Virtual System Environments

Geoffroy Vallée and Thomas Naughton and Hong Ong and Anand Tikotekar and Christian Engelmann and Wesley Bland and Ferrol Aderholdt and Stephen L. Scott [1]

Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA
{valleegr, naughtont, hongong, tikotekaraa, engelmannc, blandwb, aderholtwf, scottsl}@ornl.gov
http://www.ornl.gov

Abstract. Distributed and parallel systems are typically managed with "static" settings: the operating system (OS) and the runtime environment (RTE) are specified at a given time and cannot be changed to fit an application's needs. This means that every time application developers want to use their application on a new execution platform, the application has to be *ported* to this new environment, which may be expensive in terms of application modifications and developer time. However, the science resides in the applications and not in the OS or the RTE. Therefore, it should be beneficial to *adapt the OS and the RTE to the application instead of adapting the applications to the OS and the RTE*.
This document presents the concept of Virtual System Environments (VSE), which enables application developers to specify and create a virtual environment that properly fits their application's needs. For that four challenges have to be addressed: (i) definition of the VSE itself by the application developers, (ii) deployment of the VSE, (iii) system administration for the platform, and (iv) protection of the platform from the running VSE. We therefore present an integrated tool for the definition and deployment of VSEs on top of traditional and virtual (*i.e.*, using system-level virtualization) execution platforms. This tool provides the capability to choose the degree of *delegation* for system administration tasks and the degree of protection from the application (*e.g.*, using virtual machines).
To summarize, the VSE concept enables the customization of the OS/RTE used for the execution of application by users without compromising local system administration rules and execution platform protection constraints.

1 Introduction

The architecture for modern distributed and parallel execution platforms differ from single head node/multiple compute nodes Beowulf clusters to distributed Grids and large-scale system with specialized nodes (*e.g.*, I/O nodes). Several tools are available for the management of such platforms [6, 8, 10, 11].

Furthermore, as different system solutions emerge on top of traditional computing platforms, such as system-level virtualization, system management tools have also been extended [15]. While these enhancements allow for the deployment of environments on

[1] ORNL's research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

new platforms like virtual machines, they are lacking in terms of customizability – specifically from the perspective of application developers.

Because of that, with current system management solutions, application developers do not gain any flexibility. Applications still have to be *ported* every time developers want to use a new execution platform. However, the science resides in the applications, not in the system software for the execution platforms. Based on this contrast, it is critical to provide a solution that allows application developers to customize their execution environment that will then be deployed on top of the execution platforms. In other words, *the operating system (OS) and the runtime environment (RTE) have to be adapted to the application and not the application adapted to the OS and the RTE of a specific platform*.

To address this issue we propose the concept of a *virtual system environment* (VSE), which decomposes these challenges into two different aspects: (i) the definition of the environment needed to run the application, both according to application developers and system administrators perspective – this high-level description is actually very agnostic about the system configuration of the target system for application execution, and (ii) the deployment of a defined VSE on a target environment. We currently support disk-full/disk-less and physical/virtual systems; and also system partitioning (*e.g.,* I/O nodes versus login nodes versus compute nodes).

The remainder of this paper is organized as follows: Section 2 presents how a VSE can be defined by both application developers and system administrators. Section 3 presents a tool for the deployment of VSEs on top of various system configurations (*i.e.,* physical/virtual, disk-less/disk-full). Section 4 presents VSE benefits for system administration. Section 5 presents the effect of the VSE concept on system protection. Section 7 concludes.

2 Virtual System Environment Definition

An application is typically designed to be executed with a specific version of an OS and RTE. For instance, an MPI application can be designed to run on top of RedHat Enterprise Linux 4.0 with LAM/MPI 7.1.3. This kind of information is decided by developers in order to simplify development. It also means that every time another environment has to be used, most likely the application will have to be modified, ported.

On the other hand, computing centers today provide different execution platforms: clusters, shared memory systems, or even large-scale high-performance systems such as Cray XT or IBM BlueGene systems. Each of these systems typically provide a different execution environment and applications have to be "adapted" to each of them. However, the science resides in the applications and therefore application developers should not have to deal with such porting issues, and should be able to focus on the science.

It is important to decouple the definition of the application's needs in term of RTE and what components system administrators want to have in each environment used by applications.

System-level virtualization provides a first step in that direction, decoupling the environment used for the execution of the applications and the environment used on top of the hardware (virtual machines versus host OS). For instance, it is possible to

create a *virtual appliance*, *i.e.,* a specialized virtual machine for the execution of a given application. However, the concept of appliance does not ease the definition of the application environment; the system within the VM still need to be more or less manually installed. Furthermore, because of the lack of meta-data defining what the application environment is, it is not possible to deploy an existing virtual appliance on top of a standard system (*i.e.,* disk-less or disk-full system).

Additionally, as discussed in Section 6, system management tools have been extended to support virtual environments but suffer from some significant limitation. OSCAR-V [15] is such a tool, managing virtual machines and creating images for virtual machines with a minimal system footprint (only needed software is included into the image), but it is not possible for administrators and users to easily express their definition of execution environments (OSCAR-V recognizes only Beowulf clusters [14], and not Grids or large-scale systems) and it is still difficult for users & application developers to define their execution environment [1].

The VSE concept aims to address these challenges and has been implemented as an extension of the OSCAR-V prototype: (i) the VSE fits application needs, no unnecessary system footprint in included; (ii) the OS type & version and the RTE are chosen by application developers and not by system administrators; (iii) system administrators can check the VSE before deployment; (iv) application developers can define their VSE off-line from the execution platform; and (v) the VSE can be deployed automatically by system administrators. Because the VSE implementation is actually a non-intrusive extension of OSCAR-V, performance for VSE creation and deployment is actually similar to OSCAR-V performance. Thus, we do not present performance results in this document, only the VSE architecture and implementation is described in details. Application developers define their RTE needs using a high-level language based on XML, which describes a set of software packages (*Package Sets*). 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. More details are presented in Section 2.2. In mathematical terms, our notion of sets follows the Zermelo-Fraenkel set theory, with the axiom of choice (ZFC). 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 provide a very flexible method for the definition of complex VSEs.

2.1 Package Sets Definition

Package Set Combination It is possible to combine package sets together:

$$PackageSet_A \cup PackageSet_B$$

This enables the combination of VSE definitions from application developers and system administrators (see Figure 1).

[1] This may lead to conflicts between applications' needs and system administrators' needs; we do not provide an automatic solution to manage these conflicts since most of the time there are policy issues.

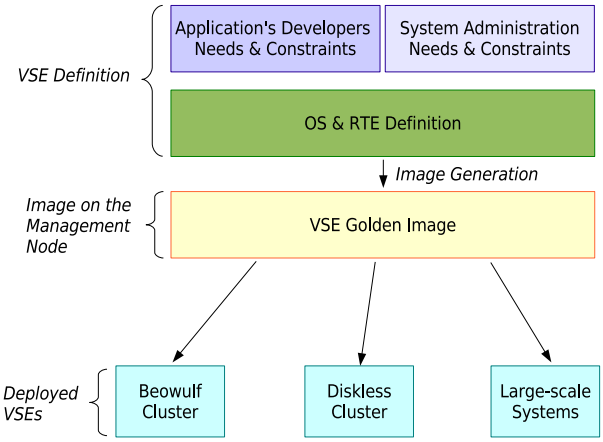

Fig. 1. VSE Definition and Management

For instance, if system administrators, based on local policy, want to include the Moab software [9] in all VSEs because it is the chosen workload manager used by the computing center, they can create a VSE definition that will be combined with an application's developers definition. The resulting specification incorporates the constraints from both the application and system administrators.

Package Set Intersection It is also possible to define the intersection of package sets:

$$PackageSet_A \cap PackageSet_B$$

.

The intersection operation is more suitable for advanced capabilities rather than the strict definition of a new VSE. For instance, the intersection operation can be used to identify common software components between several VSEs. That can be used later on by system administrators in order to identify current needs of application developers in term of settings of the execution environment, and therefore try to address more efficiently present needs and anticipate future needs.

Package Set Validation The package set mechanisms also include basic validation capabilities in order to ensure that package sets can be correctly combined. This validation tool is based on a versioning mechanisms (comparison of software version), and a dependency mechanism (set A depends on set B but conflicts with set C).

If users combine several package sets together the system also checks that the Linux distribution from the different package sets are the same.

Versioning Users can specify the version of each package within a package set. This allows for fine grain software management. For instance, application developers can specify that their application needs a specific version of a library. We provide standard operators to deal with versioning: *equal (eq)*, *superior to (gt)*, *inferior to (lt)*, *superior or equal to (gte)*, and *inferior or equal to (lte)*.

2.2 Package Sets Usage

Package sets define a VSE and are used to create a "golden image" which is agnostic of the target platform execution configuration. The current VSE implementation relies on OSCAR [10], a system management software.

The package sets implementation actually directly relies on *OSCAR Packages* (OPKGs) which allow one to define a software package for software installation in distributed or parallel systems, including information such as a list of binary packages, configuration scripts and versioning information. It typically extends the standard notion of binary packages, adding information about what has to be done to have the software setup at the global level of the distributed or parallel systems. Note that we assume application developers provide their application via an OSCAR package.[2]

The current implementation supports the definition of single package sets and the combination of package sets. The package set intersection operation has not yet been implemented.

Package Set Analysis In order to ease system administration tasks and to track modifications, information about package sets are stored in a database, the OSCAR Database (ODA).

The first step for the creation of a VSE is therefore the parsing of the VSE's XML file, its validation via XML tools and the update of the OSCAR database. We will see later that information in the database is used to update a basic image, which ultimately results in a golden image that matches the VSE definition.

The validation is composed of two phases: (i) the basic validation of the XML file using standard XML tools, (ii) the validation of the list of OPKGs from the package set. OSCAR provides a tool (OSCAR Package Downloader - OPD) for managing OPKG repositories, which can be used to download OPKGs. OPD2, especially developed for the VSE support, allows us to get the list of all the available OPKGs, for all supported Linux distributions. OPD2 also saves information about OPKGs into the database. Based on this list, it is possible to validate package sets (*e.g.*, checking if OPKGs are available).

Creation of a Basic Golden Image Based on package sets, a "golden image" [2, 10] can be created and used for the actual deployment of a given VSE. For that, we (i) analyze the package set(s), (ii) create a basic golden image for the target Linux distribution, and (iii) install the different OPKGs based on the package set definition.

The creation of basic golden image relies on the OSCAR version of the SystemInstaller software [2]. The creation of the basic image is based on the Linux distribution and the architecture specified in the package set(s). During the image creation, the OSCAR database (ODA) is updated in order to initialize information about the new image.

Once the basic image is created, and based on information about the package set from ODA, it is possible to finalize the golden image installing the OPKGs associated to the package set.

[2] OSCAR packages are based on binary packages (*e.g.*, RPMs or Debian packages), the creation of new OSCAR packages is fairly simple if application developers already provide binary packages for their application.

OSCAR did not have a stand alone tool for OPKG management, initially all OPKGs were installed directly into the image, using SystemInstaller, without using information in ODA. We therefore implemented the *OSCAR Package Manager* (OPM) tool. This tool queries the database to know the exact status of images, OPKGs and nodes. Based on this information, OPM installs OPKGs into images but also on remote nodes if nodes have already been deployed. The image is then ready to be deployed and the database up-to-date for management purpose.

3 Virtual System Environment Deployment

We target three different platform architectures: (i) Beowulf clusters, (ii) disk-less clusters, and (iii) large-scale systems (*i.e.*, platforms with specialized nodes).

We saw in Section 2 that it is possible to have an XML file which describes the VSE that has to be deployed and to create the associated golden image. Because we do not want to have to recreate the image every time we deploy it (for instance to have image persistence), we take care to separate tools for image creation and mechanisms to deploy them. In other words, the VSE's XML description is used to generate a "golden image" on the management node. Then, this golden image is "adapted" to fit the platform architecture. This adaptation is based on the description of the target platform.

3.1 Machine Sets

In order to express the topology of the target system, we introduced the notion of node sets (also called node groups). This concept, like the package sets concept, follows the Zermelo-Fraenkel set theory, with the axiom of choice (ZFC). This includes the support of union and intersection operations on machine sets:

$$MachineSet_A \cup MachineSet_B$$

$$MachineSet_A \cap MachineSet_B$$

A key characteristic of node sets is the need to express dependencies between the machines in the set. To address this issue, we assume that all relationships between nodes can be expressed as server/client dependencies, *i.e.*, more precisely as a one-to-one dependency (set A is dependent upon set B). For instance, for I/O nodes where three types of nodes are used (meta-data server, data storage server and compute nodes), two different node sets can be defined: (i) a set for the meta-data server and the data storage server where the meta-data implements the notion of server for the machine set, the data storage server being the client; and (ii) a set for the data server (both meta-data and data storage server) and the compute nodes. To define complex systems (*e.g.*, a large-scale system composed of login nodes, compute nodes, and I/O nodes), different machine sets can be defined, one for each type of nodes.

3.2 Image Deployment

Figure 2 shows the overall architecture for VSE deployment mechanism for the support of Beowulf clusters, disk-less systems and large-scale systems. Note that the figure includes the relationship with the mechanism for the creation of the golden image.

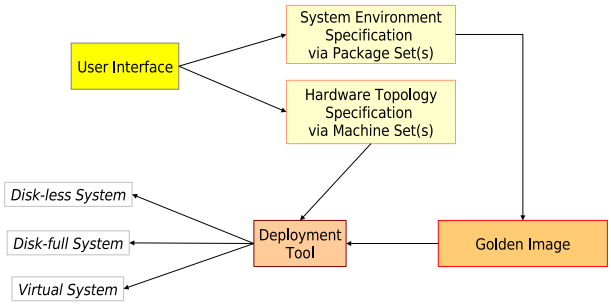

Fig. 2. Overall Architecture for the Deployment of VSEs

Beowulf Clusters Beowulf clusters are still the standard architecture for clustering: a headnode provides all clustering services and compute nodes do the application computation using services from the headnode. This is a standard client/server architecture.

To describe this architecture only two node sets are needed: (i) the headnode, and (ii) the compute nodes.

Disk-less Clusters Disk-less clusters may be deployed in many different ways. Currently, we use the standard NFS-ROOT [3] or RAMFS solutions, which is sufficient for small to medium sized clusters (we assume that for large-scale systems, the standard design for such systems is used).

In this case, the system may be categorized into two groups, like for Beowulf clusters: (i) the headnode, or server; (ii) the compute nodes.

The difference with Beowulf clusters is that the image is not "deployed": the image is copied on the headnode, making the difference between shared data and modified data. Then compute nodes are booted and use their own image.

OSCAR did not initially support disk-less clusters. We developed an extension of OSCAR for the support of NFS-ROOT and RAMFS based disk-less support. This support is based on the tuning of images on the headnode. A golden image is divided into two parts: (i) a *shared image* for the part of the file system that can be shared between nodes (read-only), and (ii) a *private image* for the part of the file that needs to be in read/write mode. It is also possible to fall back to a disk-full solution (logically merging the two images for deployment).

Large-Scale Systems For large-scale systems, the situation is different because this kind of architecture is no longer based on the idea of one single server and many compute nodes. Typically, for this kind of architecture, nodes are grouped into different sets: compute nodes, "service nodes" (*e.g.*, for the parallel file system), and "login nodes".

It is possible to describe a server/client dependency relationship between the different nodes involved in a single service (for instance the I/O subsystem), combining node groups together. For instance, PVFS [12] has three kinds of nodes in order to implement the parallel file system: meta-data server, storage nodes and clients. It is possible to say that the meta-data server has a server/client dependency and then combine these two

into a single node group and create a dependency between this group and the compute nodes.

The current implementation allows one to describe different node groups and to combine them together. Based on this mechanism and the assignment of one specific image to a group of nodes, it is possible to deploy complex large-scale systems.

Virtual Systems Another solution for the deployment of VSEs is the usage of virtual machines. In this context, the VSE can be instantiated via virtual appliances that can be viewed as a minimal system configuration specialized for the execution of a given application. Thus, VSEs can be considered as a specification tool for virtual appliances.

We previously extended OSCAR, creating OSCAR-V [15], to support system-level virtualization. One of the benefits of OSCAR-V is the ability to support several system-level virtualization systems via the V2M abstraction layer. This allows one to switch between virtualization solutions without re-deploying virtual machines.

Combining the concept of VSEs and features from OSCAR-V, users can take full advantage of virtualization, simplifying the management of virtual systems and improving the customizability of execution environments.

4 System Management

The administration of computing systems must strike a balance between the required system aspects and those which are strictly end-user specific. The ultimate goal being to support users and their computational needs. However, the responsibility of maintaining the system typically does *not* lie in the hands of the individual(s) most familiar with the applications using the resources.

VSEs offer an interesting means by which system administration tasks can be delegated to the end-user who is most aware of the application's needs. The extent to which these system administration tasks are delegated may differ based on the approach used for implementing the VSE, *e.g.*, node partitions, disk-less nodes, virtual machines. The extent of delegation must be commensurate with the selected protection scheme. For example, the VSE might be a common system image that users customize and deploy on a set of disk-less nodes or could be entirely user generated based upon virtual machine platform specifications. In either case, the proper level of privileges is matched with the degree of customization by the system administrator.

This provides a basis to use VSEs to improve user control for specialization, which can be used for systems research testbeds or to simply provide a consistent platform environment for scientists. A VSE also provides a good basis to empower user expertise, which is commonly found on large scientific systems. In many situations these users may require older libraries/compilers or even operating system kernels, which are easily supported through the use of a VSE.

As presented in Section 2, both the system administrators and the users can define their own package set and machine sets. These sets are then merged to describe both the system environment and the hardware partitioning that fits both system administrators and users needs. In summary, the VSE concept allows more flexible management of the environment used by applications without compromising the local system administration policies.

5 Protection

Because the VSE concept enables the customization of different types of systems (*e.g.*, disk-full & disk-less Beowulf clusters, virtualized systems), we provide a sliding scale of protection. In other words, based on the system configuration described via package sets and machine sets, it is possible to increase or decrease the degree of protection for both the user and system administrator: (i) system administrators can protect the execution platform from malicious applications or trust application and give them direct access to the hardware for performance purpose; and (ii) the application developers can choose to run the application directly on top of the bare hardware, with the risk to have to modify the application, or to run in a virtual environment in order to ensure a similar execution environment on all the VSE enabled platforms.

The protection mechanism is typically tied to the degree of customization supported by the system, *i.e.*, the more you can change/customize the more likely you may want to dial the protection level up, ultimately using virtual environments for maximum isolation (see Table 5).

System Type	Protection Level	System Administration Type
Disk-full Beowulf Cluster	Low	Central system administration
Disk-less Beowulf Cluster	Medium	Central system administration
Virtual System	High	Delegation possible

Table 1. System Characterization According to Protection and System Administration Delegation Capabilities

6 Related Work

The HARNESS project [5] studies the launch of a virtual environment at job start, this virtual environment being installed by the runtime environment for a particular application. This capability enables the deployment of virtual RTEs that fit application's needs. However, this study suffers of limitations in term of flexibility for the creation of a complete virtual RTE, especially in term of "virtual hardware" (it is not possible to support solutions based on system-level solutions) and HARNESS does not provide integrated tools for system deployment.

The *Modules* system provide environment customization at the level of a user's command interpreter (shell) [4]. The Modules system is responsible for managing the differences between command shells, *e.g.*, bash, csh. A system administrator provides the available software and configuration setting via a Tcl file that may be loaded at shell invocation. The users customize their execution environment by loading the appropriate "modules", *e.g.*, `module load mpi/lam-7.0.6`. These command can be made persistent using a higher level tool like Env-Switcher [10]. While the Modules system is widely used and quite useful, it is limited to changes that can be made at

the command interpreter level. Therefore, alternate kernel versions or entirely different operating systems are not an option with this approach.

VMPlants [8] is a solution for the management of *virtual execution environments* in a grid context. A virtual execution environment is defined by a graph which allows users to customize their virtual execution that can be then deployed within virtual machines (using VMWare [16] or User Mode Linux [7, 1]). However, VMPlants assumes that a system already exists on each machine the user will use. VMPlants does not provide any solution for the management of this system but also does not provide tools and methods for the interaction between the site system administrator and the application's users. It is therefore not possible to enforce the use of specific software within the virtual machine (for instance the use of a checkpoint/restart solution); it is not possible to check if the virtual environment defined by the application users is compliant with local system usage policies, and the management flexibility offered to users is not available to system administrators. Finally, it is not possible to deploy various type of execution platforms based on VMPlants, the use of virtual machines is mandatory.

The *Collective* project [13] is based on the idea of *virtual appliances*, which are application specific bundles that an author (vendor) maintains and end-users use with limited or no administration responsibilities. They employ a virtual machine monitor (VMM) to provide a trusted computing base and effectively a hardware abstraction layer (HAL) to ease appliance portability. An appliance is a specialized single purpose system, *e.g.*, word-processing-appliance, that a user may use but does not maintain (administration is done by the appliance author). This concept is similar to that of a VSE, but differs in scope and how composition is achieved. The VSE is primarily targeted at HPC environments, whereas the virtual appliances are focused on general purpose desktop environments. The appliances are not extended or changed by the user to build up their environment, instead a collection of separate appliances are used in concert (each administered independently). Both approaches use virtualization to assist with portability and enhance usability. However, the Collective assumes multiple appliances (virtual machines) could be run simultaneously where the VSE would typically encompass a single virtual machine.

7 Conclusion

This document presents the concept of a *Virtual System Environment* (VSE), which has been implemented via extensions and/or modifications of the OSCAR and OSCAR-V system management suites. A VSE decouples the definition of the execution environment from the actual deployment method. Users can therefore define application needs and constraints in a generic way. On their side, system administrators, depending on the degree of trust and local management policies, can deploy the VSE into various system types (*e.g.*, disk-less versus disk-full systems, physical versus virtual systems) on specific system partitions. Therefore, VSEs introduce a high degree of customization for application developers, end-users and system administrators. For that, we introduce the notion of *package sets* and *machine sets* which provide a flexible way to define the execution environment and the hardware topology, respectively.

To summarize, through the VSE concept, it is possible to revisit the traditional system administration rules without compromising the platform protection or increasing the number of administration tasks. In fact, the system protection can even be increased when using virtual environments, which can be done without a great deal of management effort (system-level virtualization is natively supported by OSCAR-V). System administration tasks can also be decreased, delegating some tasks to application developers.

This is especially useful for virtual systems: the system administrator can define what is the host OS, without paying attention to the system-level virtualization solution; and application developers can focus on the definition of the VSE that will be deployed into the virtual machines.

References

1. Joe Brockmeier. *The Definitive Guide to User Mode Linux*. APress, 2004.
2. Sean Dague. System Installation Suite Massive Installation for Linux. In *The 4^{th} Annual Ottawa Linux Symposium (OLS'02)*, Ottawa, Canada, June 26-29, 2002.
3. Hans de Goede. Root over nfs clients & server howto. http://www.clusterresources.com/pages/ products/moab-cluster-suite/workload-manager.php.
4. John L. Furlani and Peter W. Osel. Abstract Yourself With Modules. In *Proceedings of the 10th Large Installation Systems Administration Conference (LISA'96)*, pages 193–204, Chicago, IL, September 29 – October 4, 1996.
5. G. A. Geist, J. A. Kohl, S. L. Scott, and P. M. Papadopoulos. HARNESS: Adaptable virtual machine environment for heterogeneous clusters. *Parallel Processing Letters*, 9(2):253–273, 1999.
6. Yiannis Georgiou, Julien Leduc, Brice Videau, Johann Peyrard, and Olivier Richard. A tool for environment deployment in clusters and light grids. In *Second Workshop on System Management Tools for Large-Scale Parallel Systems (SMTPS'06)*, Rhodes Island, Greece, April 2006.
7. H. jorg, H. Oxer, H. Hoxer, K. Buchacker, and V. Sieh. Implementing a user mode linux with minimal changes from original kernel, 2002.
8. Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.
9. Moab workload manager. http://www.clusterresources.com/pages/ products/moab-cluster-suite/workload-manager.php.
10. John Mugler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, Jeffrey M. Squyres, Benot 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.
11. Philip M. Papadopoulos, Mason J. Katz, and Greg Bruno. Npaci rocks: tools and techniques for easily deploying manageable linux clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):707–725, 2003.
12. PVFS: Parallel virtual file system. Available at http://www. parl.clemson.edu/pvfs.
13. Constantine Sapuntzakis and Monica S. Lam. Virtual Appliances in the Collective: A Road to Hassle-free Computing. In *Proceedings of HotOS'03: 9th Workshop on Hot Topics in Operating Systems*. USENIX, 2003.

14. T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.
15. Geoffroy Vallée, Thomas Naughton, and Stephen L. Scott. System management software for virtual environments. In *Proceedings of ACM Conference on Computing Frontiers 2007*, Ischia, Italy, May 7-9, 2007.
16. VMware, Inc. http://www.vmware.com.