XtreemFS, to federate data provided by distinct physical infrastructures like several clouds, and (iii) the support of resource aggregation through the use of Kerrighed [2], a single system image that aggregates the capacity of several nodes to provide more powerful machines.

The management of OSCAR package repositories is performed by the OSCAR Repository Manager and the OSCAR Package Manager. Currently OSCAR packages cover standard Linux distributions such as CentOS and Debian. OSCAR is able to create "golden images" based on a VSE, which contain all the necessary software.

6. Use case

This section presents a use case to illustrate the capacities provided by our system.

We take the case of a user wanting to run their application (MyApplication) with a specific library (MyLibrary) only available on a Solaris system with 32 GB of memory and a SPARC CPU. The user is accustomed to running their application on a dedicated Solaris node with 32 GB of memory and a SPARC CPU.

However, if this resource becomes unavailable and only four x64 nodes with a Debian OS and 8 GB of memory are available, the user will have to make a choice: (i) the user can wait until the same or another dedicated node with the appropriate characteristics becomes available (this can take hours or days), (ii) the user can adapt their application and required library to the new physical infrastructure, and (iii) the user can use an intelligent system such as the one we propose to adapt the infrastructure (the four nodes) to their needs.

In this use case, we consider the user to take choice (iii): the user wants to use an intelligent system to adapt the infrastructure to their needs. The user makes *request*[0], presented in Table 1. We consider in this use case that the available infrastructure (VP/VSE) and tools are those presented in Tables 2 and 3. In that case, the action plan module will build the following plan (see Fig. 7):

Table 3
Data structure used for the catalog of usable tools.

ID	Input	Output	Type
tool[0]	env = *deployable*, cpu = z (x64), mem = y (GB)	env = Debian, cpu = z (x64), mem = y (GB)	Identity (Debian)
tool[1]	env = Debian, cpu = $n * z$ (x64), mem = $n * y$ (GB)	env = $m*$ Debian, cpu = z (x64), mem = y (GB)	Partitioning, Debian-container lxc ($n < 1$)
tool[2]	env = $m*$*deployable*, cpu = $n * z$ (x64), mem = $n * y$ (GB)	env = Debian, cpu = z (x64), mem = y (GB)	Aggregation, Kerrighed ($n > 1$)
tool[3]	env = Debian, cpu = i386, mem = $n * y$ (GB)	env = *deployable*, cpu = x64, mem = y (GB)	Emulation,QEMUwith$q$_action(x64) = i386 and $n > 0$
tool[4]	env = *deployable*, cpu = z (SPARC), mem = y (GB)	env = Solaris, cpu = z (SPARC), mem = y (GB)	Identity
tool[5]	env = Debian, cpu = x64, mem = $n * y$ (GB)	env = *deployable*, cpu = SPARC, mem = y (GB)	Emulation,QEMUwith$q$_action(SPARC) = x64 and $n > 0$

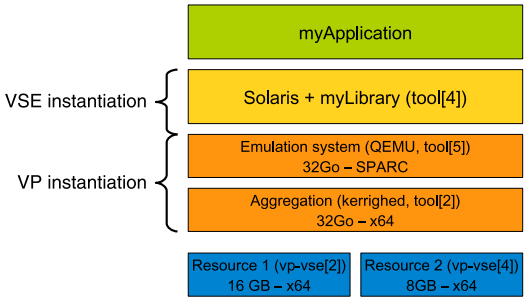


Fig. 7. General use case.

- use of resources *vp-vse*[2] and *vp-vse*[4],
- use of *tool*[2] to install and configure the Kerrighed aggregation tool,
- use of *tool*[5] to install and configure the QEMU emulation tool,
- use of *tool*[4] to deploy a Solaris environment.

7. Conclusion and future work

In this paper, we have proposed a new approach to make the use of distributed infrastructures easier. Our goal is to relieve scientists of the burden of dealing with platform considerations when they run their applications on different infrastructures (clusters, grid, and clouds). We advocate that *flexibility*, i.e. the adaptation capabilities provided by a computing environment to meet the expectations of scientists, is a key concept for abstracting the complexity of distributed systems not only for the software as it is usually applied, but also for the hardware.

We argue in favor of a unique method where scientists may express their expectations in terms of hardware and software by defining respectively a *virtual platform* (VP) and a *virtual system environment* (VSE) without making any assumptions about the infrastructure. Relying on these definitions, we have proposed a system that automatically performs all necessary operations to deploy and to configure the requested VP/VSE according to available resources.

In addition to providing a more transparent and convenient way for scientists to manage their applications throughout different platforms, the configuration process also supports the considerations of administrator requirements. This insures that the deployment of the environments requested by scientists does not compromise the control capabilities of system administrators.

To infer the operations that must be performed to deliver the expected environment, we refine Goldberg's theory for recursive virtual machines. This refinement leads to new abstractions that are used to formally define a larger set of computing environments.

By leveraging several existing tools, we have designed and implemented a first prototype. It supports on-the-fly construction of complex VP/VSE configurations by transparently "adapting" the infrastructure. One advantage of our solution is that it can be extended at any time with additional tools in order to improve the flexibility.

We are working on cost models for the configuration process. The objective is to automatically select the plan that leads to the most efficient VP/VSE when several reconfiguration processes can be performed. Although our system can deliver a VP/VSE configuration by using several kinds of physical infrastructure, we should find a way to produce the most efficient VP/VSE. In the meantime, we are also investigating whether performance can be expressed as another criterion of the VP/VSE. This will enable the user to automatically remove some reconfiguration possibilities. An example is a way to enable scientists to express particular constraints such as: do not use a given virtualization technology.

We are confident that, in addition to performance requirements, many scientists will have other criteria for consideration, such as security or reliability. These and other refinements are topics for future work in this area of systems research.

Acknowledgments

The INRIA team carried out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576. ORNL research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

References

[1] Amazon EC2. URL: http://aws.amazon.com/ec2.
[2] C. Morin, P. Gallard, R. Lottiaux, G. Vallée, Towards an efficient single system image cluster operating system, Future Gener. Comput. Syst. 20 (2004) 505–521. doi:10.1016/S0167-739X(03)00170-5.
[3] D. Schmidl, C. Terboven, A. Wolf, D. an Mey, C. Bischof, How to scale nested openmp applications on the scalemp VSMP architecture, in: Cluster Computing, IEEE International Conference on, vol. 0, 2010, pp. 29–37. doi:10.1109/CLUSTER.2010.38.
[4] J. Mugler, T. Naughton, S.L. Scott, B. Barrett, A. Lumsdaine, J.M. Squyres, Benoît des Ligneris, Francis Giraldeau, C. Leangsuksun, OSCAR Clusters, in: OLS'03: Annual Ottawa Linux Symposium, Ottawa, Canada, 2003. URL: http://www.linuxsymposium.org/2003.
[5] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, I. Touche, Grid'5000: a large scale and highly reconfigurable experimental grid testbed, Int. J. High Perform. Comput. Appl. 20 (2006) 481–494. doi:10.1177/1094342006070078.
[6] R.P. Goldberg, Architecture of virtual machines, in: Proceedings of the Workshop on Virtual Computer Systems, ACM, New York, NY USA, 1973, pp. 74–112. doi:10.1145/800122.803950.

[7] G. Vallée, T. Naughton, H. Ong, A. Tikotekar, C. Engelmann, W. Bland, F. Aderholdt, S.L. Scott, Virtual system environments, in: Systems and Virtualization Management. Standards and New Technologies, in: Communications in Computer and Information Science, vol. 18, Springer, Berlin, Heidelberg, 2008, pp. 72–83. doi:10.1007/978-3-540-88708-9.

[8] M. Margo, K. Yoshimoto, P. Kovatch, P. Andrews, Impact of reservations on production job scheduling, in: E. Frachtenberg, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 4942, Springer, Berlin, Heidelberg, 2008, pp. 116–131. doi:10.1007/978-3-540-78699-3_7.

[9] Torque Resource Manager. URL: http://www.clusterresources.com/products/torque-resource-manager.php.

[10] N. Capit, G.D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, O. Richard, A batch scheduler with high level components, in: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, 2005.

[11] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva, Job submission description language (JSDL) specification, Version 1.0, November 2005. URL: www.gridforum.org/documents/GFD.56.pdf.

[12] I. Rodero, F. Guim, J. Corbalan, J. Labarta, How the JSDL can exploit the parallelism? in: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, CCGRID'06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 275–282. doi:10.1109/CCGRID.2006.55.

[13] D. Garlan, R.T. Monroe, D. Wile, Acme: architectural description of component-based systems, in: Foundations of Component-Based Systems, Cambridge University Press, New York, NY, USA, 2000, pp. 47–67. URL: http://portal.acm.org/citation.cfm?id=336431.336437.

[14] J. Magee, N. Dulay, J. Kramer, Structuring parallel and distributed programs, Softw. Eng. J. 8 (1993) 73–82.

[15] L. Kanies, Next-generation configuration management, in: USENIX—LOGIN, 2006.

[16] Chef: a systems integration framework, built to bring the benefits of configuration management to your entire infrastructure. URL: http://wiki.opscode.com/display/chef/Home.

[17] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, P. Toft, The smartfrog configuration management framework, SIGOPS Oper. Syst. Rev. 43 (2009) 16–25. doi:10.1145/1496909.1496915.

[18] P. Toft, S. Loughran, Configuration description, deployment and lifecycle management working group, CDDLM-WG, Final Report, March 2008. URL: www.gridforum.org/documents/GFD.127.pdf.

[19] OASIS, Solution Deployment Descriptor Specification 1.0, September 2008. URL:.

[20] R. Figueiredo, P. Dinda, J. Fortes, A case for grid computing on virtual machines, in: Distributed Computing Systems, 2003, Proceedings, 23rd International Conference on, 2003, pp. 550–559. doi:10.1109/ICDCS.2003.1203506.

[21] K. Keahey, I. Foster, T. Freeman, X. Zhang, D. Galron, Virtual workspaces in the grid, in: 11th International Euro-Par Conference, Lisbon, Portugal, 2005. URL: http://www.nimbusproject.org/files/VW_EuroPar05.pdf.

[22] K. Keahey, T. Freeman, Contextualization: providing one-click virtual clusters, in: Proceedings of the 2008 Fourth IEEE International Conference on eScience, IEEE Computer Society, Washington, DC, USA, 2008, pp. 301–308. doi:10.1109/eScience.2008.82.

[23] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Comput. 13 (2009) 14–22. doi:10.1109/MIC.2009.119.

[24] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: CCGRID'09: IEEE Intl. Symposium on Cluster Computing and the Grid, Shanghai, China, 2009.

[25] L. Rodero-Merino, L.M. Vaquero, V. Gil, F. Galán, J. Fontán, R.S. Montero, I.M. Llorente, From infrastructure delivery to service management in clouds, Future Gener. Comput. Syst. 26 (8) (2010) 1226–1240. doi:10.1016/j.future.2010.02.013.

[26] DMTF, Open Virtualization Format Specification, January 2010. URL: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf.

[27] L.M. Vaquero, L. Rodero-Merino, R. Buyya, Dynamically scaling applications in the cloud, SIGCOMM Comput. Commun. Rev. 41 (2011) 45–52. doi:10.1145/1925861.1925869.

[28] R.P. Goldberg, Virtual machines: semantics and examples, in: Proceedings IEEE International Computer Society, Conference Boston Massachusetts, 1971.

[29] J. Gallard, A. Lèbre, G. Vallée, C. Morin, P. Gallard, S. Scott, Refinement proposal of the Goldberg's theory, in: A. Hua, S.-L. Chang (Eds.), Algorithms and Architectures for Parallel Processing, in: Lecture Notes in Computer Science, vol. 5574, Springer, Berlin, Heidelberg, 2009, pp. 853–865. doi:10.1007/978-3-642-03095-6_80.

[30] J. Gallard, G. Vallée, A. Lèbre, C. Morin, P. Gallard, S.L. Scott, Complementarity between virtualization and single system image technologies, in: Euro-Par 2008 Workshops—Parallel Processing, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 43–52. doi:10.1007/978-3-642-00955-6_6.

[31] G. Vallée, T. Naughton, S.L. Scott, System management software for virtual environments, in: CF'07: Proceedings of the 4th International Conference on Computing Frontiers, ACM, New York, NY, USA, 2007, pp. 153–160. doi:10.1145/1242531.1242555.

[32] C. Morin, XtreemOS: a grid operating system making your computer ready for participating in virtual organizations, in: ISORC'07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, IEEE Computer Society, Washington, DC, USA, 2007, pp. 393–402. doi:10.1109/ISORC.2007.62.

[33] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenz, B. Matthews, C. Morin, L.P. Prieto, A. Reinefeld, XtreemOS: a vision for a grid operating system, Technical Report 4, XtreemOS European Integrated Projet, May 2008. URL: https://www.xtreemos.org/publications/research-papers/xtreemos-cacm.pdf.

[34] J. Gallard, A. Lèbre, C. Morin, Saline: improving best-effort job management in grids, in: PDP 2010: The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing—Special Session: Virtualization, Pisa Italie, 2010. doi:10.1109/PDP.2010.61.

[35] E.-D. Tirsa, P. Riteau, J. Gallard, C. Morin, Y. Jégou, Towards XtreemOS in the clouds-automatic deployment of Xtreemos resources in a Nimbus cloud, Technical Report RT-0395, INRIA (2010). URL: http://hal.inria.fr/inria-00524843/en/.

Jérôme Gallard received his engineering degree in computer science from the Ecole Nationale Supérieure des Sciences Appliquées et de Technologie (ENSSAT) of Lannion (France) in 2006. Then, he received both his Master and Ph.D. degrees in computer science from the University of Rennes 1 (France), in 2007 and 2011, respectively. During his studies he focused on distributed systems and more specifically on the management of distributed physical resources through virtualization technologies. His main research interests are in distributed operating systems, distributed infrastructures (clusters, grids and clouds), virtualization in distributed environment, flexibility for the management of distributed computing infrastructures.

Adrien Lèbre is an associate professor in the ASCOLA Research Group of Ecole des Mines of Nantes (France). He received his Ph.D. from Grenoble Institute of Technologies and his master's degree from the University of Grenoble. His research activities aim at designing and implementing new distributed systems leveraging recent programming models (such as event or component programming). Currently, he is working on two specific domains: the study of new distributed file system models and the use of virtualization technologies for cluster, grid, and cloud architectures. Since 2008, Adrien has been the chair of the virtualization working group of the Grid'5000 infrastructure. He is also involved in the consortium of the European Marie Curie Initial Training Network (MCITN) "SCALing by means of Ubiquitous Storage (SCALUS)".

Christine Morin received her engineering degree in computer science from the Institut National des Sciences Appliquées (INSA), of Rennes (France), in 1987, and her master's and Ph.D. degrees in computer science from the University of Rennes I in 1987 and 1990, respectively. In 1998, she obtained her Habilitation à Diriger des Recherches in computer science from the Université de Rennes 1. She holds a senior research position at INRIA and leads the Myriads research team on the design and implementation of autonomous distributed systems. She was the scientific coordinator of the XtreemOS project funded by the European Commission. Her main research interests are in operating systems, distributed systems, fault tolerance, and cluster, grid, and cloud computing.

Thomas Naughton is an R&D associate working in the area of high-performance system software. He has been involved in the Open Source Cluster Application Resources (OSCAR) project for several years, serving as a developer, working group chair, and co-chair for the annual OSCAR Symposium. His current efforts are focused in the areas of system-level virtualization and system resilience. Prior to starting at Oak Ridge National Laboratory, Thomas received a M.S. degree in Computer Science from Middle Tennessee State University, a B.S. in Computer Science and a B.A. in Philosophy from the University of Tennessee at Martin. He is currently pursuing a Ph.D. at the University of Reading, England.

Stephen L. Scott is the Stonecipher/Boeing Distinguished Professor of Computing at Tennessee Technological University (TTU), Cookeville, Tennessee, USA, where he holds a joint appointment with the Computer Science Department and the Electrical and Computer Engineering Department. In addition to his professorship at TTU, Dr. Scott also holds a joint appointment as Senior Research Scientist with the Computer Science and Mathematics Division at Oak Ridge National Laboratory, where he leads the System Research Team. Dr. Scott's research interest is in experimental systems, with a focus on resilient high-performance distributed, heterogeneous, and parallel computing. He has published over 140 peer-reviewed papers in the areas of parallel, cluster, and distributed computing and holds both a Ph.D. and an M.S. in computer science.

Geoffroy Vallée is an R&D Associate in the Computer Science Research Group of the Computer Science and Mathematics Division of Oak Ridge National Laboratory (ORNL), USA. He received his Ph.D. from University of Rennes, France, in the framework of an French industrial collaboration between the University of Rennes, INRIA, and EDF. Geoffroy has completed his master's degree from University of Saint-Quentin-en-Yvelines, France. His research interests include research in operating systems, systems, and high availability for high-performance computing.

Geoffroy is one of the initial developers and designers of the Kerrighed Single System Image (http://www.kerrighed.org/) for clusters. He is also the project chair of the Open Source Cluster Application Resource (OSCAR) software (http://www.openclustergroup.org/). Geoffroy is currently doing research on operating systems for exa-scale computing, focusing on high performance and high availability.