Journal of Computer and System Sciences 79 (2013) 201–215

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

TopCluster: A hybrid cluster model to support dynamic deployment in Grid

Gang Chen[a,b,c], Yongwei Wu[a,b,*], Jie Wu[d], Weimin Zheng[a,b]

[a] *Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNLIST), Tsinghua University, Beijing 100084, China*
[b] *Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China*
[c] *Computer Network and Information Center, Chinese Academy of Sciences, Beijing 100080, China*
[d] *Department of Computer and Information Sciences, Temple University, Philadelphia, PA 119122, United States*

A R T I C L E   I N F O

Article history:
Received 30 December 2010
Received in revised form 29 July 2011
Accepted 1 May 2012
Available online 14 May 2012

Keywords:
Heterogeneous (hybrid) systems
Distributed architectures
Distributed systems

A B S T R A C T

Cluster virtualization is a promising approach to construct customized execution environments for Grid users. However, Virtual-Machine Cluster (VCluster) comes with the cost of the overhead caused by virtual machines, which therefore degrades system performance. In this paper, we propose a novel hybrid cluster model, namely TopCluster. By exploiting a Hybrid Batch System (HyBS) mechanism, TopCluster can flexibly allocate both physical or virtualized resources via a unified front-end. To reduce the overhead caused by the dynamic deployment of virtual machines and applications, we also propose a value-based deployment strategy along with TopCluster. An XML-based Application Deployment Description Language (ADDL) is also newly proposed to describe properties, installation procedure, and configuration operations of applications. A prototype system has been implemented and a series of experiments was conducted to evaluate TopCluster by replaying a real world trace. Results show that TopCluster can averagely generate 24.6% and 46.8% higher system throughput, 26.3% and 110.6% higher resource utility, and 35.1% and 56.1% lower waiting time of jobs than two commonly applied cluster models in Grid: Traditional Physical Cluster (PCluster) and VCluster.

1. Introduction

In high performance computing, clusters are often used in two models: the Traditional Physical Cluster (PCluster) model and the Virtual-Machine Cluster (VCluster) model. The PCluster model has a cluster scheduler (e.g., OpenPBS [1]) deployed on the front node of the PCluster model and used to allocate physical cluster nodes for jobs directly submitted to the cluster scheduler by the users. As a cluster of clusters, Grid [2] exploits the PCluster model to integrate resources of clusters. It utilizes resource management services to directly submit jobs to cluster schedulers. For example, the Grid Resource Allocation Management (GRAM) of Globus Toolkits [3] supports various local resource managers (e.g., Condor [4], PBS [1], and LSF [5]). However, the hardware and software configurations of GRAM are not changeable. Therefore, the PCluster model is not flexible enough to meet user requirements (e.g., customized operating system configuration). The VCluster model has Virtual Machines (VMs) constructed on its physical nodes, which are therefore used to create virtual clusters. Users submit jobs to these virtual clusters instead of the physical nodes. In the VCluster model, each virtual machine is usually dedicated

[*] Corresponding author at: Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNLIST), Tsinghua University, Beijing 100084, China.
E-mail addresses: c-g05@mails.tsinghua.edu.cn (G. Chen), wuyw@tsinghua.edu.cn (Y. Wu), jiewu@temple.edu (J. Wu), ygw@tsinghua.edu.cn, zwm-dcs@tsinghua.edu.cn (W. Zheng).

0022-0000/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcss.2012.05.010

to a single user. Users are permitted to choose virtual machines with a preferred operating system configured based on the users' needs. So, in terms of customization, VCluster is better than PCluster; however, this benefit of VCluster is gained at the cost of the overhead caused by its virtual machines. The authors of [40] indicated that both the VMware and Xen hypervisors have certain overhead. For many benchmarks, the overhead varies from 10% to 40%. Therefore, the overhead caused by VMs is an issue that needs to be addressed.

To reduce the overhead inherited in the VCluster model, we propose a novel hybrid cluster model in this paper, denoted as TopCluster. As opposed to VCluster, TopCluster relies on the original cluster scheduler (e.g., OpenPBS [1]) deployed on a physical cluster to schedule jobs and manage all (physical and virtual) resources. We denote this mechanism as Hybrid Batch System (HyBS) in this paper. With HyBS, TopCluster can flexibly allocate physical or virtualized resources for users' jobs; if qualified physical resources exist for a job request, then these resources are directly allocated. Otherwise, a virtual cluster is dynamically deployed and allocated for the job. By doing so, the overhead caused by virtualization can be avoided due to the mechanism of direct manipulating physical resources whenever it is feasible; while virtualization (e.g., customization) can still be utilized and benefited from when necessary.

To construct a customized cluster in TopCluster, as same as VCluster, two levels of deployment are required: one (low level) is to efficiently create and deploy virtual machines on physical clusters, and the other (high level) is to dynamically deploy (i.e., install and configure) applications (i.e., services and software) to virtual clusters created by these virtual machines. The low level deployment has been well studied in many literatures (e.g., [6–9]). The high level deployment (i.e., dynamic deployment of applications) has not however received enough attention. A common solution is to package applications, libraries and operating systems as a single image. However, packaging is of limitations, especially in the following two situations:

(1) Packages of scientific applications may update frequently, because new features may be continually introduced to the applications and it is quite common to find bugs caused by, for example, mistakes of programming, absence of important functions, unacceptable low performance, after a release.
(2) Users may need applications with a specific version. On one hand, it is not always true that the mostly updated version is definitely better than its previous versions. On the other hand, the users may need to compare results of different versions of a same application.

Therefore, a package creator needs to maintain all versions of applications. If the applications are statically deployed in a package, the package creator will have to recreate the package frequently from time to time. It is not hard to imagine that the number of packages created for all the versions of the applications will be very large and increase rapidly. With the dynamic deployment of applications, the package creator only needs to create a naked image of an operating system, without any scientific applications installed on it. Applications can be dynamically deployed according to the demands of users. In this paper, we propose a cluster model to facilitate such a dynamic deployment of applications directly or indirectly demanded by users.

As discussed above, in order to fully take advantage of resource virtualization, applications should be dynamically deployed; however, doing so would unavoidably cause overhead from the following two aspects. Aspect 1: regarding virtual machines, the system has to create profile files, transfer image files, and boot these virtual machines. Aspect 2: in terms of applications, the system has to download, unpackage, uncompress, copy, compile, install, and configure them. In order to reduce deployment overhead, along with TopCluster, we also propose a value-based strategy for the dynamic deployment of virtual machines and applications. For Aspect 1, when there is no enough free space left for a new virtual machine, the value-based strategy selects already deployed virtual machines with less value to destroy in order to free space for the new virtual machine. For Aspect 2, when there is no enough free space left to install a new application, the strategy chooses already deployed applications being of less value to undeploy in order to free space for the new application.

There exist some mechanisms that might be adapted for dynamic deployment of applications in virtual clusters, but with various difficulties. Some of them (e.g., apt [10] and rpm [11]) are system-dependent; some (e.g., CDL [12]) are not professional for application deployment; some (ant [13], DistAnt [14], DeployWare [15] and tcl [16]) are not suitable for complex scenarios, such as efficiently describing, deploying, and maintaining a large number of applications in Grid.

Besides TopCluster and the value-based strategy for the dynamic deployment of virtual machines and applications, we also propose, in this paper, an Application Deployment Description Language (ADDL). ADDL is based on the Extensible Markup Language (XML) [17], and it defines a set of elements to describe properties, the installation procedure, and configuration operations for scientific applications. ADDL also is of the following important characteristics: system-independency, reusability, parameterization, localization, and parallelization.

A prototype system has been developed to verify our TopCluster model. A set of experiments was conducted by replaying a real workload trace. Experimental results show that TopCluster can more efficiently utilize resources than the other two models: PCluster and Vcluster.

The rest of the paper is organized as follows: In Section 2, we discuss the architecture of TopCluster and its major components. Section 3 presents the value-based deployment strategy. Section 4 presents the detailed design and definition of ADDL. The prototype system of TopCluster is described in Section 5. Section 6 presents the setting of experiments. Experimental results are discussed in Section 7. Related work is given in Section 8, and we draw a conclusion in Section 9.

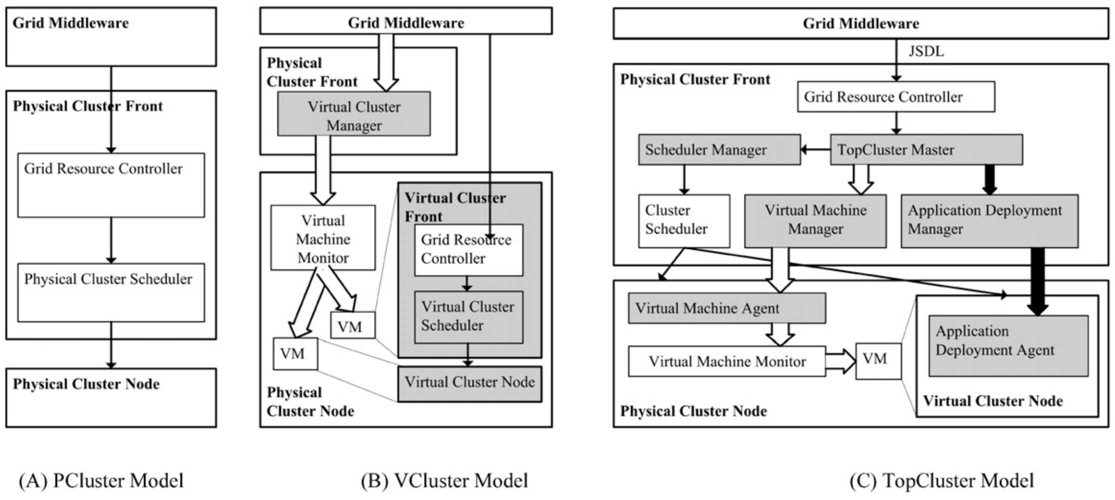

Fig. 1. Three cluster management models in Grid. The gray components are newly added to the current model, compared with the model on the left.

2. Architecture of TopCluster

Most existing Grids (e.g., OSG [18], TeraGrid [19], and ChinaGrid [20,22]) rely on clusters to perform actual computations; a Grid system integrates its clusters into its infrastructure and makes the infrastructure available for users, in which case, a resource management mechanism is usually required. In this section, we describe three different models/mechanisms for resource (cluster) management, one of which is our TopCluster, and the other two (PCluster and VCluster) are also introduced for comparison purpose. Their architectures are presented in Fig. 1.

2.1. PCluster model

Before virtualization technologies were used in Grids, clusters were integrated by deploying a *Grid resource controller* (e.g., GRAM of Globus Toolkits) on each of the front nodes of each physical cluster, as shown in Fig. 1(A). The *Grid Middleware* passes a user request to the *Grid Resource Controller*, which then submits the request to the *Physical Cluster Scheduler* (e.g., OpenPBS). The scheduler is responsible for allocating resources for the request, starting and monitoring user requested jobs.

However, as previously discussed in Section 1, the PCluster model is not flexible enough to meet user requirements; users in Grid may need resources with some specific operating systems, which may not be supported by physical resources. In this case, a user's job may be rejected or has to wait until resources with proper operating systems are available.

2.2. VCluster model

Because of many advantages of virtualization technologies, virtual clusters tend to be a trend in Grids. The VCluster model is one of the models which takes advantage of these technologies and is based on virtual clusters. As shown in Fig. 1(B), to compare with PCluster, several new components (highlighted with darker color) are added to VCluster and the major differences between these two models are summarized as follows: (1) A *Virtual Cluster Manager* is added to the physical cluster front. When a user request submitted through the *Grid Middleware* comes in, the *Virtual Cluster Manager* dynamically creates (the fat hollow arrows in Fig. 1(B)) a qualified virtual cluster by orchestrating Virtual Machine Monitors (VMM, e.g., VMware and Xen) on a set of *Physical Cluster Nodes*. (2) The *Grid Resource Controller* is moved from the physical cluster (PCluster) to the dynamically created virtual cluster (VCluster). When a virtual cluster is successfully constructed, the *Grid Resource Controller* and the *Virtual Cluster Scheduler* are deployed on the virtual cluster front node. The *Grid Resource Controller* also has to register itself to the *Grid Middleware* so that the *Grid Middleware* is informed of the availability of new resources. (3) The *Grid Middleware* submits (the thin arrows in Fig. 1(B)) user requests to the *Grid Resource Controller* deployed on the virtual cluster front instead of the physical cluster front (PCluster).

The drawback of VCluster is that virtual machines in VCluster may cause some overhead. Therefore, the system performance may be degraded. To reduce the overhead, the unnecessary virtualization should be avoided.

2.3. TopCluster

Our TopCluster model, as shown in Fig. 1(C), utilizes the advantages of PCluster and VCluster and overcomes their shortcomings. It is composed of four modules: *Scheduler Management*, *Virtual Machine Management*, *Application Deployment Management*, and *TopCluster Master*.

- *TopCluster Master*: is a backend progress and responsible for orchestrating the other three modules.
- *Scheduler Management*: contains a *Scheduler manager*, which is in charge of managing jobs (see thin arrows in Fig. 1(C)) and interacting with *Cluster Scheduler*.
- *Virtual Machine Management*: is composed of a manager and a group of agents. The manager is installed in the *Physical Cluster Front*. The agents are on the *Physical Cluster Node*. This module is responsible for the deployment and undeployment of virtual machines on physical nodes (see fat hollow arrows in Fig. 1(C)).
- *Application Deployment Management*: is also composed of a manager and a group of agents. The manager is installed in the *Physical Cluster Front*. The agents are on the *Virtual Cluster Node*. This module is responsible for the deployment and undeployment of applications on computing nodes (see fat solid arrows in Fig. 1(C)).

When a new request from the *Grid Middleware* arrives, the *Grid Resource Controller* passes it to the *TopCluster Master*, which manages the request during its whole lifetime. TopCluster takes the following four steps to handle a user request.

First, *TopCluster Master* calls *Scheduler Management* to allocate resources for the user request. If the available physical resources are qualified for the request, the physical resources are directly allocated. Otherwise, *Scheduler Management* selects some nodes of the physical cluster by starting a placeholder job on them. Then, *Virtual Machine Management* starts booting virtual machines on the selected physical nodes. Each virtual machine can be considered as a virtual node. For a job which needs N nodes, N virtual machines will be created. And the N virtual machines will be organized as a virtual cluster to specially run the application. When there is not enough disk space to accommodate a new virtual machine, our value-based strategy (Section 2) is invoked by *Virtual Machine Manager* to undeploy some virtual machines with less value in order to free space for the new virtual machine.

Second, after the virtual nodes are all successfully booted, *Application Deployment Management* edits application deployment documents to describe automatic deployment operations required to be executed on the virtual nodes. These documents, defined with ADDL (Section 4), are then sent to the virtual nodes by *Application Deployment Manager*. According to the information contained in the documents, application packages are downloaded from a central application repository, and then automatically installed and configured on the virtual nodes by *Application Deployment Agent*. When there is no enough space for a new application, our value-based strategy (Section 3) is called by *Application Deployment Manager* to properly undeploy some applications with less value to free space so that the newly requested application can be accommodated.

Third, *Scheduler Management* obtains information about the selected virtual nodes from *Application Deployment Management*. It registers virtual nodes to *Cluster Scheduler*, and then submits the user job to *Cluster Scheduler*. *Cluster Scheduler* then allocates the registered virtual nodes for the user job and starts it.

Last, when the user job exits, *Scheduler Management* unregisters the related virtual nodes and stops the placeholder job (started in the first step). *Virtual Machine Management* is then invoked to destroy or suspend the related virtual machines.

Compared with the other two models (PCluster and VCluster), TopCluster has the following three features: (1) TopCluster closely combines the physical cluster and the virtual cluster. As opposed to VCluster, TopCluster relies on single *Cluster Scheduler* (e.g., OpenPBS [1]) on the physical cluster to schedule jobs and manage all (physical and virtual) resources. For the VCluster, each virtual cluster contains its own virtual front node. On each virtual front node, there deploys a cluster scheduler (e.g., OpenPBS [1]), which is specially for the virtual cluster. While, for the TopCluster, there is no cluster scheduler (e.g., OpenPBS [1]) on any virtual node. All the virtual nodes and physical nodes are managed by the single cluster scheduler (e.g., OpenPBS [1]) deployed on the front node of the physical cluster. This feature is named Hybrid Batch System (HyBS) in this paper. If the physical resources are qualified for a user request, they can be directly allocated and therefore virtualization overhead can be avoided. Otherwise, TopCluster dynamically deploys customized virtual clusters for the request. The virtual nodes are dynamically registered and unregistered to *Cluster Scheduler*.

The deployment frequency of virtual machines is determined by the scale of resources and the requirements of jobs. The less is the number of a specific operating system, the bigger is the probability of deploying a virtual machine with the specific system. If there are more jobs request for a specific operating system, the bigger is the probability of deploying a virtual machine with the specific system.

The deployment frequency of an application is determined by the number of resources it has been deployed on and the requirements of jobs. The more physical and virtual resources a specific application has been deployed on, the less is the probability of deploying the application. If there are more jobs request for a specific application, the bigger is the probability of deploying the application.

In the experiment, the scale of the physical resources and the initial deployment of application are fixed. The deployment frequency of virtual machines and applications are mainly influenced by the requirements of jobs, which are then determined by the workload trace [30] from Los Alamos National Lab.

3. Value-based deployment strategy

Reducing overhead caused by the dynamic deployment of virtual machines and applications is an issue that has to be dealt with by TopCluster. This is because resource virtualization and dynamic application deployment are two important features of TopCluster; however, both of them can cause some overhead and therefore degrade the overall system performance. Therefore, in order to reduce the overhead and improve the system performance, we, in this section, introduce our value-based strategy, which is particularly proposed for TopCluster.

3.1. Virtual machine deployment strategy

TopCluster exploits the Virtual Machine Cache Pool (VMCP) to enable the reusability of virtual machines and applications. VMCP is co-located with *Virtual Machine Manager* (Fig. 1(C)) and also maintained by it. When a Grid job requests resources, requirements for virtual machines are extracted from its JSDL [21] document and then are passed to *Virtual Machine Manager*. *Virtual Machine Manager* searches VMCP for qualified virtual machines, which should at least meet the hard requirements (e.g., CPU number, operating system) of the Grid job. It's preferred, but not enforced to meet the soft requirements (e.g., deployed applications) though. If the number of qualified virtual machines is less than the number of requested virtual machines, *Virtual Machine Manager* then informs the *Virtual Machine Agents* on the physical nodes to create virtual machines for the job.

When a Grid job is terminated, its corresponding virtual machines are suspended, rather than destroyed. Their handlers are cached in VMCP for future potential reuse. An expiration time is set for each virtual machine in VMCP, which determines when a virtual machine is not reusable anymore, and therefore should be destroyed. If no virtual machine is expired and the free space on the (limited) local disk of a physical host is not enough to accommodate a new virtual machine, then virtual machines with minimum value are selected from VMCP to be destroyed to free space for the new virtual machine. As the following expression shows, the value of the j_{th} virtual machine is defined as the sum of the values of all the applications deployed on it. How to decide the value of an application is discussed in the next subsection.

$$Value_j^{vm} = \sum_{k=1}^{K} B_k^j Value_k^{app},$$
$$B_k^j = \begin{cases} 1 & \text{the } k_{th} \text{ has been deployed on the } j_{th} \text{ virtual machine,} \\ 0 & \text{the } k_{th} \text{ is not deployed on the } j_{th} \text{ virtual machine} \end{cases} \tag{1}$$

where K is the total number of applications in the central application repository.

3.2. Application deployment strategy

Since virtual machines can be reused in TopCluster, as previously discussed, it is then necessary to have a strategy for dynamic deployment of applications. Each virtual machine usually has its own virtual disks, which are one or more files located in the host system. However, the space restrictions of the virtual disks are specified when these virtual machines are created. These limitations are fixed and cannot be revised after the virtual machines are booted. Therefore, if a virtual machine is reused for a number of times or large size applications are deployed, its virtual disks may be filled up. In order to deploy a new application, some older applications have to be undeployed to free disk space. We therefore need a strategy to determine what applications should be undeployed in such situations.

The objective of an application deployment strategy is to choose proper applications to undeploy in order to minimize the overhead caused by application deployment. To design a good deployment strategy for TopCluster, the following three factors must be taken into account: package sizes, access frequencies, and replica numbers of applications. During the procedure of deployment, an application may need to be downloaded, unpackaged, uncompressed, copied, compiled, installed, and configured. For an application with larger size, it is more likely to take a longer time to deploy it. Therefore, it is preferred to undeploy applications with smaller sizes to minimize the deployment overhead. Besides, a frequently-accessed application (a hot application) might be requested in a short time after it is undeployed, and undeploying a hot application may cause more significant overhead than undeploying a rarely-accessed application. So, it is preferred to undeploy a rarely-accessed application instead of a hot application. Additionally, if an application has been deployed on many resources, there would be less opportunity to deploy it when the application is requested. Therefore, this kind of application should be considered to be undeployed.

In the case that a new application needs to be deployed on a resource, but there is no sufficient free space available, *Application Deployment Manager* exploits the application deployment strategy to select older applications with minimum value to undeploy. The rest of this section describes how to determine the value of an application.

Let F_i be the access frequency of the i_{th} application, N be the total number of applications, and P_i be the probability that a request is for the i_{th} application. Then P_i can be defined as the following expression:

$$P_i = \frac{F_i}{\sum_{k=1}^{N}(F_k)}. \tag{2}$$

Let D_i be the overhead caused by deploying the i_{th} application. As mentioned previously, during its deployment the application may need to be downloaded, unpackaged, uncompressed, copied, compiled, installed, and configured and the size of the application significantly impacts the deployment overhead. Therefore, D_i can be considered as a function of the application size. In other words, D_i increases along with the increase of the size of an application in most cases. To simplify the problem, D_i can be estimated using the following expression:

$$D_i = \alpha_i \times f_{download}(size_i) + \beta_i \times f_{unpackage}(size_i) + \chi_i \times f_{uncompress}(size_i) + \delta_i \times f_{copy}(size_i) \\ + \varepsilon_i \times f_{compile}(size_i) + \phi_i \times f_{install}(size_i) + \varphi_i \times f_{configure}(size_i) + \gamma_i. \tag{3}$$

In the above expression, the six $f_{\times\times\times}(size_i)$ are the functions used to calculate the time to download, unpackage, uncompress, copy, compile, install, and configure the i_{th} application, respectively. These functions are statically set and the same for all applications. A set of coefficients (i.e., α_i, β_i, χ_i, δ_i, ε_i, ϕ_i, φ_i and γ_i) is also statically set for the i_{th} application to adjust the time obtained with the six $f_{\times\times\times}(size_i)$ for the i_{th} application.

Let M_i be the replica number of the i_{th} application deployed in all existing virtual machines. Then, the value of the i_{th} application is defined as the following expression. Recall that our value-based strategy selects applications with less value to free space so that a new application can be accommodated. In other words, if the package size of an application is small (small D_i), its access frequency is low (low P_i), and its replica number is large (large M_i), then this application would be a candidate to undeploy

$$Value_i^{app} = \frac{D_i \times P_i}{M_i}. \tag{4}$$

4. Application Deployment Description Language (ADDL)

ADDL is an XML-based language designed to provide a system-independent description approach for application deployment operations. It is exploited in TopCluster to define *application deployment template* and *application deployment document*. Each application has a static *Application deployment template* to define its installation and configuration. Each *application deployment document* is a runtime document, generated by instantiating the *Application deployment template* according to a user request and the local setting of a virtual node. Due to diverse requirements of scientific computation (most Grid applications are developed for scientific researches) and flexible software development, ADDL is expected to have the following important features: system-independency, parameterization, reusability, localization, and parallelization, each of which is described in the rest of the section.

4.1. System-independency

Applications could be very diverse with regard to their programming language, compatible platform, dependent software, installation and configuration method, and execution environment. To construct a proper execution environment for diverse applications, ADDL documents need to be recognizable for various systems. XML is able "to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion" [23]. So, it is selected as the foundation of ADDL. With different parsers for different operating systems, ADDL can be easily transplanted into them, and therefore it is system-independent.

4.2. Parameterization

Some information required in the *application deployment template* (e.g., the identification number of a job) is unknown until runtime, which therefore has to be represented by a temporary variable in an ADDL document. During runtime, when the required information is available, the temporary variable is populated by its actual information as a value of the variable. The variable is defined as a property in ADDL:

```
<properties>
        <property name="arch" value="x86"/>
</properties>
```

Some repeatedly-used information can also take advantage of parameterization. Temporary variables (properties in ADDL) can be used and then populated with actual values during runtime. By doing so, administrators are able to modify values of the variables conveniently.

Other than user-defined properties, ADDL also specifies some internal/pre-defined properties for special purposes. For example, dependency relationships between applications are recorded in the property "software.require":

```
<properties>
        <group name="software">
                <property name="require" value="mpich2"/>
        </group>
</properties>
```

4.3. Reusability

Generally, reusability is very helpful to rapidly develop, modify, and maintain a software project. In our context, high reusability can also bring benefits to describe application deployment from the following several aspects:

(1) Reusability across different applications

Although *application deployment templates* are rarely completely identical to each other, they often share common fragments. For example, many applications for Linux have the common installation procedure: "configure; make; make install;". It is more convenient to call a predefined procedure instead of repeatedly copying the sequential operations: "configure; make; make install;" for each application. When the above installation procedure needs to be changed to "configure –prefix=/home/program; make; make install;", the administrator only needs to modify the predefined procedure without bothering to check all applications. An element "import" is specified in ADDL for the convenience of importing a common procedure library to an application. The example below describes a situation of importing the "InstallScript" action from the common procedure library rpm. ADDL:

```
<import>
        <library>
                    /home/Application/Library/rpm.ADDL
        </library>
        <actionpath>
                    /top/InstallScript
        </actionpath>
</import>
```

(2) Reusability across different versions of a single application

For the multiple versions of a single application, their *application deployment templates* may be highly similar to each other; an updated version of the application may highly duplicate its old version with only a few modifications. It is better to let the updated version inherit the part that is common with its old version. Element "extend", specified in ADDL, is used in such a case to import an ADDL document of the old version (e.g., 1.0) to the updated version (e.g., 2.0) of the application (e.g., MPICH2 [24]):

```
<extend>
        /home/Application/MPICH2/1.0/linux.ADDL
</extend>
```

The contents (e.g., version 1.0 of MPICH2) imported by "extend" cannot replace the contents with the same element name in the current ADDL document (e.g., version 2.0 of MPICH2). Therefore, the new deployment operations of version 2.0 of MPICH2 can cover the original deployment operations of version 1.0 of MPICH2. "extend" can be used to presents the inheritance relationship between ADDL documents.

(3) Reusability within a single ADDL document

A segment of a configuration procedure can be extracted from the original procedure and form a sub-procedure. With the mechanism, multiple procedures can share sub-procedures, and a complex procedure can be better structured. And therefore, more readable. A sub-procedure can be referenced in another procedure by using the element "actionref":

```
<action name="deploy">
        <actionref >InstallScript</actionref>
</action>
```

4.4. Localization

Local configurations of virtual nodes are usually not identical. For example, hostnames and IP addresses of virtual nodes have to be unique in a local domain. These unique properties (e.g., hostname) of a virtual machine should be saved as an individual ADDL profile document. This document can be imported by *application deployment documents* of applications. The element "include", as shown in the following example, is used to import a local profile to an application:

```
<include>
        /home/program/TopCluster/local.ADDL
</include>
```

The priority of “include” is different from that of “extend”. The contents imported by “include” in an ADDL document can replace its existing information with the same element name. It is because local settings should be loaded to cover general settings.

4.5. Parallelization

Parallel and distributed computing are technologies commonly used to achieve high performance for scientific computations. To support them, *application deployment documents* should be able to support installation and configuration of applications on multiple virtual nodes, which is the case with ADDL. Furthermore, installation and configuration procedures may not be identical for each node in a virtual cluster. For example, MPICH2, widely used in parallel computing, needs to execute the “mpdboot” command on only one node of the virtual cluster. Therefore, in addition to describing the installation and configuration procedures, ADDL supports specifying which portion of the virtual cluster the procedures are actually executed. The following is a fragment of an ADDL document, which illustrates a situation that the “InstallScript” action is executed only on three virtual nodes: vnode2, vnode5, and vnode8:

```
<actionref location= “vnode2, vnode5, vnode8”>
        InstallScript
</actionref>
```

5. TopCluster prototype system

To validate and evaluate TopCluster, we designed a prototype system, which complies with the architecture of TopCluster, discussed in Section 2. The prototype system has four modules: *TopCluster Master, Scheduler Management*, *Virtual Machine Management*, and *Application Deployment Management*. They are all developed in the C++ program language.

Except *TopCluster Master*, the other three modules obtain required information from a JSDL document (see Fig. 1(C)) submitted by a user. The *Scheduler Management* module mainly generates an executable job script from the JSDL document. The *Virtual Machine Management* module needs hard resource requirements (e.g., CPU number) to create qualified virtual machines. The *Application Deployment Management* module completes dynamic deployment of applications according to soft resource requirements (e.g., scientific application). In the rest of the section, we discuss in detail how each of these three modules is implemented in our prototype system.

5.1. Scheduler management

Scheduler Management only contains a *Scheduler Manager* (Fig. 1(C)). *Scheduler Manager* interacts with *Cluster Scheduler* to manage jobs and virtual nodes. In the prototype, we selected OpenPBS [1], an open-source and representative cluster scheduler, as *Cluster Scheduler*. OpenPBS is composed of three components including server, sched, and mom. OpenPBS provides two approaches to interact with its components: user command and batch interface library (IFL). The batch interface library is more flexible to program than user commands; therefore, we chose it in our prototype to submit requests, query status, and operate on the OpenPBS server. *Scheduler Manager* has three main functions: Placeholder Job Management, Virtual Nodes Management, and Grid Job Management, which are described as follows, respectively:

(1) Placeholder job management

Placeholder jobs are used to notify *Cluster Scheduler* that some resources have been taken by virtual clusters and *Cluster Scheduler* should not allocate the occupied resources to other jobs until they are freed. A placeholder is often a very simple job (e.g., sleep for some time), which usually does nothing but occupies some portion of the physical cluster. When a placeholder job is submitted to the *Cluster Scheduler*, OpenPBS allocates physical resources for the placeholder. Then, the *Virtual Machine Management* module is informed of the allocated physical resources, and it therefore dynamically constructs virtual machines. During its lifetime, the placeholder takes a place in *Cluster Scheduler* and holds the physical resources exclusively for its corresponding virtual cluster. When the virtual cluster is stopped or destroyed, the placeholder is cancelled.

(2) Virtual node management

Allocation of virtual nodes for Grid jobs is also performed by *Cluster Scheduler*. Just like allocating physical nodes, *Cluster Scheduler* searches for virtual nodes in its node list to match the resource requirements in the Grid job scripts. Virtual machines are dynamically created by the *Virtual Machine Manangement* module, and are not permanently alive. In order to let *Cluster Scheduler* know when the virtual machines are available, the virtual nodes have to be dynamically added to or deleted from the node list. These two operations are both performed by invoking OpenPBS IFL [25].

(3) Grid job management

According to the resource requirements described in the JSDL document of a Grid job, a virtual cluster is dynamically constructed and virtual nodes are added to *Cluster Scheduler*. Then the JSDL document is transformed into an OpenPBS job script. The following is an example of such a script:

```
#PBS-l nodes= virtual:job000001
```

As shown in the above example, "virtual:job000001" is one of the requirements that should match the properties of virtual nodes when the job is scheduled.

5.2. Virtual machine management

Virtual machines are actually controlled (i.e., created, started, suspended, resumed, stopped, and destroyed) by a Virtual Machine Monitor (VMM) (recall Fig. 1(C)). In our prototype, the Virtual Machine Monitor is the VMware Server 2.0 [26], which supports a wide range of guest and host operating systems (e.g., NetWare and Solaris), and additionally it can be installed on both Windows server and Linux operating systems. In the rest of the section, we discuss the *Virtual Machine Management* module from these two aspects: Virtual Machine Control and Virtual Machine Monitoring.

(1) Virtual machine control

In the prototype, virtual machines are controlled through invoking the programming API (known as "Vix" [27]) of the VMware Server. The *Virtual Machine Management* module has a centralized manager and an agent on each physical host. The *Virtual Machine Manager* receives virtual machine operations and passes them to the *Virtual Machine Agents* on physical hosts. The agents then call the Vix interface of the local VMware Server to perform the operations.

The "create" operation creates new virtual machine profile files (e.g., "vmx" file of virtual machine for VMware), and returns a valid virtual machine handler. Except the "create", all the other operations are performed on a specific virtual machine.

After the virtual machine is started, it has to be configured properly. For example, an IP address should be obtained by the agent from the local IP pool of a physical node, and set for the virtual machine.

(2) Virtual machine monitoring

During the lifetime of a virtual machine, internal errors may occur and serious errors may fail (shutdown or stop) the virtual machine. To effectively tolerate such failures, a virtual machine should be continuously monitored once it boots up. Our prototype system monitors two kinds of failures of virtual machines. The first type of failure occurs during the boot time of a virtual machine. If the virtual machine does not complete the boot within a specific time, it is then considered to have crashed, in which case the virtual machine will simply be rebooted. The second type of failure occurs in the case that the virtual machine has been successfully started, but *Virtual Machine Manager* failed to receive the heartbeat signal from the virtual machine for a specific period of time. In this case, the virtual machine is also considered to have crashed. *Virtual Machine Manager* will therefore destroy all the virtual machines allocated for the Grid job and put it to the head of the job queue for rescheduling.

5.3. Application deployment management

Global paths (e.g., FTP, NFS [28] and local) of application packages can be specified in an ADDL document, so that *Application Deployment Agents* can find the locations to download required application packages. In our prototype, application packages are stored in a NFS directory, which is shared by both physical and virtual nodes. *Application Deployment Agents* only need to copy packages from the NFS directory to the local disk of virtual machines.

6. Experimental setting

With the prototype system, we conducted a set of experiments to validate and evaluate TopCluster. One of the experiments (Experiment 1) was conducted on a small cluster with 12 physical nodes. One of them, namely "toppbs", was used as *Cluster Scheduler*, where the OpenPBS server and sched are installed. Another one, denoted as "topfront", is used as the central *Master* of the prototype, where *Scheduler Manager, Virtual Machine Manager,* and *Application Deployment Manager* were installed. The other 10 nodes are computing nodes, where *Virtual Machine Agents* were installed. The hardware configurations of the 10 computing nodes are presented in Table 1. Five of them have redhat Linux AS3 [29] as their operating systems, and the other five nodes have Ubuntu 8.04.

Total of 400 jobs were sequentially submitted to the prototype. The execution times and arrival rates of these jobs were generated according to the trace [30] from the Los Alamos National Lab. mpiBLAST [31] is an open-source, parallel implementation of NCBI BLAST [32]. Four different versions of mpiBLAST were requested by these jobs. The ratio of the requests for these four versions is nearly 4:3:2:1. Half of the job requests were for redhat Linux, and the rest were for Ubuntu.

Fig. 2 shows the major actions during the experiment of TopCluster. There are four physical nodes with different configurations in the figure. Some of them have two processors, the others have a single processor. Some of them have Windows installed, the others have Linux installed. At the beginning of the experiment, only Node1 and Node2 have App1 deployed. Three jobs arrive at the system one by one.

Table 1
Configuration of computing nodes.

CPU model	Core*CPU	Memory size	Storage space	Operating system	Architecture
Intel Xeon 2.13 GHz	2*1	2 GB	160 GB	Ubuntu/RH AS3	x86_64

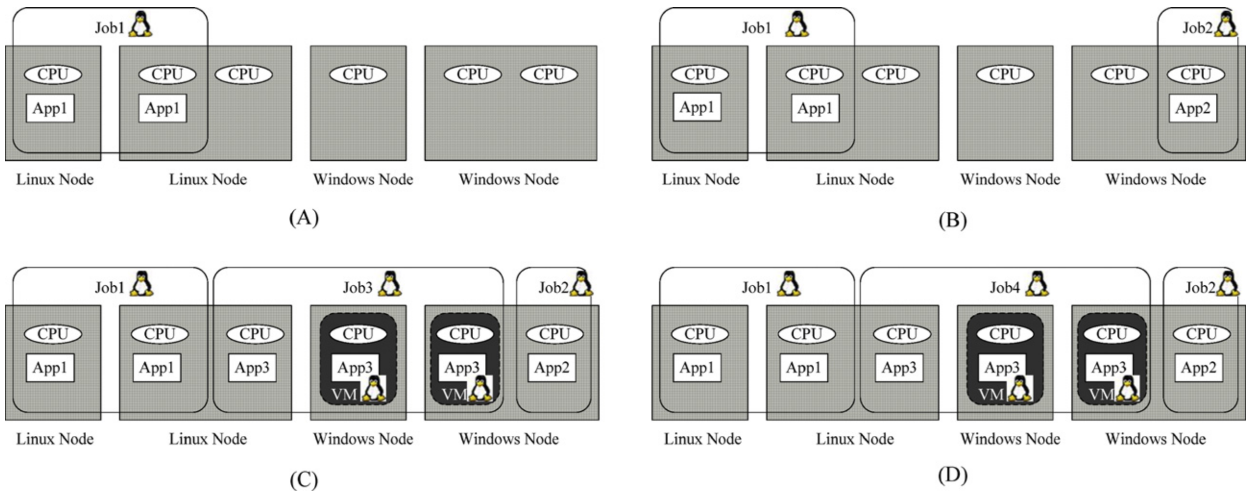

Fig. 2. Major actions during the experiment of TopCluster.

Job1 requires a two-processor system with Linux and App1 deployed. Therefore, Node1 and part of Node2 are allocated to it (see Fig. 2(A)). Job2 requires a single-processor system with Windows and App2 deployed. Part of Node4 is allocated to Job2. Then App2 is dynamically deployed to Node4 (see Fig. 2(B)). Job3 requires three processors with Linux and App3. Because Node3 and Node4 among the available nodes are systems with Windows installed, virtual machines with Linux are required to be dynamically deployed on them. Then App3 is dynamically on physical Node2, virtual VNode1 and VNode2. Then Job3 runs on Node2, VNode1 and VNode2 (see Fig. 2(C)). When Job3 exits, VNode1 and VNode2 are not destroyed immediately. They are suspended and reserved in the Virtual Machine Cache Pool on Node2 and Node3 respectively. Since Job4 also requires three processors with Linux and App3, the cached VNode1 and VNode2 are resumed and directly allocated (see Fig. 2D).

With more and more jobs complete, the virtual machines reserved in the Virtual Machine Cache Pool may be increased to the limit of disk space. In this case, some virtual machines need to be destroyed according to the principle proposed in Section 3.1. If a virtual machine is reallocated for a number of times, more and more deployed applications may also reach the limit of disk space of this virtual machine. In this case, some applications need to be undeployed according to the principle proposed in Section 3.2.

In order to compare with TopCluster, we also conducted two experiments on a cluster without virtual machines (complying with the PCluster model) (Experiment 2) and a virtual machine cluster (complying with the VCluster model) (Experiment 3), respectively. For VCluster, there is also a Virtual Machine Cache Pool (VMCP) implemented inside to save the time of obtaining virtual machines, as TopCluster does. The same 400 jobs mentioned above were also submitted in these two experiments. The one conducted on a cluster without virtual machines (Experiment 1) has the same configurations of the 10 computing nodes as the ones shown in Table 1. The number of nodes where each version of mpiBLAST was deployed equals other versions. The jobs were scheduled to the nodes whose operating system and deployed applications meet user requirements. The other experiment (Experiment 2) was conducted on a VCluster. When a job comes, it was scheduled to dynamically constructed virtual machines.

Strict FIFO was used by OpenPBS in all the three experiments as their scheduling strategies. In other words, jobs have to be started strictly according to the order they are submitted.

7. Experimental results

Based on the result of Experiment 1, we discuss the overall overhead of TopCluster first, and then analyze the overhead caused by obtaining virtual machines and deploying applications, respectively. With the results of Experiments 1, 2, and 3, the overall performance of the three models (PCluster, VCluster, and TopCluster) is analyzed and compared with respects to their job throughputs, resource utilization, and accumulated waiting time.

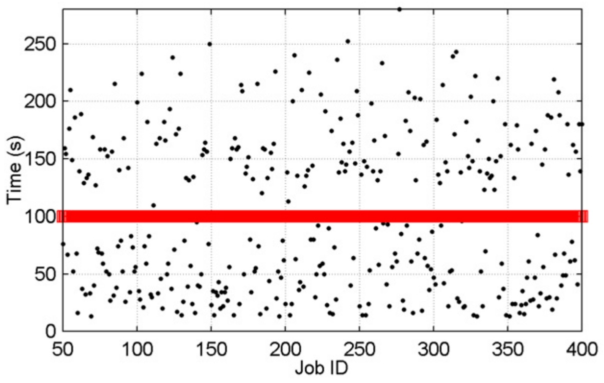

Fig. 3. Obtaining time of virtual machines for all jobs of TopCluster.

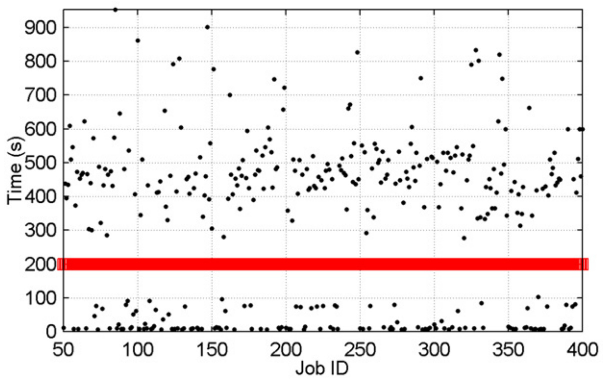

Fig. 4. Application deployment time for all jobs of TopCluster.

7.1. Overall overhead of TopCluster

In our TopCluster prototype system, the lifetime of a job can be divided into three sequential stages: (1) obtaining time of virtual machines, (2) application deployment time, and (3) execution time. The first two stages can be considered as the overhead caused by TopCluster. From the experiment, we observed that the total overhead of TopCluster is less than 1/15 of the job execution time. In other words, the application deployment and the obtaining time of virtual machines is significantly less than the execution time of jobs. This is because (1) TopCluster can flexibly allocate physical or virtual resources for users. The overhead caused by virtualization can be partially avoided. (2) TopCluster benefits from exploiting our value-based strategy for virtual machine and application deployment (Section 3), which aims to minimize the overhead of TopCluster.

7.2. Overhead of TopCluster caused by obtaining virtual machines

Fig. 3 presents the obtaining time of virtual machines for all 400 jobs in Experiment 1. From the figure, we can see that the obtaining times of virtual machines are mostly within the range of 10 to 250 seconds. The maximum value (331 seconds) of virtual machine obtaining time is almost 25 times its minimum value (13 seconds). We discovered that the obtaining times of virtual machines of the jobs rarely follow within the range of around 100 seconds, highlighted as the red horizontal line. This large variety between the obtaining times of different jobs is because the Value-based strategy (Section 3) of virtual machines can cache virtual machines with higher values to save time.

7.3. Overhead of TopCluster caused by application deployment

Fig. 4 shows the application deployment time for all 400 jobs in Experiment 1. From this figure, we can observe that the time costs to deploy applications are clearly divided into two groups: the ones below the red horizontal line (the range of 200 seconds) and the others above the line. It is measured and known that the installation (execute “cp; tar –zxvf; ./configure; make; make install”) of the mpiBLAST in a virtual machine costs about 360 seconds; therefore, it is clear that the jobs in the second group performed the installation operations. By using value-based strategy of virtual machines and applications, the applications with higher values can be cached. The jobs in the first group did not need to actually

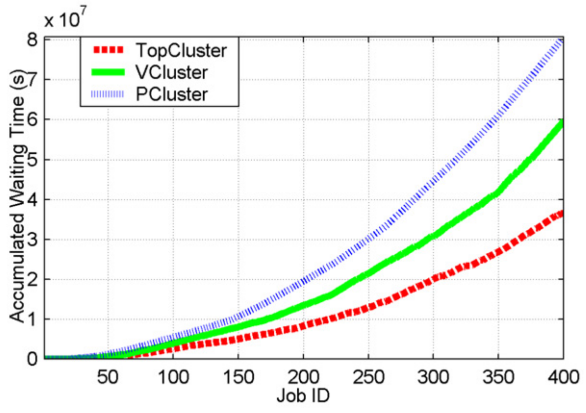

Fig. 5. Job throughputs of three cluster models.

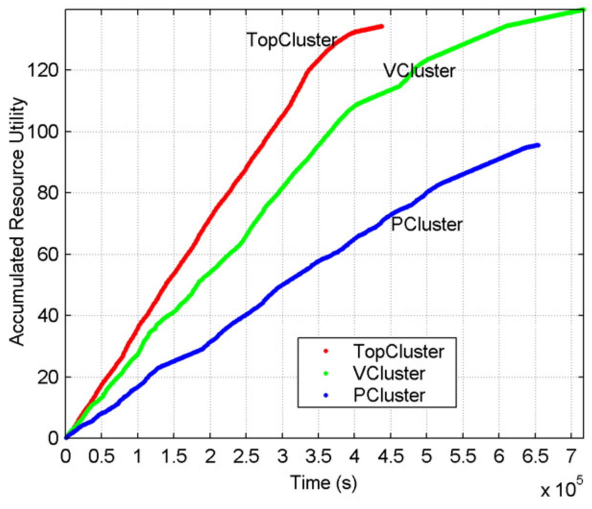

Fig. 6. Resource utilities of three cluster models.

perform the installation operations. These jobs reused those virtual machines where their required applications have just been deployed.

7.4. Overall performance of three cluster models

Fig. 5 presents the job throughputs in the experiments conducted to evaluate all three cluster models. The throughput values were obtained by counting the number of finished jobs during each recording interval (1000 seconds). From the figure, we can see that the final completion times of 400 jobs across three experiments are different. In Experiment 1 (TopCluster), all the 400 jobs finished within 438,000 seconds, which is the earliest. The number of finished jobs for TopCluster is always remarkably larger than that for the other two models. By doing a simple calculation, we know that the finished jobs of TopCluster average about 24.6% (or 46.8%) more than that of VCluster (or PCluster). So, TopCluster can greatly improve the overall throughput of clusters.

The target of TopCluster is to more efficiently make use of the computational capability of the system. The CPU utilization shows the resource utilization of the computational capability for the three models. Fig. 6 shows the resource utilization of the three experiments. Similar to the throughput (Fig. 5), the resource utilization was measured for each interval of 1000 seconds. The resource utilization equals the ratio of the occupied processors to the total processors in a cluster. To underline the trends of variations in three experiments, the figure gives the accumulated value of resource utilization. As shown in the figure, the makespans of the three experiments are different, which also can be observed in Fig. 5. The processor utility of TopCluster averages around 26.3% (or 110.6%) higher than that of VCluster (or PCluster).

Fig. 7 shows the waiting time of all the 400 jobs in the three experiments. It is clear that the jobs in TopCluster received generally shorter waiting times than the other two models. The accumulated waiting times of TopCluster averages 35.1% (or 56.1%) lower than that of VCluster (or PCluster). So, TopCluster can significantly reduce the waiting time of jobs.

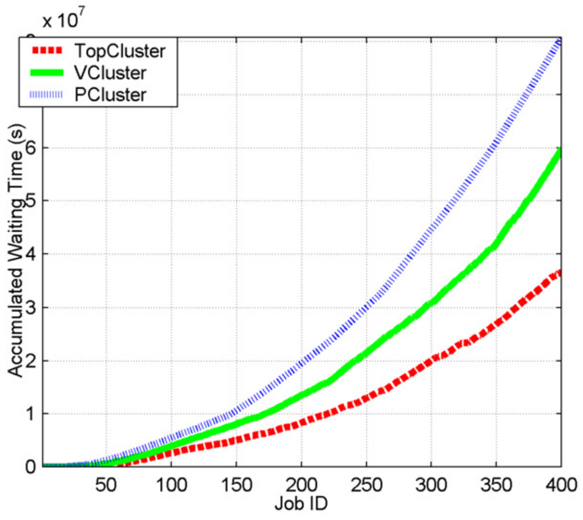

Fig. 7. Accumulated waiting time of all 400 jobs across three models.

In the experiments, 400 jobs are submitted to the three models. From Fig. 7, it can be seen that the waiting times of jobs increase steadily, and the system is busy all the time except in the beginning. At the beginning of the test (the beginning 50 jobs), the clusters are empty. Therefore, the waiting times of jobs are very short. Since the jobs need some time to complete, the clusters will be occupied by the started jobs after a period of time. The jobs submitted later will have to wait for more time to obtain free resources. The system is almost always with a heavy workload after the first 50 jobs. Therefore, the experiment is able to show the response of the three models with heavy workloads.

8. Related work

There are some existing works on virtual machines and clusters in Grid. However, they either only support virtual machines but not virtual clusters, or only support virtual clusters but cannot manage and allocate physical and virtual resources simultaneously.

Some of the existing works only support virtual machines but not virtual clusters. VMPlants [33,34] is a Grid service for configuration and creation of flexible virtual machines. It supports flexible virtual machine configuration based on the DAG specification of a virtual machine. Nova [8] is proposed to reduce the overhead of dynamically creating and destroying virtual clusters. Virtual machine technology was originally introduced into Grid computing by the authors of [35]. The major issues, including performance overhead caused by Virtual Machine Monitor, are discussed in [35]. And their possible implementation technologies are also presented. However, the above related works [8,33–35] do not support virtual clusters. Virtual machines in them are independent of each other, and directly managed by Grid middleware while virtual machines in TopCluster are organized as clusters.

Some of the existing works support virtual clusters, but cannot manage physical resources simultaneously, and do not support dynamic deployment of applications. VW (virtual workspace) [36] is such an existing work. It is an execution environment, which can be dynamically created, configured, and managed. Cluster-On-Demand (COD) [9] is a mechanism to manage dynamic virtual clusters in a Grid site. An adaptive resource management model for Grid is proposed in [37], which is based on COD. This model creates dynamic Grid resources in the container-level by using virtualization technologies. Usher [6] is a virtual machine management system, which combines a core set of interfaces of basic mechanisms, clients, and a framework for administrative policy customization. The approach proposed in [38] creates and manages virtual clusters across multiple sites distributed over wide area networks. A generic infrastructure is proposed in [39] to extend the Workspace Service and implement virtual clusters for Grid while TopCluster can simultaneously manage and flexibly allocate physical and virtualized resources.

There exist several approaches (e.g., ant [13], DistAnt [14], apt [10], rpm [11], CDL [12], DeployWare [15], and tcl [16]) of application deployment that are not suitable for application deployment in TopCluster for several reasons. First, some of them are not system-independent (e.g., apt and rpm are not proper to be used for OSs such as Windows). Second, some are not professional for application deployment: CDL is a configuration description language, and therefore, cannot describe executable operations. Third, some are not suitable for complex scenarios: to efficiently describe, deploy, and maintain a large number of applications in Grid, some critical characteristics such as, reusability, should be ensured. For example, the grammar of build files of ant and DistAnt is not suitable for complex scenarios, though ant is mature to deploy a java application. DeployWare also does not support reusability to ease the management of applications. Tcl is specially for programmers, and too complicated for normal users to master. Fourth, all these approaches, except DistAnt and DeployWare, do not support parallel deployment of applications on distributed resources while ADDL in TopCluster is suitable for TopCluster, because it has the following important characteristics: system-independency, reusability, parameterization, localization, and parallelization.

The recent researches on management policy of virtual machines do not focus on the overall system performance. Resource management policy is proposed by Beloglazov [41] to minimize power consumption by migrating virtual machines and switch off idle nodes. The decentralized affinity-aware migration technique [42] incorporates heterogeneity and dynamism in network topology and job communication patterns to minimize communication overhead. A novel approach named vSaaS [43] is proposed to provide the software as a service from a cloud computing environment over the Internet. An autonomic resource manager [44] aims to optimize a global utility function which integrates both the degree of SLA fulfillment and the operating costs. Different with the above researches, the TopCluster proposed in this paper uniquely try to improve the overall system performance.

9. Conclusion

Before cluster virtualization technologies were applied in Grids, a traditional physical cluster model (PCluster) was often used in high performance computing. PCluster directly allocates physical cluster nodes for jobs and therefore it is not flexible to meet user requirements such as customized operating system configuration. Then, the Virtual-Machine Cluster (VCluster) model came along, in which jobs are submitted to virtual clusters instead of physical clusters. With VCluster, users are permitted to choose virtual machines with a preferred operating system configured based on the users' needs. However, this advantage is gained at the cost of the overhead caused by virtual machines.

In order to reduce the overhead, we proposed a new virtual cluster model denoted as TopCluster. TopCluster can flexibly allocate both physical and virtualized resources for users. If available physical resources are qualified for a user request, they will be directly allocated. Thus, the overhead caused by virtualization can be avoided. If no qualified physical resources can be found, virtual clusters will be dynamically deployed. In order to further reduce the overhead caused by TopCluster, we also proposed, in this paper, a value-based strategy for virtual machine and application deployment.

To support dynamic deployment of applications in various guest operating systems, we also proposed the Application Deployment Description Language (ADDL). ADDL is an XML-based description approach for application deployment operations, which supports the following important characteristics: system-independency, reusability, parameterization, localization, and parallelization.

A prototype system of TopCluster has been implemented. By replaying a real workload trace, a set of experiments were conducted on the prototype system to evaluate the overhead of TopCluster and compare its performance with VCluster and PCluster. Experiment results reveal that the average overhead of TopCluster is less than 1/15 of the job execution time. Our TopCluster model can significantly improve the overall system performance, compared with PCluster and VCluster. The throughput and system utility of resources can be improved by at least 24.6% and 26.3%, respectively. The overall waiting time of jobs can also be reduced by 35.1%.

In the future, we plan to further evaluate TopCluster from the following two aspects: First, the scheduling strategy used to evaluate TopCluster in this paper is the strict FIFO. Our next step is to apply some other strategies. Second, no more than two virtual machines are permitted on an individual physical node in this paper. In the future, we also want to investigate the performance of TopCluster when different numbers of virtual machines deployed on an individual physical node are allowed.

Acknowledgments

This Work is supported by National Basic Research (973) Program of China (2011CB302505, 2007CB310900), Natural Science Foundation of China (61073165, 61040048, 60803121, 60911130371, 90812001, 60963005) and Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

References

[1] OpenPBS home page, http://www.openpbs.org/.
[2] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid: enabling scalable virtual organizations, Int. J. High Perform. Comput. Appl. 15 (2001) 200.
[3] I. Foster, Globus toolkit version 4: Software for service-oriented systems, J. Comput. Sci. Tech. 21 (2006) 513–520.
[4] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurr. Comput. Pract. Exper. 17 (2005) 323–356.
[5] Platform LSF, http://www.platform.com/Products/platform-lsf.
[6] M. McNett, D. Gupta, A. Bahdat, G. Voelker, Usher: An extensible framework for managing clusters of virtual machines, in: Proceedings of the 21st Large Installation System Administration Conference (LISA 07), Dallas, TX, USA, 2007, pp. 1–15.
[7] H. Nishimura, N. Maruyama, S. Matsuoka, Virtual clusters on the fly – fast, scalable, and flexible installation, in: Proceedings of Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07), Rio De Janeiro, Brazil, 2007, pp. 549–556.
[8] S. Sundarrajan, H. Nellitheertha, S. Bhattacharya, N. Arurkar, Nova: An approach to on-demand virtual execution environments for Grids, in: Proceedings of Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), Singapore, 2006, pp. 544–547.
[9] J.S. Chase, D.E. Irwin, L.E. Grit, J.D. Moore, S.E. Sprenkle, Dynamic virtual clusters in a Grid site manager, in: Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing, 2003, pp. 90–100.
[10] apt-get(8) – Linux man page, http://linux.die.net/man/8/apt-get.
[11] RPM Guide, http://docs.fedoraproject.org/drafts/rpm-guide-en/index.html.
[12] CDDLM Configuration Description Language Specification, Version 1.0, http://www.ogf.org/documents/GFD.85.pdf.

[13] Apache Ant, http://ant.apache.org/.
[14] W. Goscinski, D. Abramson, Distributed Ant: A system to support application deployment in the Grid, in: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004.
[15] A. Flissi, J. Dubus, N. Dolet, P. Merle, Deploying on the Grid with DeployWare, in: Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, 2008.
[16] J.K. Ousterhout, Tcl and the Tk Toolkit, Addison–Wesley, Reading, MA, USA, 1994.
[17] Extensible Markup Language (XML) 1.0 (Third Edition), http://www.renderx.com/demos/xmlspec/xml/REC-xml-20040204.pdf.
[18] The open science Grid, http://www.opensciencegrid.org/.
[19] TeraGrid official website, http://www.teraGrid.org.
[20] H. Jin, ChinaGrid: Making Grid computing a reality, in: Digital Libraries: International Collaboration and Cross-Fertilization, in: Lecture Notes in Comput. Sci., vol. 3334, 2004, pp. 13–24.
[21] JSDL workgroup, http://forge.Gridforum.org/projects/jsdl-wg.
[22] Y. Wu, S. Wu, H. Yu, C. Hu, CGSP: An extensible and reconfigurable Grid framework, in: Lecture Notes in Comput. Sci., vol. 3756, 2005, pp. 292–300.
[23] XML – wiki, http://en.wikipedia.org/wiki/XML.
[24] W. Gropp, MPICH2: A new start for MPI implementations, in: Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, Linz, Austria, 2002.
[25] A. Bayucan, R.L. Henderson, C. Lesiak, B. Mann, T. Proett, D. Tweten, Portable Batch System: External reference specification, 1999.
[26] VMware, http://www.vmware.com.
[27] Programming API Programming Guide, http://www.vmware.com/pdf/Prog_API_Prog_Guide.pdf.
[28] Network file system – wiki, http://en.wikipedia.org/wiki/Network_file_system.
[29] Redhat Linux, http://www.redhat.com/.
[30] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/index.html.
[31] A. Darling, L. Carey, W. Feng, The design, implementation, and evaluation of mpiBLAST, in: Proceedings of the 4th International Conference on Linux Clusters: The HPC Revolution 2003 in Conjunction with ClusterWorld Conference & Expo, San Jose, CA, 2003.
[32] NCBI BLAST home, http://blast.ncbi.nlm.nih.gov/Blast.cgi.
[33] I. Krsul, A. Ganguly, J. Zhang, J.A.B. Fortes, R.J. Figueiredo, VMPlants: Providing and managing virtual machine execution environments for Grid computing, in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Pittsburgh, PA, USA, 2004.
[34] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, X. Zhu, From virtualized resources to virtual computing Grids: the In-VIGO system, Future Gener. Comput. Syst. 21 (2005) 896.
[35] R.J. Figueiredo, P.A. Dinda, J.A.B. Fortes, A case for Grid computing on virtual machines, in: Proceedings of the 23rd International Conference on Distributed Computing Systems, Providence, RI, USA, 2003, p. 550.
[36] K. Keahey, I. Foster, T. Freeman, X. Zhang, D. Galron, Virtual Workspaces in the Grid, Lecture Notes in Comput. Sci., vol. 3648, 2005, p. 421.
[37] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, J. Chase, Toward a doctrine of containment: Grid hosting with adaptive resource control, in: Proceedings of ACM/IEEE SC 2006 Conference (SC'06), Tampa, Florida, 2006, p. 20.
[38] T. Hirofuchi, T. Yokoi, T. Ebara, Y. Tanimura, A multi-site virtual cluster system for wide area networks, in: Proceedings of First USENIX Workshop on Large-Scale Computing, Boston, Massachusetts, 2008.
[39] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, X. Zhang, Virtual clusters for Grid communities, in: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 06), Singapore, 2006, pp. 513–520.
[40] A Performance Comparison of Hypervisors, http://www.vmware.com/pdf/hypervisor_performance.pdf.
[41] Beloglazov, Anton Buyya Rajkumar, Energy efficient allocation of virtual machines in cloud data centers, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010), Melbourne, Australia, 2010, p. 577.
[42] J. Sonnek, J. Greensky, R. Reutiman, A. Chandra, Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration, in: Proceedings of the 39th International Conference on Parallel Processing (ICPP 2010), San Diego, CA, 2010, p. 228.
[43] Liang Zhong, Tianyu Wo, Jianxin Li, Bo Li, A virtualization-based SaaS enabling architecture for cloud computing, in: Proceedings of the Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010), Cancun, Mexico, 2010, p. 144.
[44] H.N. Van, F.D. Tran, J.-M. Menaud, Autonomic virtual resource management for service hosting platforms, in: ICSE Workshop on Software Engineering Challenges of Cloud Computing, Vancouver, BC, 2009, p. 1.