Architecture for the Next Generation System Management Tools for Distributed Computing Platforms

Jérôme Gallard, Geoffroy Vallée, Thomas Naughton, Adrien Lebre, Stephen Scott, Christine Morin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Architecture for the Next Generation System Management Tools for Distributed Computing Platforms

Jérôme Gallard — Geoffroy Vallée — Thomas Naughton — Adrien Lèbre — Stephen L. Scott — Christine Morin

N° 7325

May 2010

ISSN 0249-6399 ISRN INRIA/RR--7325--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Architecture for the Next Generation System Management Tools for Distributed Computing Platforms

Jérôme Gallard[*] , Geoffroy Vallée[†], Thomas Naughton[†] , Adrien Lèbre[‡] , Stephen L. Scott[†] , Christine Morin[*]

Thème : Calcul distribué et applications à très haute performance.
Équipe MYRIADS

Rapport de recherche n° 7325 — May 2010 — 28 pages

Abstract: In order to get more results or greater accuracy, computational scientists execute mainly parallel or distributed applications, and try to scale these applications up. Accordingly, they use more and more distributed resources, using local large-scale HPC systems, grids or even clouds. However, in most of cases, the use and management of such platforms is static. Indeed generally, the application has to be adapted to the environment rather than adapting the environment to the applications' needs. In addition, platforms are managed through the concept of time and space partitioning mainly via the use of batch schedulers: time partitioning enables the execution of several applications on a same resources, and space partitioning enables the execution of applications across several distributed resources. This leads to some usage limitations, where applications can only be executed on a subset of the available resources. Therefore, scientists have to manage technical details related to the execution of their applications on each target HPC platforms, which could result in application modifications, rather than focusing on the science.

In this article, we advocate for a system management tool enabling the transparent configuration of the HPC platform and the customization of the execution environment for large-scale HPC systems (such as clusters or MPPs), grids, and clouds. We propose a new approach to manage these systems in a more dynamic

This work is done in the context of the INRIA SER-OS associated team – http://www.irisa.fr/myriads/ser-os/.

[*] INRIA Rennes – Bretagne Atlantique, Rennes, France – firstname.lastname@inria.fr – The INRIA team carries out this research work in the framework of the XtreemOS project partially funded by the European Commission under contract #FP6-033576.

[†] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA – {valleegr, naughtont, scottsl}@ornl.gov – http://www.ornl.gov – 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.

[‡] EMN, INRIA Rennes – Bretagne Altantique, LINA, Nantes, France – adrien.lebre@emn.fr

way, where the resources can be configured and reconfigured automatically and transparently. The proposed solution is not removing the benefit of resource management systems such as batch system (they still provide a well-known interface for job submission), but rather redefine the underlying system capabilities. Our approach is based on a refinement of the concept of emulation and virtualization introduced by Goldberg. Furthermore, the proposed approach leads to the definition of a method that provides a unique interface to scientists for the deployment and management of their applications on HPC platforms. This method is based on two concepts: (i) the *Virtual System Environment (VSE)*, and (ii) *Virtual Platforms (VPs)*.

Key-words: HPC system resource management, Flexibility, Virtualization, Emulation, Distributed Systems, Virtual Platform, Virtual System Environment.

Proposition d'Architecture pour la Prochaine Génération de Systèmes de Gestion des Plates-Formes de Calculs Distribués

Résumé : Les applications de calcul conçues pour être exécutées de manière parallèle ou distribuée peuvent généralement obtenir des résultats plus rapidement ou avec une plus grande précision en augmentant le nombre de ressources de calcul qui leur est allouées. Ainsi, il est de plus en plus courant d'utiliser, des ressources distribuées hétérogènes comme par exemple, des calculateurs à haute performance (HPC, *High Performance Computing*) qui peuvent être des grappes ou des calculateurs massivement parallèles, des grilles, des centrales de calcul (*Cloud*).

Cependant, dans la plupart des cas, les environnements d'exécution étant fixes pour une plate-forme physique, il est nécessaire d'adapter l'application afin de l'exécuter. Ainsi, bien souvent, les scientifiques doivent passer un temps non négligeable dans la gestion technique liée à l'exécution de leur application sur la plate-forme physique, temps qui aurait été sûrement plus utile d'investir dans la science de l'application.

Dans ce document, nous proposons un système de gestion des ressources permettant de gérer les plates-formes physiques ainsi que les environnements d'exécution. Notre approche se fonde sur un raffinement des concepts d'émulation et de virtualisation proposés par Goldberg dans les années 1970. L'approche proposée mène à une méthode fournissant aux scientifiques une interface unique de déploiement et de gestion de leurs applications sur des systèmes distribués. Cette approche repose sur deux concepts: les environnements d'exécution (VSE, *Virtual System Environment*) et les plates-formes virtuelles (VP, *Virtual Platform*).

Mots-clés : Gestion des Systèmes à Haute Performance, Flexibilité, Virtualisation, Emulation, Systèmes Distribués, Plate-Forme Virtuelle, Environnement d'Execution Virtuel.

1 Introduction

Nowadays, computational scientists mainly 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 local large-scale HPC systems (such as clusters or MPPs), grids or even clouds.

Many system management tools have been studied and developed for these platforms but, to the best of our knowledge, all are static:

- in most of cases, the application should be adapted to the target platform, *i.e.,* it is very difficult to switch from a platform to another without making some modifications to the application and/or the new execution environment,
- the management of the platform is mainly done by partitioning the resources, *i.e.,* generally batch scheduler systems are used,
- system management tools are dedicated to a kind of platform.

The first point induces a huge wast of time in order to adapt the application to the platform according to the scientist point of view, and the two other points induce some limitations in the usage and the management of the physical platforms.

We think that the best way to address challenges associated to the management of distributed resources is to focus on the *scientists' efficiency*: based on what the scientists want to execute, the system should provide the adequate capabilities such as tools and mechanisms to abstract the distributed nature of computational resources, tolerate failures and provide the necessary runtimes.

The proposed work addresses these challenges from a system software management tool prospective, with the ultimate goal of enabling the automatic and transparent configuration of the platforms and the customization of the execution environment for large-scale HPC systems.

In this document, we call *flexibility* of a computing system, the capacity for this system to be configure and reconfigured automatically according to the application needs. Considering an action (A) requiring sub-action (B) and (C) in order to be done, in this document, we say a (A) is transparent if it implicitely calls (B) and (C).

To do that, we break down a computing system into several layers: *(0) set of physical resources, (1) physical resource manager, (2) environment configuration, and (3) application.* In this document, we propose to add more flexibility in the usage of those platforms by focusing on the first two layers (layer (0) and (1)). We propose a refinement of the concept of emulation and virtualization introduced by Goldberg in the 1970s. This refinement allows us to leverage the concept of partitioning generally used in HPC systems by adding the concepts of *abstraction*, *aggregation*, and *identity*. These concepts enable the comparison of the physical resources as "Lego" that it is possible to put them together in order to have something more powerful.

In addition, we propose an approach for the on-demand creation of execution environments, by focusing on the last two layers (layer (2) and (3)). This should be done in a transparent way from the scientist point of view.

The proposed approach leads to the definition of a method in order to expose to scientists a unique interface to deploy and manage their applications on HPC

platforms. This method is based on two different concepts: (i) *Virtual Platforms (VPs)* for the layers (0) and (1) and (ii) *Virtual System Environment (VSE)* for the layers (2) and (3). A VP allows one to describe application's needs in terms of resources. These resources can be physical or virtual. A VSE allows one to describe application's needs in terms of software environment and software configuration.

The management tool presented in this document is able to instantiate VPs on top of physical resources and VSEs on top of VPs in order to transparently execute applications.

The remainder of this document is organized as follows: Section 2 introduces existing solutions for the management of HPC systems, grids, and clouds. The proposed approach in order to make computing systems more flexible is presented in Section 3. Section 4 presents a method in which scientists have a unique way to interact with the platforms for the specification of their needs in term of execution environment and platform configuration. Section 5 presents the current implementation of our prototype, and Section 6 shows a typical use case. Finally, Section 7 concludes and presents some future work.

2 Standard Approaches in Order to Manage Distributed and Parallel Platforms

The most common way of exploiting distributed architectures like clusters or grids relies on a reservation scheme where a *static* set of resources (a partition of the resources) is assigned to an application during a bounded amount of time [16]. Torque [2], or Sun Grid Engine [1] are example of well known distributed resource manager.

This model of using platforms, even with the use of back-filling methods [23], leads generally to a coarse-grain exploitation of the architecture since resources are simply reassigned to another application at the end of allocated slots without considering the actual application completion. In the best case, the time slot is lnoger than the execution time of the application and resources are simply under used. In the worst one, running applications are withdrawn from their resources before completion, potentially leading to the loss of all performed computations and requiring to execute again the application from scratch. If the former case is not really critical from the scientist point of view, the latter requires to deal with checkpointing issues to ensure the progress of the jobs. However, to be able to use checkpointing mechanism, scientists generally need to adapt the application to the system. For instance, these solutions are implemented in (i) user space, i.e., at application level (a linkage with special libraries is generally required) [19, 22] or (ii) in kernel space (generally using specific OS) [11, 24]. Ultimately, even if resource managers are generally very popular, they have some limitations: (i) from the resource point of view, they only allow partitioning of the resources, and (ii) the application needs to be adapted to the system.

Many significant organizations focus on providing a highly configurable environment, which meets the needs of the user application. Such approaches have been developed by the GLOBUS alliance with its Workspace Service [12]. However, they do not provide an easy and transparent way to dynamically manage resources.

In the area of Cloud computing, some open-source projects like Nimbus [13], Eucalyptus [18] or SnowFlock [14] allow users to deploy their cloud and manage a virtual cluster on a physical one. However, resources are partitioned and applications should be adapted to the cloud environment.

ALADDIN-G5K, also known as Grid'5000 [4], is a French national Grid Platform used for computer science research in order to experiment all layers of Grid software. Scientists reserve Grid'5000 nodes, and deploy their experiments. The platform is composed of 9 sites distributed around France. Grid'5000 allows scientists to have the full control of the resources assigned to them. The Grid'5000 infrastructure is a good example of resource flexibility. However, it is complicated to domesticate it and natively allows only partitioning of the resources.

Job Description Language (JSDL) [3] and its extension to describe parallel application [21] allows one to describe a non-interactive application. A JSDL file is typically sent to a batch scheduler. However, the description of the application, the application execution environment and their configurations is limited.

Architecture Description Language (ADL) allows one to describe an application without making any assumptions concerning its execution environment. Acme [8] or Darwin [15] are example of ADL languages. This is a good way to separate the description of a complex platform and the complexity of the execution environment. We want to base our work with the same environment management concepts. However we would like to extend this kind of work in order to provide more flexibility to the physical platform.

3 A New Approach for the Management of Distributed and Parallel Platforms

Clouds, grids, clusters and MPPs are by nature different computing platforms: some span across multiple sites, and multiple administrative domains, whereas others are single site and 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, adequate abstractions are mandatory. Therefore, we base our approach on the definition of formalisms that enables the description of resource and execution environment required by a given application, without making any assumption on to the physical resources.

3.1 Resource and Execution Environment Management

We assume a computing system is composed of 4 layers: *(0) physical resources (the platform), (1) physical resource manager, (2) environment configuration, (3) application* (see Figure 1).

The first two layers are linked with the physical resources and its management. The physical resources (layer (0)) corresponds to the resources composing the platform. These resources can be distributed across a single site, like clusters, or across multiple site, like grids. The resource manager (layer (1)) corresponds to the system managing the resources, for instance, an operating system, or a batch scheduler. Of course, the resource manager layer can be

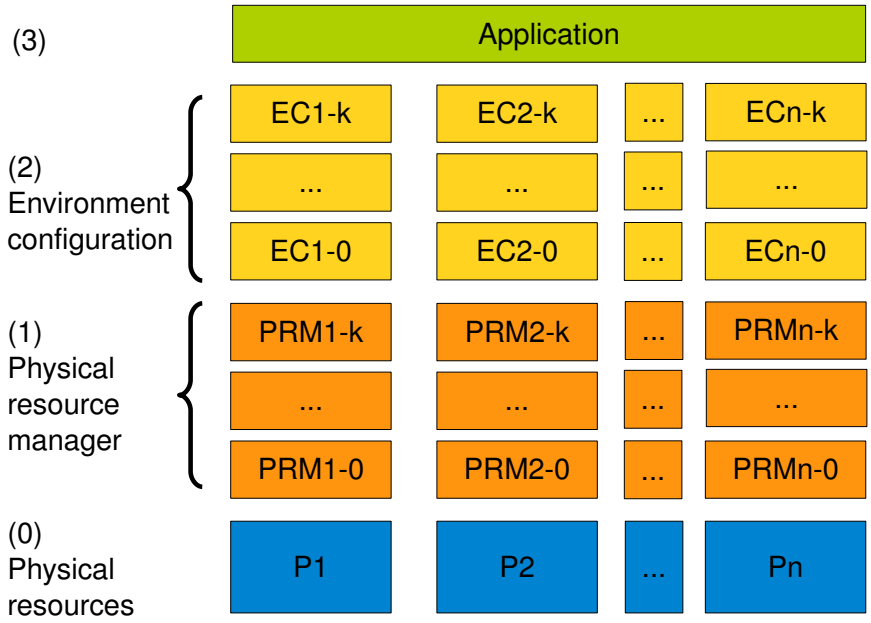


Figure 1: Computing System Layers

break down into sub-layers: one resource manager can be setup on top of another. Currently, most of the resource managers enable resource partitioning. In this document, we leverage this point by introducing new concepts to enable a dynamic management of these physical resources.

The software configuration (layer (2)) corresponds to the configuration of the environment for execution of a given application. The application (layer (3)) is the scientific application.

The abstraction proposed by our management tool aims at hiding the challenges created by the execution of parallel/distributed applications on top of parallel/distributed platforms, from the multi-sites/multi-users case, down to the uni-site/uni-user case.

To summarize, our solution focuses on the three following aspects:

- From a resource management point of view, we need a formalism in order to be able to describe the required platform without making any assumption according to the physical resources.
- From an execution environment point of view, we need a formalism in order to build execution environment adapted to the application, this will enable the adaptation of the execution environment to the application instead of adapting the application to the environment.
- Finally, we need a "driver" to manage these tools and "build" the needed platform, and configure the needed execution environments.

3.2 Formalisation of Virtualization System

To propose a formalism that can be used both for the description of all computing systems without making assumptions about the physical resources, and for the increase of the flexibility for resource management, we propose a refinement of the theory of Goldberg. Before to present our refinement, we briefly introduce its theory.

In 1973, Goldberg proposed a formalization of the virtualization concept. His model relies on two functions, ϕ and f [10, 20]. The function ϕ makes the association between processes running in the VM and resources exposed within the VM; whereas the function f makes the association between resources allocated to a VM and the bare hardware. Functions ϕ and f are totally independent, as ϕ is linked to processes in the VM, f is linked to resources.

Definition of the f function of Goldberg Let:

- $V = \{v_0, v_1, ..., v_m\}$ be the set of virtual resources.
- $R = \{r_0, r_1, ..., r_n\}$ be the set of resources present in the real 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.

Definition of the recursion in the meaning of Goldberg Recursion could be reached interpreting V and R as two adjacent levels of virtual resources. Then, the real physical machine is level 0 and virtual resources is level n. As a consequence, f does the mapping between level n and level $n + 1$.

Recursion example 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: $f_1 o f_2 o ... o f_n(y)$.

Definition of the ϕ function of Goldberg Let: $P = \{p_0, p_1, ..., p_j\}$ 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.

Execution of a virtual machine Running a process on a virtual machine means running a process on virtual resources. Thus, if processes $P = \{p_0, p_1, ..., p_j\}$ run on the virtual machine composed of virtual resources $V = \{v_0, v_1, ..., v_m\}$, then $\phi : P \rightarrow V$. The virtual resources, in turn, are mapped into their equivalents: $f o \phi : P \rightarrow R$.

General Virtual Machine From the previous statement, Goldberg defined the execution of a virtual machine: $f_1 o f_2 o ... o f_n o \phi$.

Virtualization vs Emulation Godlberg defined the main difference between virtualization and emulation by [9, 25]: *virtualization is when the part of the non-protected code is run directly on the bare hardware whereas emulation is when the code (protected or not) is run by means of a micro-code (interface) that is not a physical part of the underlying resource.*

Goldberg Classification Limitations: Need for a Refinement The theory proposed by Goldberg gives the foundation of the system-level virtualization. We want to refine the theory in order to extend it to all computing systems. This will allow us to describe all computing systems with a specific formalism. In addition, we propose a formalism for the adaptation of execution environments.

3.3 Resource Management: Use Resources in a More Flexible Way

In this section, we present a formalism based on a refinement of the theory of Goldberg for the description of all combinations of resources and resource managers, focusing on the layers (0) (the *physical resources*), and (1) (the *physical resource manager*). The proposed formalism allows us to describe a platform break down of distributed resources without any assumption about the physical resources.

Current resource managers partition the available resources. For instance, an operating system makes time-partitioning from the CPU point of view; and a batch scheduler makes space-partitioning between the available resources. In the following Section, we propose a formalism for classifying precisely what kind of tools can be used for the management of resources in a more dynamic manner. This refinement extends the f function of Goldberg, extesion which was introduced in our previous work [6, 7].

3.3.1 Refinement Proposal

We think the Goldberg definitions related to virtualization and emulation should be refined in order to introduce subsets of virtualization and emulation and apply them not only to virtualizated systems but also to all computing systems.

As this section focuses on resource management, we focus on the study of the f function of Goldberg for a system between levels n and $n+1$. In addition, and for the rest of the document, 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*" in which we apply the "*f*" function of Goldberg between the level "*n+1*" and "*n*". We note the system:* $S = (R, f, n+1, n)$.

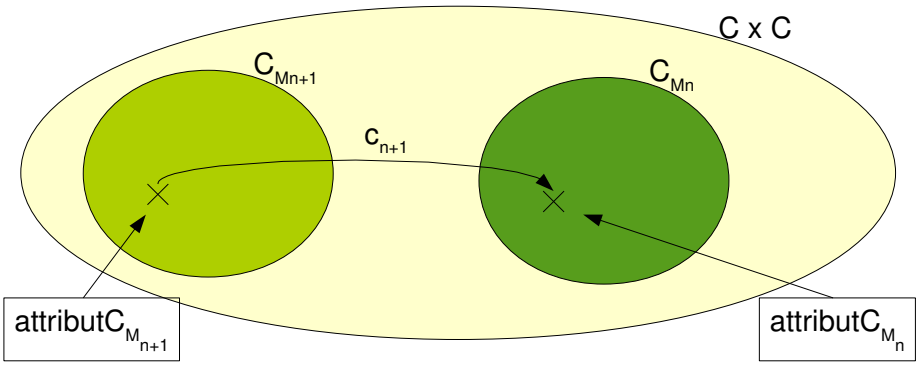
*For instance, a system "*S*" with a resource "*computer*" in which we apply the function "*operating_system*" between the level "*n+1*" and "*n*" is written:* $S = (computer, operating\_system, n+1, n)$.

Definition 3.2 (Granularity) *The granularity of a system is defined by the fact that a system can be studied in its whole or be broken down into subset.*

*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 3 set of attributes linked to a resource:*

- *capacity attributes (attributC) are linked with the notion of space available to a resource, for instance, a computer can have a capacity attribute for the memory "*2 GB*". We note* $C_{R_n} = \{attributC_{1_n}, attributC_{2_n}, ..., attributC_{k_n}\}$



In this example, $c_{n+1}(attributC_{M_{n+1}}) = attributC_{M_n}$.

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

*the set of capacity attributes for a resource "*R*" at level "*n*". In addition, $C_{R_n} \subset C$, with C the set of all the capacity attributes a resource can take.*

- *functionality attributes (attributQ) are linked with the notion of functionality of the resource, for instance, a CPU can have a functionality attribute "*add*". We note $Q_{R_n} = \{attributQ_{1_n}, attributQ_{2_n}, ..., attributQ_{k_n}\}$ 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 take.*

- *status attributes (attributE) are linked with the status of the resource, for instance, a hard disk can have a status attributes "*ext2*". We note $E_{R_n} = \{attributE_{1_n}, attributE_{2_n}, ..., attributE_{k_n}\}$ the set of status attributes for a resource "*R*" at level "*n*". In addition, $E_{R_n} \subset E$, with E the set of all the status attributes a resource can take.*

Definition 3.4 (Function of attributes) *The "*f*" function defined by Goldberg characterized the transformation of a resource between levels "*n*" and "*n+1*". We want to refined it with 3 new functions:*

- *c, a function from the set of capacity attributes at level "*n+1*" to a set of capacity attributes at level "*n*" for a resource "*R*": $c_{R_{n+1}} : C_{R_{n+1}} \rightarrow C_{R_n}$.*

- *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*": $q_{R_{n+1}} : Q_{R_{n+1}} \rightarrow Q_{R_n}$.*

- *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*": $e_{R_{n+1}} : E_{R_{n+1}} \rightarrow E_{R_n}$.*

Figure 2 presents an example of the links between two set of capacity attributes for a resource "*M*" between "*n+1*" and "*n*".

Notations

- let set A et set B, we note $A \subset B$ if $\forall x(x \in A \Rightarrow x \in B)$,
- let set A et set B, we note $A = B$ if $(A \subset B) \wedge (B \subset A)$,
- let set A and set B, we note $A \neq B$ if $\neg(A = B)$,

Considering the system $S = (R, f, n+1, n)$, in the following paragraphs, we present our definition of virtualization and emulation by introducing 4 concepts: identity, partitioning, aggregation and abstraction.

3.3.2 Virtualization

According to the previously described concept of attributes, we propose the refinement of the f function of Goldberg in order to precise the meaning of virtualization and to extend it usage to any computing systems.

Definition 3.5 (Virtualization)
$Virtualization \Leftrightarrow (Q_{R_{n+1}} = Q_{R_n}) \wedge (E_{R_{n+1}} = E_{R_n})$
*This definition is directly derived from the Goldberg definition: the non-protected part of the code at level "*n+1*" is executed directly at the level "*n*".*

Definition 3.6 (Virtualization-Identity or Identity)
$Identity \Leftrightarrow (Virtualization) \wedge (C_{R_{n+1}} = C_{R_n})$
*In case of "*identity*", resources provided at level "*n+1*" are the same that those available at level "*n*".*

Definition 3.7 (Virtualization-Partitioning or Partitioning)
$Partitioning \Leftrightarrow (Virtualization) \wedge (C_{R_{n+1}} \subset C_{R_n})$
*In case of partitioning, capacity attributes at level "*n+1*" is a subset of the capacity attributes a level "*n*".*

Definition 3.8 (Virtualization-Aggregation or Aggregation)
$aggregation \Leftrightarrow (Virtualization) \wedge (C_{R_{n+1}} \subset \uplus_{i \geq 2} C_{Ri_n}) \wedge (\exists (x, y) \in C_{R_{n+1}}, c_{n+1}(x) \in C_{Ri_n} \wedge c_{n+1}(y) \in C_{Rj_n}, i \neq j)$
*In that case, there is at least two elements belonging to the set of capacity attributes at level "*n+1*" providing by two distinct set of capacity attributes at level "*n*".*
Figure 3 presents an example of aggregation.

3.3.3 Emulation

Emulation can be broken down into emulation-simple (or emulation) and emulation-abstraction (or abstraction).

Definition 3.9 (Emulation-Simple or Emulation)
$Emulation \Leftrightarrow \neg(Virtualization)$
According to the definition of Goldberg, Emulation is not Virtualization in the fact that a microcode is used to execute the emulated code on the physical hardware.

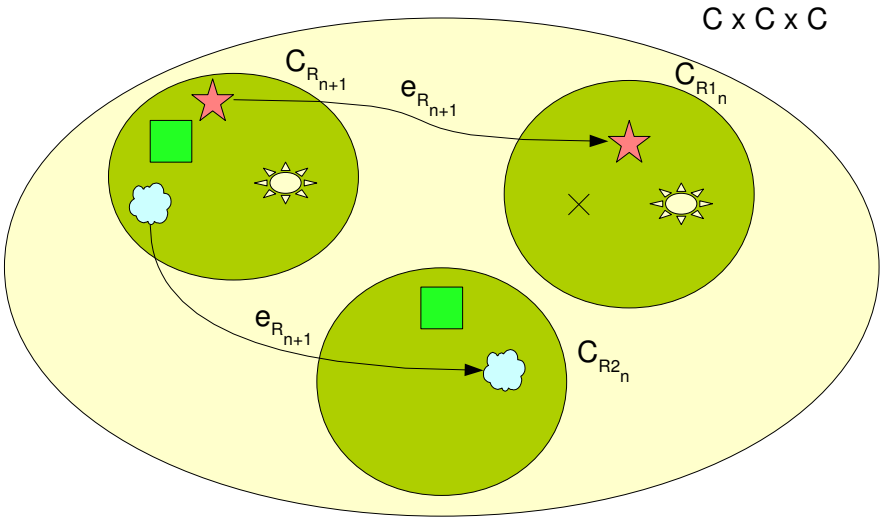


Figure 3: Aggregation

*We define "*emulationQ*" the emulation function representative of the microcode action for the functionality attributes: emulationQ* : $Q_{R_n} \rightarrow Q_{R_n}$.

*We define also "*emulationE*" the emulation function representative of the microcode action for the status attributes: emulationE* : $E_{R_n} \rightarrow E_{R_n}$.

Figure 4 represents emulation.

Definition 3.10 (Emulation-Abstraction or Abstraction) *We note* $arity(fnct)$*, 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 the level "*n*" at the level "*n+1*".*

3.3.4 Summary

We propose an extension of the theory of Goldberg in order to extend the virtualization case to any computing system:

Emulation allows one to provide at level "*n+1*" some logical characteristics not available at level "*n*" by the mean of a microcode. If these characteristics lead to a logical simplification of the level "*n*", we call it abstraction.

Virtualization allows one to provide access from the level "*n+1*" to the level "*n*". Virtualization can be break down in identity (all the resources at the level "*n+1*" are provided to the level "*n*"), partitioning (a subset of the resources at the level "*n*" are provided to the level "*n+1*") and aggregation (two or more resources at the level "*n*" are provided to the level "*n+1*" as a "single" ressource)

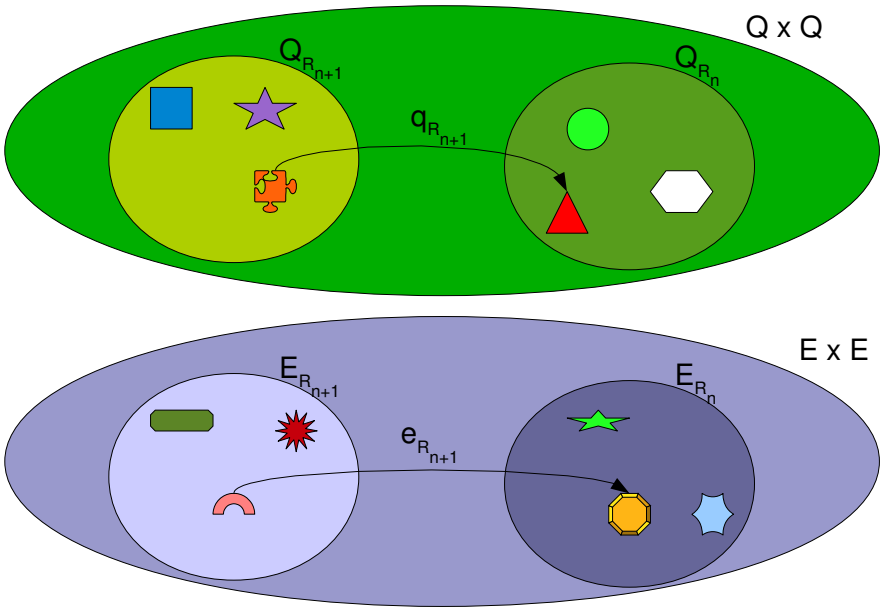


Figure 4: Emulation

This contribution allows one to describe all kind of computing system independantly of the physical resources. We based our management tool on these concepts of emulation and virtualization.

Figure 5 presents a schema of our refinement.

3.4 Execution Environment Management: Adaptation of the System Environment to the Application

In the previous section, we presented a refinement of Goldberg's theory that extends the f function to all computing systems. 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. We based our work on the package set concept [26].

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 a given local system. It means that a collection of "operations" are available for the package set mechanism. From the usage point of view, only a subset of "operations" are important: it is possible to combine package sets and get the intersection of package sets. These operations provides a very flexible method to manage execution environment.*

3.4.1 Package Sets Definition

Package Set Combination It is possible to combine package set together: $PackageSet_A \cup PackageSet_B$. For instance, if system administrators,

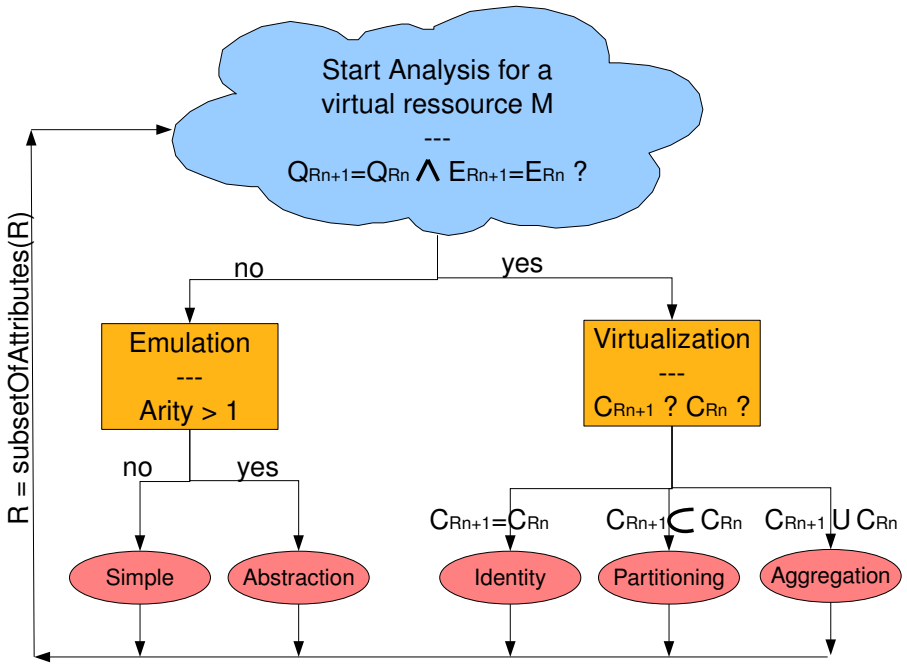


Figure 5: Refinement of the theory of Goldberg

based on local policies, want to include a specific monitoring software in all execution environments, they can specify it on the $PackageSet_{Administrator}$ which will be combined with the $PackageSet_{Application}$ from the scientists for their applications.

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.

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

3.4.2 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. After the creation of the "golden image", it can be deployed on the target platform.

4 Proposed Method

In the previous section, we described a formalism that makes both resource and software management more flexible. In this section, we describe more precisely the methods derived from the proposed approach by presenting the architecture of our management system for HPC platforms. The system is composed of five major parts (see Figure 6):

A. the Description Module: the scientist and the administrator requirements in terms of software and resources,

B. the Conciliation Module: the software and resource requirements conciliator,

C. the Toolbox Module: repositories and list of tools in order to provide flexibility for both the software and the resource management,

D. the Discovery, Allocation and Configuration Module,

E. the Execution Module.

4.1 Description Module: the Concept of VP and VSE

In this section, we present the new concept of Virtual Platform (VP), and the concept of Virtual System Environment (VSE), based on our previous works [26].

VP and VSE have the benefit of allowing both the scientists and the system administrators to express their needs and constraints in term of resource constraints and execution environment constraints withtout making any assumption regarding the physical resources: for instance, the scientist can require a set of resources with the lowest network latency, whereas the system administrators can require all deployed environments should include monitoring capabilities and system-level protection.

To increase the flexibility for both the resource management and the execution environment management, we decide to separate the resource from the environment requirements: no assumption needs to be made regarding the physical resources when the scientist describes its requirements for both resources and the execution environment.

4.1.1 Virtual Platforms

Our work targets the management of several platforms which could be local or geographically distributed. The distribution and the heterogeneity of the resources lead to the complexity in order to manage and use these resources. However, this complexity should be hidden to the scientist who, in most cases, does not care about the location and the configuration of these resources. Therefore, we need an abstraction, this is the concept of VP.

The VP is a description of the required resources for a given application. The VP description is made without any assumption regarding the physical resources available on the target platform. Our deployment tool should instantiate the VP and combine all the necessary tools, as described in the Section 3 (tools for emulation, virtualization, partioning, ...) in order to build the "custom" virtual platform.

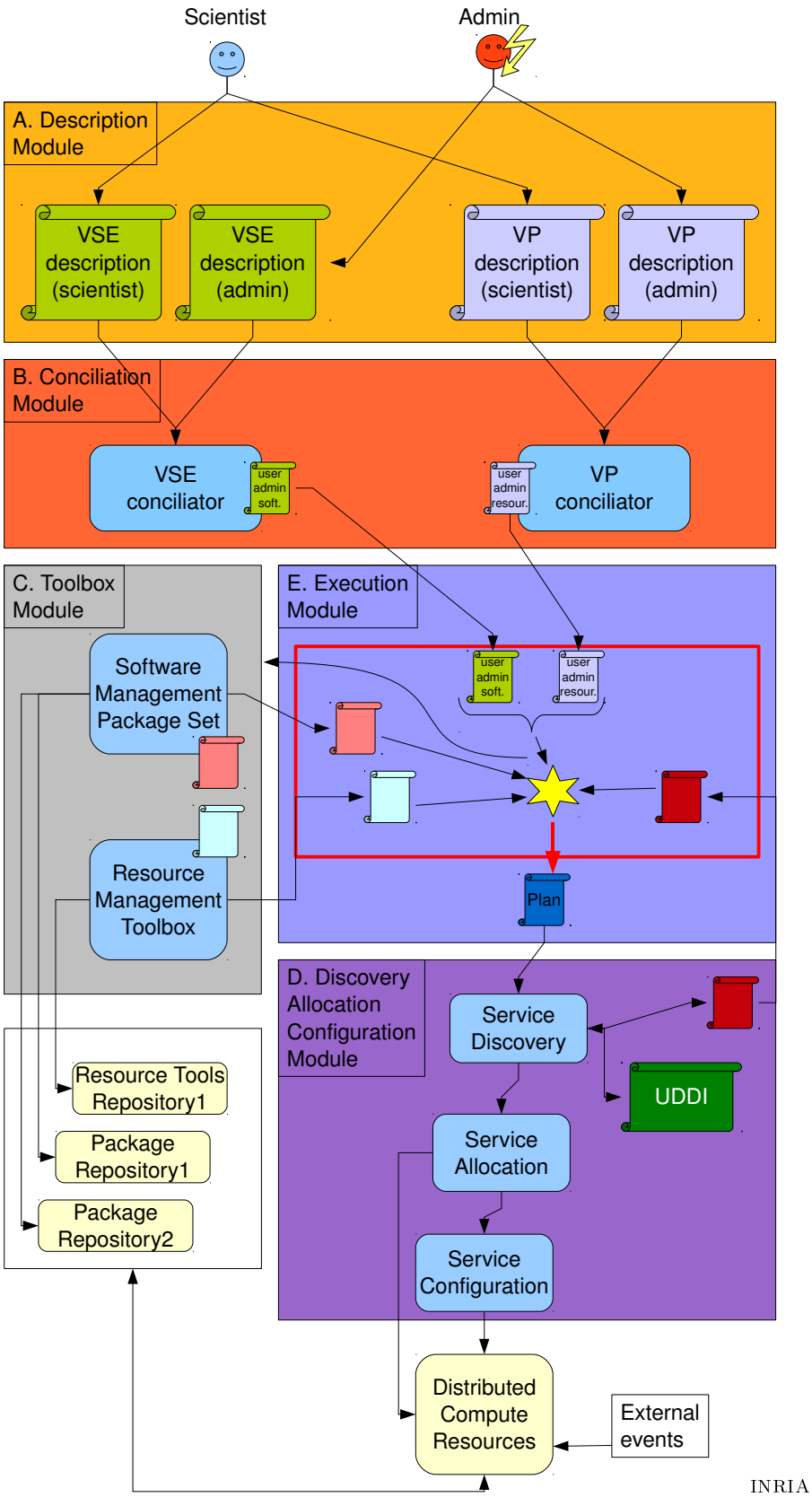


Figure 6: Architecture Overview

This approach has the advantage of enabling negotiation for the configuration of the instantiation of a VP. For instance, in case of conflicts between the requirements of the application and the requirements of the system administrators, negotiations at the VP level are made. In addition, in order to get the best resource allocation, negotiations between sites are also made.

4.1.2 Virtual System Environments

A Virtual System Environment (VSE) allows one to describe an application's software needs, which can be deployed in any kind single site / single administrative domain.

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.,* update of the target platform software configuration), most likely the application will have to be modified, *ported*. Furthermore, it is important to decouple the definition of the application's needs in terms of RTE (scientist application requirements) and what components system administrators want to have in each environment used by applications (system administrator requirements).

The science resides in the applications and not in the technical details about the requirements for the execution of those applications on HPC systems. In other words, application developers should not have to deal with application modifications due to (undesired) general software updates, which do not fit their scientific roadmap.

The VSE is therefore a meta-description of the runtime environment required by a given application. The VSE should be instantiated on the target platform in a transparent way from the scientist point of view (see Figure 7).

Based on the package set management described in Section 3, a scientist can describe a VSE without any assumption regarding the physical resources.

4.2 Conciliation Module: Merging the Scientist and Administrator Requirements

The challenge is to "merge" scientists and system administrators needs: (i) from the execution environment point of view, the system should be able to build a single description of the execution environment containing both requirements from the scientist and the administrator, and (ii) from the resource point of view, the system should be able to configure the platform automatically according to the scientist and the administrator requirements in order to deploy the execution environment.

System Administrator Requirements The system administrator defines requirements for the VSE and the VP according to the local policies regarding the usage of HPC systems. These requirements can be made for a specific scientist or for a group of scientists. As a result, it is possible to define very precise policies for the VSEs and the VPs.

For instance, concerning the VSE requirements the policy can be the installation of a specific monitoring tool on all execution environments. Concerning

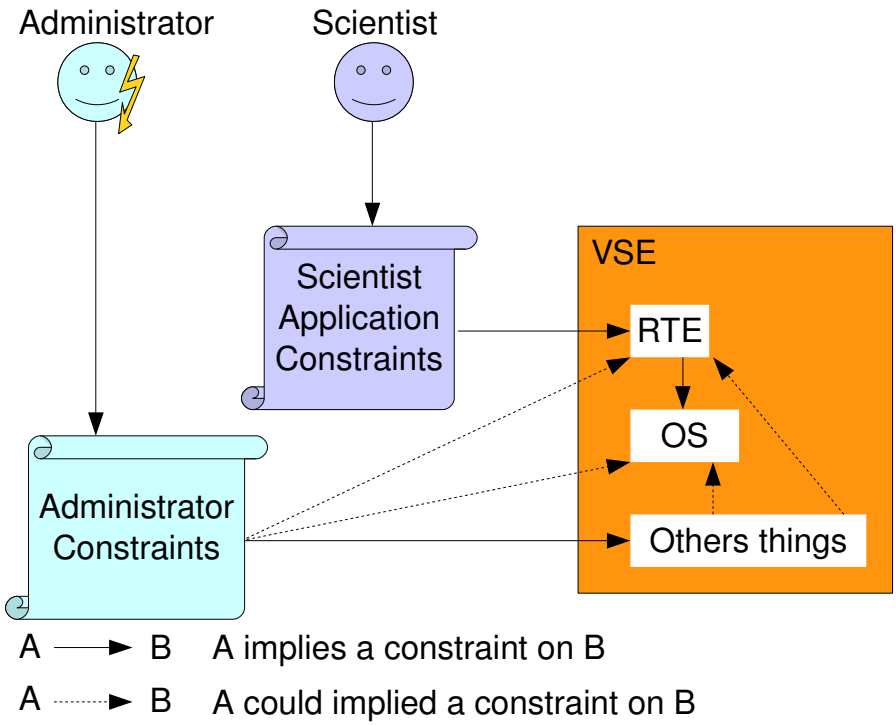

Administrator
Scientist
Scientist Application Constraints
VSE
RTE
OS
Others things
Administrator Constraints
A → B A implies a constraint on B
A ⇢ B A could implied a constraint on B

Figure 7: VSE.

the VP requirement, for instance, it is possible to implicitely aggregate some resources for a given group of scientists.

Scientist Requirements Like administrators, scientists can define VPs and VSEs without any assumptions regarding the physical resources.

Merging of the Requirements From the description of the VP and VSE received from the scientist and the administrator, our management tool is able to "merge" them and check their validity. This can be done thanks to the formalism introduced previously in Section 3.

4.3 Execution Module: Instantiation of VPs and VSEs

The Execution Module is in charge of creating the construction plan in order to provide the requested instantiation of the VP/VSE. Three steps can be enlightened: (1) the Execution Module checks if the VSE is already available on some resources, (2) if it is not the case, the Execution Module tries to instantiate the VP/VSE with the tools dedicated to software and resource management, and (3) the Execution Module allocates, deploys and configures the resources in order to set up instantiations of the VP/VSE.

Figure 8 presents an instantiation of VP and VSE on physical resources.

4.3.1 Environment Availability

The execution module is in charge of instantiate the VP and the VSE according to the description provided by the conciliator module.

First of all, the manager tries to find the required instantiation of VP/VSE with the Discovery Module.

- If the execution environment is already available on some resources, the Execution Module checks if there is no conflict between the obtained execution environment/resources and the users/administrators requirements.
  - If there is no conflict, the Execution Module tries to allocate and configure these resources.
- In case of conflict, or if no good instantiation of VP/VSE is available, the manager tries to instantiate them in order to answer the request.

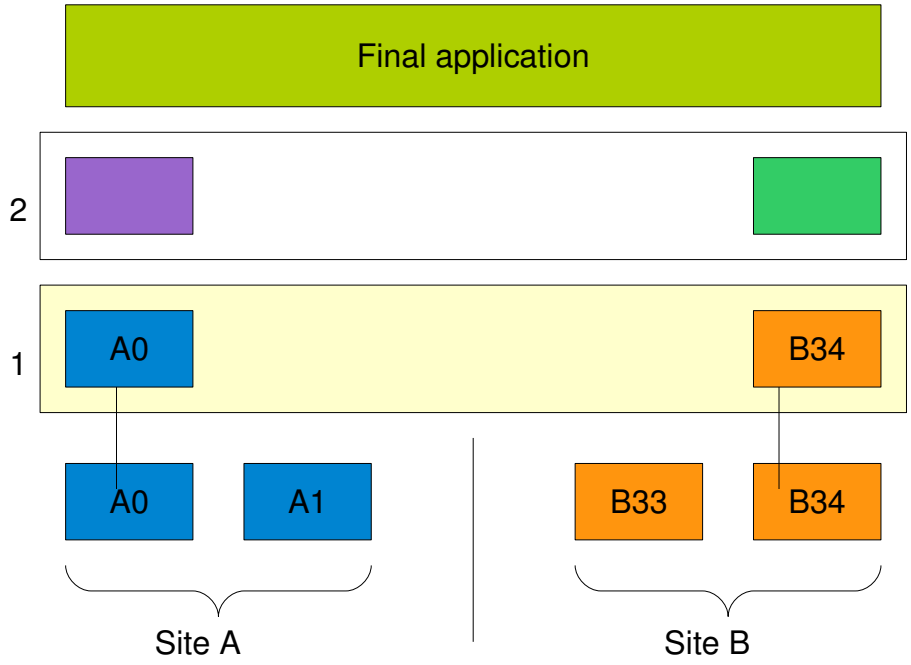
4.3.2 Concept of Plan: How to Instantiate VPs and VSEs

The Execution Module plans the action required to configure the required software on the resources. For that, the Execution Module asks to the Toolbox Module both the list of software and resource requirementse. Both lists are check for conflicts with the users and the administrators' requirements.

If everything is right, the Execution Module asks to the Discovery Module for finding some resources corresponding to the needs.

When receiving this list, the Execution Module is able, thanks to, the list of software needed, the list of resources available, the list of tools needed, to build a plan. For instance:

1. Allocate resources,



A0 and A1 are resources of the site A with architecture A. B33 and B34 are resources of the site B with architecture B. From the scientist point of view (layer 1), A0 and B34 are resources provided by the same site: the Virtual Platform, hiding the distributed complexity. In addition, the VSEs are build in order to feet the architecture A for the site A and B for the site B with all required packages, hiding the heterogeneity complexity (layer 2). As a result, the application can be run in a totally transparent way.

Figure 8: VP/VSE instantiation

2. Instantiate the VP:
   (a) Use "*tool1*" for the deployment of "*operating system*",
   (b) Use "*tool2*" for the installation of "*virtualization system*",
   (c) Use "*tool3*" to setup a VPN, ...
3. Instantiate the VSE:
   (a) Use "*tool4*" for the installation of "*libraries*",
   (b) Use "*tool3*" for the installation of the application, ...

4.4 Discovery, Allocation and Configuration of the Resources

The plan is given to the Discovery , Allocation and Configuration Module in charge of executing the plan, *i.e.,* instantiate the VP and the VSE. In case of error (*e.g.,* resource not available at the time of the allocation) the Execution Module is responsible for modifying the plan and for making a new submission.

4.5 Toolbox Module

The Toolbox Module manages the collection of tools for the instantiation of VPs and VSEs.

There are two types of tools: (1) the tools that manages the software repositories (VSE configuration), and (2) the tools that manages and configures the compute resources (VP configuration).

4.5.1 Tools to Instantiate a VSE

In order to provide an efficient system for software management, we designed our solution based on the existing concept of repository. Applications software is accessible from a well-defined source, can be downloaded, installed, and configured on the compute resources using dedicated tools. For instance, on Debian Systems, there are public Debian repositories to install and configure software on the target compute resources, using dedicated tools like *apt-get*.

For a specific software required by the user, the Software Management Toolbox is able to provide to the Execution Module a list of all the needed software (with all the dependencies and the required compute resource architecture).

4.5.2 Tools to Instantiate a VP

As done for the Software Management Toolbox, we use accessible repositories for the Resource Management Toolbox too. The repositories contain the list of available tools in order to provide resource partitioning, resource aggregation, and resource abstraction. In that way, tools for flexibility are accessible and can be installed, in order to set up and configure compute resources.

5 Current Implementation

A prototype is currently under development, merging and extending existing tools: we based our developments on software such as the OSCAR system management tool [17] and its extension OSCAR-V [27], the Saline virtual machine (VM) manager [5], and the Aladdin/G5K [4] grid system management tools (we are actively involved in all these projects).

5.1 Description and Conciliation Modules

Currently, VSEs are defined via XML files and allows one to describe the needed set of software packages (using package sets). The XML file is validate by OSCAR in two phases: (i) the basic validation of the XML file using standard XML tools, and (ii) the validation of the list of packages from the package set.

5.2 Toolbox Module

5.2.1 Tools to instantiate a VSE

OSCAR provides a tool for the management of OSCAR packages repositories: the OSCAR Repository Manager (ORM) and the OSCAR Package Manager (PackMan). Currently OSCAR packages cover standard Linux distribution such as CentOS, Fedora, Debian. OSCAR is able to create "golden images" containing all the necessary execution environment for a given distribution, based on a VSE definition.

5.2.2 Tools to instantiate a VP

Saline allows one to deploy virtual clusters, perform their network configuration, and manage them at grid level. Practically, Saline makes efficient periodical snapshots of the virtual cluster and can, if needed, restart the cluster on another site of the grid from the latest snapshot.

5.3 Execution Module

At each site, OSCAR-V actually receives the configuration information from the driver and ultimately deploys the environment needed for the execution of the application. By leveraging OSCAR-V, this environment can be based on a number of system configurations, to include: standard disk-full & disk-less system configurations and physical & virtual machines.

5.4 Discovery, Allocation and Configuration Module

Currently, the service discovery relies on a existing batch scheduler. This is a limitation of our system and we are working on other solutions like UDDI in order to find and use resources.

6 Use Case

To illustrate the capabilities of the proposed solution, we present a use case as example. A scientist wants to execute its application (myApplication) with

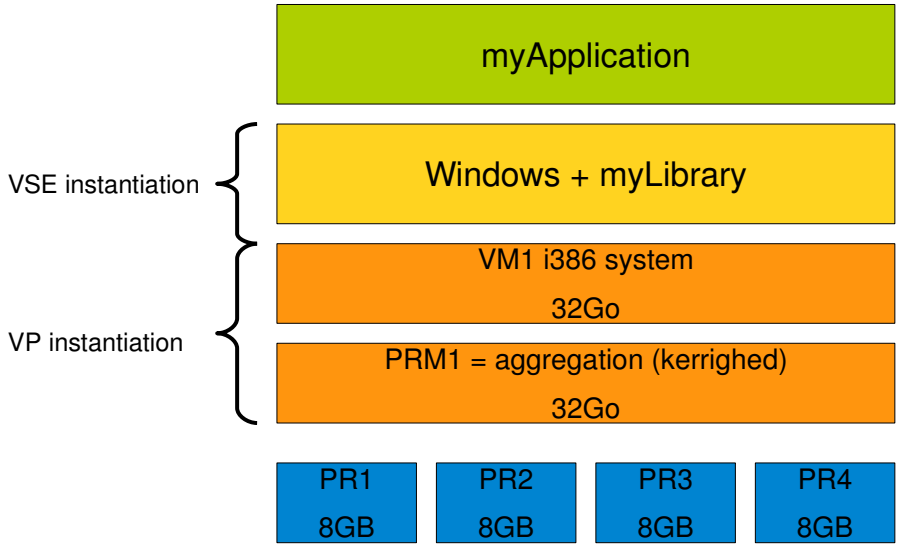


Figure 9: Use Case

a dedicated library (myLibrary) only available on Microsoft Windows systems, and need at least 32 GB of memory. The scientist has the habit of running its application on a dedicated Windows machine with 32 GB of memory.

However, if for some reasons, this machine is not available and, the only available machines are 4 Debian machine with 8 GB of memory, the scientist is confronted to three choices: (i) she or he waits for a similar Windows machine with 32 GB of memory to be available, (ii) she or he tries to modify the application to adapt it to the available environment, and (iii) she or he use a intelligent deployment tool, like the one we present in this paper, in order to take benefit of whatever HPC system available without modifications of the application. We assumethe scientist takes the choice (iii). In that case, our system should deploy an adequate instantiation of the VP, *i.e.,* an aggregation tool on the 4 physical resources (like the Kerrighed SSI OS). This provides the illusion to have one Linux SMP node with 32 GB of memory. Then the system should deploy a partitioning tool, like a VM in order to install and provide a Microsoft Windows environment. Now our tool can deploy the instantiation of the VSE inside the VMs. Finally, the scientist application can be deployed (see Figure9).

7 Conclusion and Future Work

In this document, we present the architecture of a new tool for the management of clusters, MPPs, grids, and clouds. The key of the architecture is to include adequate abstractions, assuming a computing system is composed of the following layers: *physical resources, physical resources manager, execution environment, and application.*

We propose a refinement of the Goldberg theory in order to be able to give more malleability in terms of management to the resources and the environment

execution. We also propose the concepts of Virtual Platform and Virtual System Environment in order to abstract and differentiate the resource management and the execution environment from the application. In addition, the abstraction we make allow one to use many already existing software.

We propose the design and present the current implementation of our management tool.

From the resource point of view, one of the major benefits is that our management tool does not only provide partitioning in order to manage the resources. It allows the "composition" of several tools for resource management in order to completely abstract the resources to the scientists.

From the execution environment point of view, one of the major benefits of the proposed architecture is to adapt the system environment to user needs without compromising the control capabilities of system administrators. This enables scientists to easily describe their application's needs in terms of software and the hardware resources and then the system will automatically create the correct environment for the target systems. For that, the proposed architecture includes abstractions for standard resource, users and application management software.

Ultimately, the proposed architecture provides powerful tools that could be reused in many different contexts, guaranteeing stability and compatibility which ultimately should ease the scientists' life.

Currently, we continue to work on the implementation of our prototype. More precisely, we are working on the integration of OSCAR and Saline. In addition, we are working on the definition of the VP description file.

We believe "*flexibility*" is a key concept for abstracting the complexity of HPC systems away from scientists so they can foxus on the science and not technical details associated to the execution of their applications on top of different platforms. This is why it is necessary to (i) dissociate the exection environment from the resources, and (ii) dissociate the view of the resources from the physical resources. Furthermore, the concept of flexibility should be of importance for the emergence of the "Everything as a Service" concept, in which everything is considered to be a service and can be configured, and re-configured "on-the-fly".

References

[1] Sun Grid Engine Welcome Page: http://gridengine.sunsource.net/. 2

[2] Torque Resource Manager Welcome Page: http://www.clusterresources.com/products/torque-resource-manager.php. 2

[3] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Andreas Savva. Job Submission Description Language (JSDL) Specification, Version 1.0, November 2005. 2

[4] Raphael Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frederic Desprez, Emmanuel Jeannot, Yvon Jégou, Stéphane Lantéri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, November 2006. 2, 5

[5] Jérôme Gallard, Adrien Lèbre, and Christine 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 – To appear*, Pisa Italie, 2010. 5

[6] Jérôme Gallard, Adrien Lèbre, Geoffroy Vallée, Christine Morin, Pascal Gallard, and Stephen Scott. Refinement Proposal of the Goldberg's Theory. In *ICA3PP'09: International Conference on Algorithms and Architectures for Parallel Processing*, pages 853–865, Tapei Taïwan, Province De Chine, 2009. 3.3

[7] Jérôme Gallard, Geoffroy Vallée, Adrien Lèbre, Christine Morin, Pascal Gallard, and Stephen Scott. Complementarity between Virtualization and Single System Image Technologies. In *Euro-Par 2008 Workshops - Parallel Processing: VHPC 2008, UNICORE 2008, HPPC 2008, SGS 2008, PROPER 2008, ROIA 2008, and DPA 2008*, Las Palmas de Gran Canaria Espagne, 2009. 3.3

[8] David Garlan, Robert T. Monroe, and David Wile. Acme: architectural description of component-based systems. pages 47–67, 2000. 2

[9] R. P. Goldberg. Virtual machines: semantics and examples. Proceedings IEEE International Computer Society, Conference Boston Massachusetts, 1971. 3.2

[10] R. P. Goldberg. Architecture of virtual machines. AFIPS National Computer Conference, July 1973. 3.2

[11] Paul H Hargrove and Jason C Duell. Berkeley lab checkpoint/restart (BLCR) for Linux clusters. *Journal of Physics: Conference Series*, 46:494–499, 2006. 2

[12] Kate Keahey, Ian Foster, Tim Freeman, Xuehai Zhang, and Daniel Galron. Virtual Workspaces in the Grid. In *11th International Euro-Par Conference, Lisbon, Portugal*, 2005. 2

[13] Kate Keahey and Tim Freeman. Contextualization: Providing One-Click Virtual Clusters. In *4th International Conference on e-Science*, Indianapolis, Indiana, USA, 2008. 2

[14] H. Andres Lagar-Cavilla, Joseph Whitney, Adin Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and M. Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proceedings of the 3rd European Conference on Computer Systems (Eurosys)*, Nuremberg, Germany, April 2009. 2

[15] J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. pages 73–82, 1993. 2

[16] Martin W. Margo, Kenneth Yoshimoto, Patricia Kovatch, and Phil Andrews. Impact of Reservations on Production Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 116–131. 13th Worshop on Job Scheduling Strategies for Parallel Processing, 2007. 2

[17] John Mugler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, Jeffrey M. Squyres, Benoît des Ligneris, Francis Giraldeau, and Chokchai Leangsuksun. OSCAR Clusters. In *Proceedings of the* 5^{th} *Annual Ottawa Linux Symposium (OLS'03)*, Ottawa, Canada, July 23-26, 2003. 5

[18] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 2009. 2

[19] James S. Plank, Micah Beck, Gerry Kingsley, and Kai Li. Libckpt: Transparent Checkpointing Under Unix. In *Proceedings of the USENIX 1995 Technical Conference (TCON)*, pages 213–223, Berkeley, CA, USA, 1995. 2

[20] Gerald J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. July 1974. 3.2

[21] Ivan Rodero, Francesc Guim, Julita Corbalán, and Jesús Labarta. How the JSDL can Exploit the Parallelism? Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006. 2

[22] Joseph F. Ruscio, Michael A. Heffner, and Srinidhi Varadarajan. DejaVu: Transparent User-Level Checkpointing, Migration and Recovery for Distributed Systems. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06)*, page 158, New York, NY, USA, 2006. ACM. 2

[23] Edi Shmueli and Dror G. Feitelson. Backfilling with Lookahead to Optimize the Performance of Parallel Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 228–251, 2003. 2

[24] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, 2005. 2

[25] R. P. Goldberg U. O. Gagliardi. Virtualizable architectures. Proceedings ACM AICA International Computing Symposium Venice Italy, 1972. 3.2

[26] Geoffroy Vallée, Thomas Naughton, Hong Ong, Anand Tikotekar, Christian Engelmann, Wesley Bland, Ferrol Aderholdt, and Stephen L. Scott. Virtual System Environments. In *Systems and Virtualization Management. Standards and New Technologies*, volume 18 of *Communications in Computer and Information Science*, pages 72–83. Springer Berlin Heidelberg, October 21-22, 2008. 3.4, 4.1

[27] Geoffroy Vallée, Thomas Naughton, and Stephen L. Scott. System management software for virtual environments. In *CF '07: Proceedings of the 4th international conference on Computing frontiers*, pages 153–160, New York, NY, USA, May 7-9, 2007. ACM. 5

Contents

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399