Virtual Clusters on the Fly — Fast, Scalable, and Flexible Installation

Hideo Nishimura[†], Naoya Maruyama[†], and Satoshi Matsuoka[†,‡]
[†] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
[‡] National Institute of Informatics
nish@matsulab.is.titech.ac.jp, naoya.maruyama@is.titech.ac.jp, matsu@is.titech.ac.jp

Abstract

One of the advantages in virtualized computing clusters compared to traditional shared HPC environments is their ability to accommodate user-specific system customization. However, past attempts to providing virtual clusters are not scalable with increasing number of VMs, nor do they allow fine-grained customization of VMs, assuming that preconfigured VM images are always available on the Grid. We propose a new virtual cluster installation technique that achieves efficiency and scalability, and yet simultaneously fine-grained customizability. It allows the user to create VMs on the fly for fine-grained customization of VMs, and pipelined data transfer for scalable installation with increasing number of VMs. To achieve efficiency in the presence of such full customization, it automatically caches frequently-constructed virtual disk images to save software installation time in common cases. Our experimental studies using a prototype implementation show that installation of a 190-node virtual cluster can be done in 40 seconds. From this result along with a scalability study, we estimate that installation of a 1000-node virtual cluster could be done in less than two minutes.

1 Introduction

Virtual clusters are virtualized computing clusters that consist of underlying multiple clusters on the Grid interconnected by physical or overlay networks. One of the main advantages in such environments is their ability to accommodate user-specific customization of system configurations without destructively interfering with underlying hosts [7, 11]. Possible customization ranges from difference of version number of a particular application to selection of different operating system kernels. To realize such a potential advantage, an automated installation system that creates and configures a virtual cluster based on a given user request is necessary, since manual creation and configuration would be time-consuming and error-prone, if being pragmatic at all. Such a system should provide a simple yet flexible installation and customization interface to the users, as do typical cluster installers [14, 16]. Besides, to be used as a transparent alternative to the traditional physical platforms, a virtual cluster must be constructed and configured instantly. Furthermore, such an installation system needs to work at scale: virtual clusters consisting of hundreds to thousands of VMs would not be exceptional but commonplace.

Some of the issues in the implementation of virtual clusters have been addressed by Krsul [11] and Foster [7]. Krsul et al. proposed a framework to create VMs and configure them as specified in a DAG-based custom schema. Dependency between each configuration, such as software installation and network settings, is encoded as edges in the specification DAG. A VM is instantiated by cloning a predefined VM image and applying the configuration DAG to the cloned VM. Foster et al. proposed a mechanism to deploy user-specific computing environment on Grids by extending the Virtual Workspace Service [10]. It takes a VM image with its configuration information as input and deploys it to allocated physical resources requested by the user. In these existing approaches, insufficient attention is paid to the issue of how to prepare the VM images themselves, and instead they implicitly assume that for each user request there is a VM image that can be adapted to the request with little extra configuration. While such an assumption might hold in computing environments for a small number of users, it will not be the case in large-scale environments shared by multiple organizations, where the user requests can be significantly diverse. In addition, none of the existing projects addresses the issue of scalable installation, as predefined images are typically distributed through NFS, resulting in linear installation time [7]. Such poor scalability is more problematic as the number of nodes increases, since NFS is not likely to scale well for a large number of clients.

We propose a new approach to virtual cluster management that achieves flexible fully-customizable system installation by on-the-fly VM creation, rather than assuming

the existence of appropriate VM images. We employ an existing cluster installer and extend it for virtual cluster installation. This design decision allows the user to express software configuration of virtual clusters as easily as the employed cluster installer. Furthermore, to reduce installation time, we propose an installation caching technique that creates *virtual disk caches*, VD caches, where frequently-requested software is already installed, yet not configured. This technique specifically aims to reduce software installation time, which is the biggest bottleneck in typical installation scenarios (see Section 3.1). Unlike the offline-created images assumed in previous work [10, 11], a VD cache is automatically created upon user requests and is destroyed to keep the total cache size within the given space in an LRU fashion. We further reduce installation time by exploiting a scalable pipelined data distribution technique for deploying install packages and VD caches [13]. Overall, our final goal is that it will take less than 20 seconds for the user to create a virtual cluster of thousands of VMs. That is to say, it will take only 20 seconds from the time when the user issues a request to create the cluster till the time when he has exclusive access to the cluster to login and launch jobs.

The contributions of this paper are as follows:

- We present an architecture of virtual cluster installation system using three key techniques: a cluster installer, pipelined data transfer, and VD caching. Our VD caching algorithm automatically finds frequently-used software combinations by using a statistical cluster analysis technique [9]. We define dissimilarity between each user request and cluster them based on the dissimilarity. For each identified cluster, we create a VD cache where the common software in the cluster is installed, yet not configured. As far as we know, the application of cluster analysis to cache selection has not been reported in literature. We believe that it would also be applicable to other repetitive computing tasks that support partial computation.

- We present preliminary performance results obtained with a prototype implementation using a cluster installer called Lucie [16], and show that the prototype can setup a virtual cluster of 67 VMs of typical configurations in 113 seconds in average without caching, and 39 seconds with caching. We also demonstrate that the prototype achieves near-O(1) installation time, indicating that it will install 1000 virtual cluster nodes in less than two minutes. These results indicate that the speed of our installation system significantly outperforms those of the previous work.

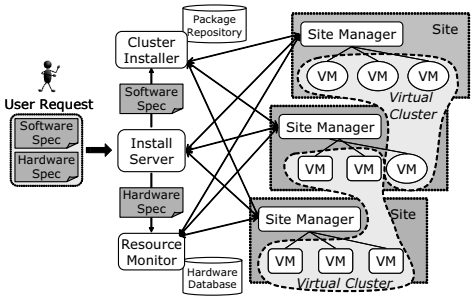

Figure 1. Overview of the virtual cluster installation system.

2 Overview of the Virtual Cluster Installation System

In this section, we describe the high-level overview of the virtual cluster installation system. Given an installation request from the user, the system selects physical resources to host a virtual cluster for the request, instantiates a set of new VMs, and finally installs operating system and other requested software to them. Figure 1 illustrates the overview of the system architecture, which consists of three key components *install server*, *cluster installer*, and *site manager*.

Note that, although the primary purpose of a virtual cluster is to provide a job execution environment, we deliberately skip the discussion of how a user job is shipped to and executed on a virtual cluster; our focus in this paper is creation of such execution environments. Other necessary functions could be implemented, for example, with the interoperable Grid services proposed in [10]. Also, to schedule a requested virtual cluster to the appropriate hosting resources, we need to collect hardware and usage statistics of available physical resources distributed across wide-area networks. For this purpose, existing monitoring middleware for Grids, such as Network Weather Service [17], could be used straightforwardly; we do not discuss resource monitoring in the rest of this paper.

2.1 Install Request Submission

The user initiates a virtual cluster creation by sending a request to the install server. The request consists of hardware and software specifications, each of which describes the requirements of either hardware or software. The hardware specification includes CPU type and speed, RAM amount, disk space, and the number of nodes, The software specification describes OS kernel and other user-level packages to be installed. The structure of the software specification depends on the particular cluster installer being used.

2.2 Virtual Cluster Creation for a User Request

The install server initiates virtual cluster creation upon receiving a user request. The scheduler contacts the resource monitor for physical resources that satisfy the hardware specification in the user request. For each selected host, the scheduler requests the site master to create a VM on the host, and contacts the cluster installer to install the requested software specified in the software specification to the VMs. Note that, depending on a specific implementation, it could be necessary to extend the cluster installer to be able to install machines distributed over multiple network sites.

2.3 Software Installation to Virtual Clusters

Installation of software is done by the cluster installer with necessary extension for multi-site installation and the specific VMM being used. For simplicity, we assume as an installation model that the cluster installer installs software using a *package manager* along with an extensive collection of pre-compiled binary packages stored in *package repositories*. A package manager supports package installation, deletion, and dependency resolution of packages. Such an assumption is valid in one of the popular cluster installers [14]; we believe that it is the case for most of other such tools, given the fact that software installation and maintenance with binary packages is a well accepted convention in most Linux distributions.

Using a cluster installer, the packages requested by the software specification are fetched from package repositories, and transferred to each VM. The packages are then unpacked and copied to the VM and configured as specified by the software specification.

3 Optimization of Package Installation Time

In this section, we discuss our package installation time optimization technique. First, we show that the most dominant bottleneck step in package installation time is the time to extract software packages. Next, we present the VD caching technique that aims to optimize the bottleneck step. Finally, we show the overall virtual cluster installation algorithms with the VD caching functionality.

3.1 Preliminary Analysis of Installation Time

In the installation steps described in Section 2.3, principal part of the time to install packages for a virtual cluster is divided into the following three sub-steps:

Table 1. Package installation time onto 67 VMs

Number of packages	Size (MB)	Total sec	Transfer sec	Transfer %	Extraction sec	Extraction %	Config. sec	Config. %
18	14.8	15.1	3.6	24	11.0	73	0.5	3.6
33	31.8	22.3	4.1	18	17.7	79	0.5	2.5
51	57.8	43.2	4.9	11	37.8	87	0.5	1.3
94	90.6	70.3	5.9	8.4	63.8	91	0.5	0.8

Package transfer time The time to make the necessary packages available on each cluster node.

Package extraction time The time to extract and copy the contents of packages to the virtual disk, as well as to check the dependency among the packages.

Package configuration time The time to configure the installed packages.

Table 1 shows the breakdown of the above steps in a preliminary performance study installing several numbers of packages onto a 64-node virtual cluster, using the same environment used in Section 5. The result indicates that the most dominant factor in the installation steps is the package extraction time.

3.2 Optimization Methodology

To optimize the package extraction time, we propose a VD caching technique that is based on the assumption that user requests exhibit time locality, i.e., recently-used packages are likely to be used again in the near future. A VD cache is a virtual disk image where a frequently-requested combination of packages is already installed, but not yet configured. With VD caching, for each user request, the installation process first looks for the largest VD cache that is a subset of the requested package set and uses a copy of the VD cache as an initial disk image. It then resolves the difference between the VD cache and the user request by installing the remaining packages. By starting from a VD cache instead of a scratch disk, it avoids the transfer and extraction time of packages included in the VD cache.

In VD caching, a method to select the combinations of packages for caching is a key to effectively reducing installation time. One could use a simple method that produces only a single VD cache that only stores the common packages throughout the history of all requests. Such a naïve method, however, would achieve little installation time reduction when there was substantial diversity in the requests

issued by many users. In such a case, the size of the resulting VD cache would in fact be quite small compared to size of the individual request, thereby only reducing a small fraction of the total installation time.

To automatically select optimal package combinations for caching, we first identify groups of similar requests by a hierarchical cluster analysis [9], as explained below. Each cluster identified by the analysis is a possible caching candidate: a VD cache containing the commonality of the requests in the cluster. Among the candidates, we select as many clusters as the space available for VD caches allows. To do so, we compute a ranking of each cluster that reflects expected contribution to installation time reduction, and select the highest rankings within the given cache space.

3.2.1 Dissimilarity between User Requests

Given the above software installation model, we define the dissimilarity between user requests such that it effectively quantifies the sizes of the commonalities and dissimilarities among multiple requests. Let r be a request, $P(r)$ be the set of packages that r specifies to install, and $|P(r)|$ be the total size of the packages. Then, we define the dissimilarity between requests r_1 and r_2, $d(r_1, r_2)$, as the inverse of the size of the common packages: $d(r_1, r_2) = |P(r_1) \cap P(r_2)|^{-1}$. If the size of the common packages is 0, we define it as ∞.

For example, suppose that there are three user requests, r_1, r_2, and r_3: $P(r_1)$ is $\{\text{Fortran}, \text{Python}\}$, $P(r_2)$ is $\{\text{Python}\}$, and $P(r_3)$ is $\{\text{Fortran}\}$. In the Debian Linux distribution for the i386 platform, the size of Fortran and Python is 1.6MB and 3.4MB, respectively. Hence, $d(r_1, r_2)$ is calculated as 0.3, whereas $d(r_1, r_3)$ is 0.63. Note that the fact that $d(r_1, r_2)$ is smaller than $d(r_1, r_3)$ reflects that the pair of r_1 and r_2 has a larger common package (i.e., Python) than that of r_1 and r_3 (i.e., Fortran).

3.2.2 Clustering User Request History

To find caching candidates, we apply hierarchical cluster analysis to user request history [9]. Given the dissimilarity metric, the analysis first finds the closest pair of requests, and replaces the pair with a new cluster consisting of the pair. The analysis then repeats the step until every request is merged into a single cluster. Note that the dissimilarity between clusters is defined as the inverse of the size of the common packages of the clusters. The final result is a dendrogram representing the clustering steps. Figure 2 depicts the dendrogram of the clustering of the example requests introduced in Section 3.2.1.

3.2.3 Selecting Caching Candidates

Our final step is to construct the ranking of each package cluster for cache selection, and select as many caches as the available space for caches allows. For this purpose, we compute a *speedup score* of each cluster that estimates the impact on installation time reduction. We expect that the more frequently a package cluster appears in the history of user requests, the higher the probability of reappearance of the cluster in the future will be. Furthermore, we estimate that there is a linear relationship between the installation time reduction and the total size of a package combination. Therefore, we define the speedup score as a product of the frequency of appearance of the cluster and the total size of the packages in the cluster. As is the case with the dissimilarity metric, the score is defined as ∞ when there is no common package in a cluster.

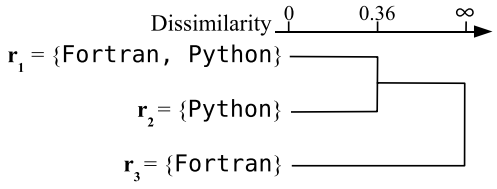

Figure 2. An example dendrogram.

We compute the final ranking based on the speedup scores of clusters. Let C be the set of clusters in the result of the cluster analysis in the previous step, and c_i be a cluster in C, where $1 \leq i \leq N-1$ and N is the number of requests. Further, let $D(c_i)$ and $A(c_i)$ be the descendants and ascendants of c_i in the dendrogram, respectively. Then, we perform an iterative analysis as follows, and select as many caches as the available space allows:

1: Compute the speedup score of each cluster.
2: Give the next highest rank to the cluster of the highest speedup score, c_m, and remove it from C.
3: For each cluster in $A(c_m)$, adjust the frequency by subtracting the frequency of c_m.
4: For each cluster in $D(c_m)$, adjust the size by subtracting the size of c_m.
5: Continue the above steps until no cluster is left in C.

3.3 Overall Installation Process

Figure 3 depicts the overall installation process, where the cache manager is a component in each site that monitors the user request history for the site. For every k requests, it recomputes the caching candidates by the described method, and recreates VD caches within the given cache space. Each time a new request comes to the site manager from the install server, the site manager queries the status of caches to the cache manager. If the cache manager holds a VD cache that contains a subset of the packages being installed, it transfers the cache to physical nodes where the virtual cluster is going to be hosted unless the

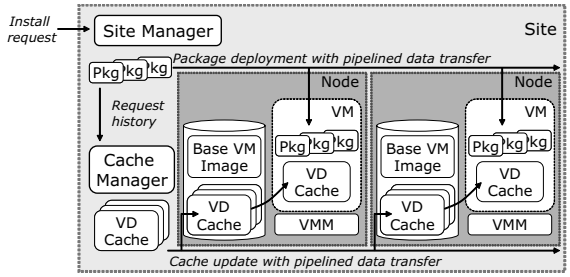

Figure 3. Intra-site installation flow.

```
# hardware specification
[Hardware]
NumberOfNodes: 32
CPUArch: x86
CPUSpeed: 3GHz
RAM: 2GB
Disk: 8GB
# software specification
[Software]
User: root:$1$abcdefgh$rV6Rh...
User: john:$1$abcdefgh$6DEi2...
Hostname: vc%02d
Network: 192.168.10.128/25
Packages: gcc mpich python ...
```

Figure 4. A sample install request to create a 32-node virtual cluster.

nodes already have the same cache. If the cache manager holds multiple such caches, it selects the largest one so as to maximize the installation time reduction.

The packages being installed are fetched from the nearest package repository. In the case of using a VD cache, the site manager only fetches the packages that are not contained in the cache. The fetched packages are transferred to each node with fault tolerant pipelined data transfer so that the transfer time is nearly O(1) with respect to the number of VMs. After the VD cache and remaining packages are made available on each node, the new VM mounts the cache and use the employed cluster installer to install the remaining packages to the mounted VD cache and to configure the installed system.

4 Prototype Implementation

To evaluate the proposed installation technique, we have developed a prototype installer using Xen v3.0.2-2 [2] as a VMM and Lucie v0.0.5 [16] as a cluster installer. Although the prototype currently lacks some functionalities such as resource monitoring, it allows the user to create a virtual cluster in a fast, scalable manner with caching and pipelined data distribution. In this section, we briefly introduce the cluster installer, and describe the implementation details of install requests, VD caches, and each step in the installation process. In principle, installers that allow fine-grained specification of automatic installation of multiple packages to constitute full Linux installations, such as Rocks [14], can be used.

4.1 Lucie Cluster Installer

Lucie is a cluster installer for the Debian Linux distribution, which allows parallel installation of a cluster of PCs. It uses the standard package management tools, `dpkg` and `apt`, and provides an extended packaging mechanism called *meta packages*, which is designed for simplifying cluster-specific configuration of packages being installed [16].

4.2 Install Request

An install request includes the hardware and software specifications. The former is independent from a specific cluster installer being used, but the latter is structured to be given to the cluster installer. Figure 4 illustrates a sample install request that specifies a virtual cluster of 32 nodes, each of which runs on an x86 CPU of 3GHz or faster with more than 2GB of RAM and more than 8GB of virtual disk. The software specification part (written in a format specific to Lucie in our prototype), describes user accounts, host names, network address ranges, and package names.

4.3 VD Caches

The cache manager at each site runs the cluster analysis on the user request history periodically in a user-specifiable interval, and creates VD caches for clusters selected by the method described in Section 3.2.3. It starts a VM and creates a virtual disk for the packages to be installed. Next, it obtains and installs the packages in the package cluster to the virtual disk using `apt` package manager. The virtual disk is then saved as a VD cache, which is a standard virtual disk image file in Xen. Installation of a VM on each node mounts the VD cache and installs the other remaining packages onto the mounted VD cache.

4.4 Installation Steps

VM Booting with Installer Kernel To boot each VM with the installer kernel of Lucie, we create a suspended VM image started with the installer kernel and paused at a state just prior to starting the installation process. The suspended VM is kept in memory when the node is idle or saved to disk when in use. In this way, we reduce the boot time of the installer kernel. To start installation to a VM on the node, we resume from a copied image of the suspended VM, change the memory amount as specified in the user request, and continue the installation steps described below.

Package Installation An install request written by the user is sent to the install server. The install server allocates matching physical nodes to the request, and sends the software specification to the cluster installer. The cluster installer delegates installation of each VM to its site manager. The site manager looks for a VD cache for the request as described in Section 3.3, and, if found, distributes the cache to each physical node using a fault tolerant, pipelined data transfer tool called Dolly+ [13]. [1] The remaining packages are downloaded from the nearest package repository by the site manager via HTTP proxy cache. The downloaded packages are also distributed to each node using Dolly+.

Final Configuration Upon finishing installation of each VM, the cluster installer performs final configuration to the new virtual cluster as specified in the user request, and restarts each VM to boot with the installed system.

5 Experimental Evaluation

To evaluate the effectiveness of VD caching and the scalability with increasing number of VMs, we conduct two performance studies using the prototype installer. In these studies, we only evaluate performance to create single-site virtual clusters; evaluation of multi-site installations is left as future work, but in principle, with sufficiently fast WAN, the results should not be greatly different.

Because we do not have real user request data at hand, we generate synthetic request data as follows. First, based on the assumption that typical users would run their application using some cluster middleware, we select Condor [12] and MPICH [8] as sample middleware, and bioinformatics and numerical computing tools as sample applications. The packages for bioinformatics and numerical computing tools are determined based on the included packages in the Bio and Numeric Rolls in the Rocks cluster installer [14]. Next, with these four package sets, we create four combinations as {Condor, Bio}, {Condor, Numeric}, {MPICH, Bio}, and {MPICH, Numeric}. Finally, to add slight variations to the requests, we select ten auxiliary packages that are not related to any of the four middleware and application packages. We create ${}_{10}C_2$ pairs from the auxiliary packages, and combine the four pairs of middleware and application packages and the 45 pairs of the auxiliary ones, resulting in 180 different package combinations in total. We randomly select one of the 180 combinations for each installation in the following performance studies.

As an experimental environment, we use the Presto III cluster in our laboratory consisting of 256 nodes, each of which embodies a dual AMD Opteron 242 or 250 with 1GB of RAM for Dom0 and 256MB for DomU, and a hard disk drive of IDE or SATA, depending on the node. Each node runs Linux kernel v2.6.16 with Xen v3.0.2-2 patch applied. The site manager runs on a dual AMD Athlon 2000+ node with 1GB of RAM, running Linux kernel v2.6.12.6. All the nodes including the cluster nodes and the site manager node are connected to 13 Gigabit Ethernet switches, which are then connected to two Gigabit Ethernet switches with four gigabit links. These two gigabit switches are interconnected with eight gigabit links.

[1] We have also considered using the broadcast implementation of MPICH, which also works in a pipelined fashion; however, our preliminary experiments with it exhibited significant fluctuation of transfer time. Thus, this paper only reports the implementation and evaluation using Dolly+.

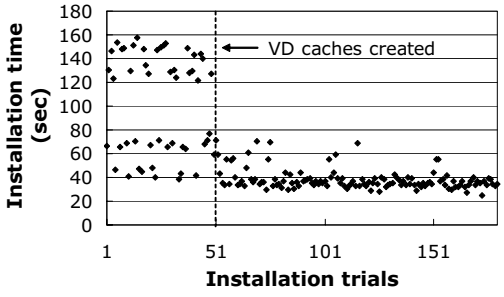

Figure 5. Installation time onto a 67-VM virtual cluster.

5.1 Installation Time Reduction by VD Caching

To evaluate the effectiveness of VD caching, we perform installations 180 times using randomly-chosen requests from the test data set. After every 50 installations, the cache manager recomputes the cluster analysis of the request history, and recreates appropriate VD caches within the given cache space of 5GB, which takes 1472 seconds in average. Since VD caches can be created in the background, we do not include the time to create the caches in the following comparison.

Figure 5 shows the time to install a 67-VM virtual cluster. Before the first cache creation point, the installation time ranges from 38 seconds to 157 seconds. At each cache creation point, eleven to twelve caches are created within the given cache space of 5GB, resulting in 17 different caches in total of the three cache creation points. The size of each cache instance ranges from 189MB to 684MB, 476MB in average. With the VD caches, the average installation time is decreased from 70 seconds to 25 seconds.

Figure 6 compares, for one of the test request set, the breakdown of installation time with and without its VD cache. The average time while no cache is populated is 102 seconds, whereas that of remaining installations is 38 seconds, resulting in a 2.7-times speedup. After VD caches

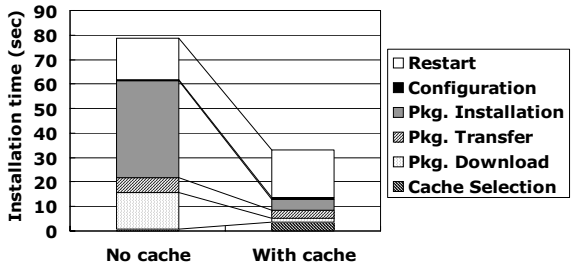

Figure 6. Installation time breakdown.

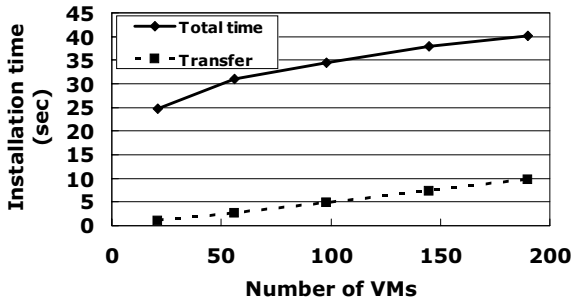

Figure 7. Installation time of virtual clusters from 21 VMs to 190 VMs.

are populated, several installations take over about 60 seconds due to VD cache transfer; other installations reuse VD caches already transferred to each node.

5.2 Scalability

To evaluate the scalability with increasing number of VMs, we conducted installations varying the number of VMs from 21 to 190, under the same configuration as Figure 6. Figure 7 shows that the installation time increased from 24.8 seconds on 21 VMs to 40.0 seconds on 190 VMs. Principal part of the linear increase is due to the packages transfer time, shown as the dotted line in the graph. We observe that the increase, despite of the pipelined transfer technique, is because of some fixed implementation overhead in Dolly. Note, however, that the current results do indicate that the installation overhead per 100 VMs is 9 seconds, resulting in installation time of less than 2 minutes for 1000 VMs.

6 Related Work

The focus of previous efforts in providing user-specific computing environments on clusters and Grids include interoperability with existing Grid services [7, 10], configuration management [11, 14], and efficient environment deployment [3, 11]. In this section, we relate our contributions to the past projects as well as several other instances of user-specific computing environments [3–5].

In the current Grid environments, there are already a number of established services for security, accounting, and resource scheduling, typically realized with the Globus toolkit [6]. Thus, interoperability with such existing services is essential to apply virtualization technologies in real Grid environments. Keahey et al. proposed the workspace service that defines secure interfaces with existing services in the context of the Globus toolkit [10]. Foster et al. presented a virtual cluster environment by aggregating multiple workspaces [7]. Note that, although our current framework lacks such interoperability, their work is orthogonal to ours; we plan to implement our efficient and flexible virtual cluster installation in such a context.

Past projects on configuration and customization of computing environments for clusters and Grids include the work of Krsul at al. [11] and Papadopoulos et al. [14]. Krsul et al. proposed a VM creation and customization framework called VMPlant [11] for virtual Grid environments such as In-VIGO [1]. It provides a graph-based configuration interface to the user, which encodes each configuration as a node and dependency between each configuration as directed edges. The package-based system installation in our framework is similar to their graph-based configuration; In essence, the package-dependency structure can be represented as a DAG. The differences between our approach and theirs lie in two points. First, we exploit an existing infrastructure for cluster configuration management and extend it for virtual clusters, whereas they rely on their own custom configuration schema. Therefore, while our framework requires the user to prepare only declarative configuration files in most scenarios, they require the user and the resource provider to provide actual implementation, typically written in scripts, of such configurations as well. Second, they do not allow the user to create VMs on the fly; rather they assume offline-created VM images available for every user request. Such assumption is unlikely to be held in multi-organization, multi-site, heterogeneous Grid environments. Papadopoulos et al. presented their cluster installation and management tool called Rocks in [14]. Rocks builds on a popular Linux distribution and its package management system, and provides a cluster-specific configuration mechanism through extended packages called Rolls. Although we use Lucie in the current prototype, we see no limiting barrier to support Rocks as well.

Efficiency and scalability of installation has been another important topic in providing user-specific environments since the scale of underlying physical clusters has continued to increase. VMPlants support reuse of partially configured VMs through partial graph matching [11]. Such

reuse is similar to our VD caching, which allows reuse of virtual disks where frequently-requested packages are installed. Although both techniques aim to reduce system deployment time, our advantage is that we automatically create and destroy such partial images with the statistical cluster analysis. Similar to our approach, system deployment in Grid'5000 uses pipelined data transfer for scalability [3].

Examples of user-specific computing environments include Grid'5000 [3], COD projects [4], and Moab Workload Manager [5]. Grid'5000, built on a large collection of computing clusters distributed across WANs, provides an on-demand physical testbed for Grid computing research [3]. COD aims to consolidate multiple clusters on a single physical cluster by dynamically reinstalls and reconfigures cluster nodes. Although their contexts and ours differ since they install and configure physical resources, our system installation techniques could be effective. A batch scheduler called Moab Workload Manager allows to use VMs on each compute node as a schedulable entity. Our on-the-fly VM creation would significantly simplify the management of VM images in such VM-based job execution environments.

7 Conclusion

We proposed a new virtual cluster installation technique that allows user-specific customization of VMs via the use of cluster configuration tools, while achieving rapid installation turn-around by eliminating much of the overhead associated with installation of VMs. Experimental studies using a prototype implementation show that installation of a 190-node virtual cluster can be done in 40 seconds, indicating that installation of a 1000-VM cluster could be done in less than two minutes.

Several issues are left as future work: further tuning of installation time, interoperability with standard Grid services, security, resource scheduling, and evaluation of multi-site installation. We also plan to work on modeling of installation time, which is necessary for accurate resource accounting [15] and more efficient installation.

Acknowledgments

This work is supported in part by the Ministry of Education, Culture, Sports, Science, and Technology, Grant-in-Aid for Scientific Research on Priority Areas, 18049028, 2006.

References

[1] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From virtualized resources to virtual computing grids: the in-vigo system. *Journal of Future Generation Computing Systems*, 21(6):896–909, 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, Bolton Landing, 2003.

[3] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard. Grid'5000: a large scale, reconfigurable, controllable and monitorable Grid platform. In *International Workshop on Grid Computing*, Seattle, USA, Nov 2005.

[4] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 90, 2003.

[5] Cluster Resources, Inc. Moab Workload Manager administrator's guide. http://www.clusterresources.com/products/mwm/docs/.

[6] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, LNCS, pages 2–13, 2005.

[7] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. In *CCGRID '06*, pages 513–520, Singapore, May 2006.

[8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sep 1996.

[9] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.

[10] K. Keahey, I. Foster, , T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In *Euro-Par*, pages 421–431, 2005.

[11] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, pages 7–18, Pittsburgh, PA, November 2004.

[12] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[13] A. Manabe. Disk cloning program 'dolly+' for system management of pc linux cluster. In *Computing in High Energy Physics and Nuclear Physics*, 2001.

[14] P. M. Papadopoulos, M. J. Katz, and G. Bruno. Npaci rocks: Tools and techniques for easily deploying manageable linux clusters. In *Proceedings of the International Conference on Cluster Computing*, 2001.

[15] B. Sotomayor, K. Keahey, and I. Foster. Overhead matters: A model for virtual resource management. In *First International Workshop on Virtualization Technology in Distributed Computing*, Nov 2006.

[16] Y. Takamiya. *Large-Scale Configuration Management and Installation of Commodity Clusters*. PhD thesis, Tokyo Institute of Technology, March 2006.

[17] R. Wolski, N. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.