An Efficient Resource Management System for On-line Virtual Cluster Provision

Yang Chen
School of Computer Science and Engineering, Beihang University, Beijing, China
chenyang@act.buaa.edu.cn

Tianyu Wo
School of Computer Science and Engineering, Beihang University, Beijing, China
woty@act.buaa.edu.cn

Jianxin Li
School of Computer Science and Engineering, Beihang University, Beijing, China
lijx@act.buaa.edu.cn

Abstract

As a prevalent paradigm for flexible, scalable and on-demand provisions of computing services, Cloud Computing can be an alternative platform for scientific computing. In this paper, we propose an efficient resource management system for on-line virtual clusters provision, aiming to provide immediately-available virtual clusters for academic users. Particularly, we investigated two crucial problems: efficient VM image management and intelligent resource mapping, either of them has remarkable impact on the performance of the system. VM image management includes image preparation and local image management on physical resources. A resource mapping refers to a mapping from user's resource constraints to specific physical resources. We explore how to simplify VM image management and reduce image preparation overhead by the multicast file transferring and image caching/reusing. Additionally, the Load-Aware Mapping, a novel resource mapping strategy, is proposed in order to further reduce deploying overhead and make efficient use of resources. The strategy takes account of both image cache and VM load distribution information. System evaluation is conducted through various real stress workloads, and results show that our approaches are effective comparing to other common solutions.

Keywords: *Cloud computing, Virtual Cluster, Resource Mapping, VM Image Management*

1. Introduction

As the combination and evolution of distributed computing, parallel computing and utility computing, Cloud technologies provide a new paradigm for flexible, scalable and on-demand provisions of computing services and applications over pervasive networks (e.g., Amazon EC2 [1], Google AppEngine [3], and Eucalyptus [2]). As a fast maturing technology, the Virtual Machine (VM) technology (e.g., Xen [4], KVM [5]) plays as an important role in Cloud Computing. By multiplexing hardware resources and encapsulating software into image files, VM technology provides the capability of deploying isolated and customized runtime environments over distributed and heterogeneous physical resources.

Our previous work, iVIC platform [6] aims to provide a flexible access to virtual cluster computing environments on the top of common resources, for academic researchers. A virtual cluster is composed of a number of VMs interconnected by a virtual or physical network over underlying physical resources; and each VM acts as a virtual node of a computing cluster. Many efforts have been taken in contexts of Grid Computing and HPC [21, 20] towards dynamic creation of virtual cluster environments over geographical distributed resources. In these work [22, 15], some implementation issues have been tackled for realizing more efficient resource sharing. However, there are two crucial problems: VM image management and physical resource mapping, proposing the new challenges in the context of Cloud Computing, which thinks highly of user interactive computing model and resource simultaneous sharing, comparing to Grid Computing [18].

For our practical reasons, we argue that on-line virtual cluster provision is extremely challenging:

On-line provision request and flexible user resource demand. On-line property makes efficient provisions more difficult to realize, due to little awareness of arrivals and various demands of future requests. The other aspect of on-line property requires that the system should accomplish deploying process immediately. Moreover, it requires that the system must perform a well-defined admission control on new incoming requests to avoid violating resources for existing virtual clusters, as the resources are limited.

Unavoidable image preparation. A VM disk image is pivotal for a VM correctly running, as it encapsulates the OS file system and software into a single file, which are the foundation for a VM. Every time a VM to start, the image should be prepared on the hosting machine. Image preparation is unavoidable during a virtual cluster deploying process, and potentially time consuming. The overhead of it potentially occupies a great proportion of the whole deploying overhead.

Dynamic resource utilization. Since virtual clusters dynamically arrive and depart from the platform; and many users simultaneously share the limited physical resources, the system is required to be aware of dynamic utilizations of underlying resources.

In this paper, we present the design and implementation of a resource management system, named

978-0-7695-3840-2/09 $26.00 © 2009 IEEE
DOI 10.1109/CLOUD.2009.64

Sulcata, for on-line virtual cluster provision. In particular, we investigate how to simplify VM image management and reduce image preparation overhead through the multicast file transferring and image caching/reusing approach. In addition, for further reducing the deploying overhead and making efficient use of resources, we present a novel resource mapping strategy, named *Load-Aware Mapping*, which takes into account both of image cache and VM load distribution information. The results of evaluations conducted through various real stress workloads quantify the benefits of our approaches, and show that they are effective comparing to other common solutions.

The remainder of this paper is organized as follows. In Section 2, we discuss the on-line virtual cluster provision scenario, and VM image management. In Section 3, we elaborate the resource mapping problem and our Load-Aware Mapping strategy. The design and implementation overview of Sulcata is described in Section 4. The performance evaluations are discussed in Section 5. Section 6 describes related work in the areas of Cloud Computing and Grid Computing. Finally, Section 7 presents the paper's conclusions.

2. On-line Provision Scenario and Image Management

Our previous work iVIC provides a platform for academic researchers to dynamically create customized virtual computing environments to launch various scientific computing, simulations and analysis, by leveraging VM technology. As depicted in Fig. 1, through a browser-based interface, users are facilitated with customizing, deploying and monitoring their owner virtual computing environments.

In iVIC, common resources (e.g., a set of workstations, PC servers and small clusters) are organized into a number of physical resource pools. Each physical machine is treated as a VM Container (VMC), which is responsible for providing VM environments (currently, it supports KVM/Qemu [5]). Each VMC exposes both controlling and querying interfaces to upper level resource manager via a collection of SOAP interfaces. The iVIC requires a special storage component – the *Image Repository* for storing all VM disk images. Before creating virtual nodes, the system should prepare the appropriate image onto corresponding VMCs, i.e. transferring the appropriate image from the repository to individual VMCs. After a successful deploying, the user can use a VNC console or SSH client to login, perform further configuration, and launch computing tasks as needed. More detailed information about iVIC platform can be found in [6].

Once receiving a user's provision request, the system needs to precisely deploy a virtual cluster. The process of a virtual cluster deploying generally can be decomposed into three steps:

Step 1: VMs image preparation. The local repository delivers the appropriate image to individual VMCs.

Step 2: VMs creation and configuration. VMCs create VMs using the prepared image and then perform necessary configurations for each VM.

Step 3: VMs reboot. Each VMC reboots hosted VMs, after finishing configurations.

To investigate influences of each step on the deploying overhead, we conducted a number of independent virtual cluster deploying experiments, in which the number of virtual nodes varied from 4 to 40 with a step 2, and each virtual cluster randomly used one image from a set of images in different size. The VM image was currently downloaded by VMCs from a NFS server, acting as a central image repository. We examined the deploying overhead curve with relation to the number of clients and the size of images. We observed similar results illustrated by [12]: the time taken by *Step 1* accounts for a significant proportion of the whole deploying overhead; and it violently increases as either the size of image or the number of virtual nodes increases; whereas, the overhead in *Step 3* can be omitted compared to the other two steps. Further examination reveals that the unicast file transferring based on C/S model (e.g., NFS, HTTP) is not suitable for image transferring in our scenario. This is because the model potentially causes network bandwidth full-filled with redundant image segments and great burden on the end server introduced by numerous concurrent clients.

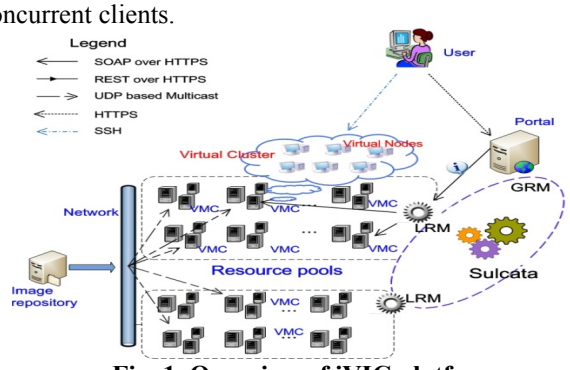


Fig. 1. Overview of iVIC platform

2.1. Image Multicast transferring

The most common way for managing images is to locally store all the VM images on individual VMCs, and make VMs access to individual copies of those images. Obviously, both the limited local disk space and the substantial cost of managing each image copy make it impractical. Another common approach [12] is to use a central NSF server as a repository, and make each VM read-only share images. However, NFS essentially falls

into C/S model, which prohibitively causes repository server being a bottleneck. Furthermore, we observed that unicast image transferring approach (e.g., copying images from a central NFS server) for downloading and locally using images suffers constraints of network bandwidth and bottleneck of repository server.

In order to efficiently use network bandwidth and avoid potential performance bottleneck on the repository, we conduct VM image transferring based on multicasting file transferring (UDPcast [8] integrated). Although UDP based multicast file transferring suffers the packet loss issue in some cases, our prototype testbed shows that multicast VM image transferring within dedicated LAN can achieve great performance over other methods, such as NFS, HTTP, FTP and so on.

2.2 Local Image Management

The other challenge is local image management on a single VMC, as inappropriate local images management may introduce additional image preparation overhead. For example, a common way is to allow each VM to use an own copy of the local image, and have fully read/write access to it. However the expense of maintaining image copies would be enormous, as each VMC may host multiple VMs.

In our work, we exploit image caching plus COW reusing (QCOW2 [10]) for local image management. It means that each VMC maintains a local image cache (using the LRU strategy); and allows VMs to reuse cached images based on COW. COW image reusing eliminates image locally copying overhead and the issues introduced by image caching scheme [12]: filesystem corruption, image fill. Furthermore, the image caching/reusing approach helps to avoid redundant image transferring and maintenance for image copies. The cache information can be used as heuristic information for resource mapping strategies, for realizing an intelligent resource mapping.

3. Efficient Resource Mapping

3.1. Resource mapping problem

In our scenario, what users are mainly concerned about are the deploying overhead and the performance of provided virtual clusters. While, from the resource-oriented point of view, a natural goal would be to maximize the resource utilizations for supporting as many provision requests as possible, at the premise of ensuring user acceptable level of QoS. Thus, how to effectively balance the expectations from both sides is an essential of which a mapping strategy takes care.

In fact, a resource mapping problem can be defined as a set mapping problem, which can be converted into a nonlinear optimization problem under some constraints, whereas it is difficult to find an optimal mapping result of the problem, especially when the scale of resources is large. From another perspective, as the providers of virtual computing services, our natural objective is to maximize the *service profit* that obtained from virtual computing services. In fact, the *service profit* can be defined in various ways according to different models. For simplicity, we in this paper describe the *service profit* by the accumulated number of virtual nodes from deployed virtual cluster, over a certain period of time. In addition, for a provision request, we define the sum of virtual nodes as its *predicted service profit*.

3.2. Resource mapping strategy

To ensure the performance of virtual clusters, the strategy would not like to distribute virtual nodes among too many VMCs. For example, one node per VMC mapping would be the worst case for applications dependent on substantial parallel network communications, due to the significant network latency. As mentioned above, a resource mapping strategy can take advantage of the cached images as heuristic information. In this paper, we propose a resource mapping strategy, named *Load-Aware Mapping* that takes into account image cache and VM load distribution information, aiming to make efficient use of physical resource and reduce deploying overhead.

We denote the maximum VM load on a VMC as *Overlapping* (i.e., the maximum VM load can run simultaneously on a VMC). In order to better describe VM load and shrink searching space, the *Load-Aware Mapping* labels the state of VMC load as three kinds: low, heavy and moderate. Meanwhile, it defines two thresholds $thres_{low}$ and $thres_{high}$: if the load of a VMC exceeds the threshold $thres_{high}$, it considers the VMC as heavy loaded, and takes care of its load increment when mapping virtual nodes onto it; correspondingly, a VMC is considered as low loaded and can be made good use of, if its load is below the threshold $thres_{low}$. Obviously, the machine with load between two thresholds is considered as moderate loaded.

We define the values of $thres_{low}$ and $thres_{high}$ as following: $thres_{low} = \alpha \times Overlapping$; $thres_{high} = \beta \times Overlapping$, where the two coefficients, α and β, $0 < \alpha <= 0.5 <= \beta < 1$. Moreover, in order to shrink searching space, the strategy maintains three queues according to three kinds of load states: *queue_low*, *queue_moderate* and *queue_high*; and holds a reference to each VMC in one queue according to VMC's load.

The strategy always prefers a provision request with the maximum *predicted service profit*. Once preferentially

selecting a request, it starts the workflow of making a resource mapping, which is depicted as follows:

Step 1 Scan *queue_low* and preferentially choose a VMC with cached image; for a chosen VMC, assign virtual nodes to it until predicated VM load equals $thres_{low}$. If all the VMCs in the queue cannot take all virtual nodes, go next step with un-mapped virtual nodes.

Step 2 Similar with *Step 1*, scan *queue_moderate* prefer the VMC with the image cached, whereas assign virtual nodes to a VMC until its predicated VM load has an $(thres_{high} - thres_{low})/2$ increment on the basis of the current load. If all VMCs in the queue cannot take all virtual nodes, go next step with un-mapped virtual nodes.

Step 3 Similar with *Step 1*, scan *queue_high* prefer the VMC with the image cached, whereas assign one virtual node per machine.

As we see, the strategy tries to acquire a better profit by preferentially selecting a request with the maximum predicted profit; and tries to avoid redundant image transferring by preferentially selecting a machine with cached image. Moreover, the strategy assigns VM loads to a VMC according to its load state: a VMC with low load would be assigned more virtual nodes; while fewer nodes would be assigned to a heavy loaded VMC. Distinguishing the load distributions among VMCs can achieve a better performance assurance for virtual clusters, as virtual nodes can be deployed with a high probability onto low loaded VMC, which have more available resources. We argue that the *Load-Aware Mapping* can be an efficient resource mapping strategy applicable for other projects providing on-line virtual computing environments like Enomaly's ECP [7] and Eucalyptus [2].

For comparisons, we also examined the other two common used resource mapping strategies: the *Random Mapping* and the *Greedy Mapping*. Due to space limitations, we briefly describe the two strategies. Unlike the *Load-Aware Mapping*, the two strategies aren't aware of image cache and VM load distribution information. The *Random Mapping* randomly maps each virtual node to physical machines, which results in the virtual node randomly assigned onto physical machines. While, the *Greedy Mapping* prefers a request with the maximum predicted service profit, similar with the *Load-Aware Mapping*; and it prefers the VMC that has maximum available resources, and greedily assigns virtual nodes just until its predicted load equals the threshold Overlapping. Two strategies can be thought as two extremes: the *Random Mapping* tries to uniformly distribute load among physical machines for balancing resource utilizations; while the *Greedy Mapping* tries to gain the maximum service profit from preferential request selections and greedily virtual nodes assignments.

4. Designing and Implementation

Motivated by the primary goals: user-transparence, scalability, as well as modularity, we adopt a hierarchical architecture design for Sulcata. The two hierarchical levels of Sulcata are depicted in Fig. 1. Two level of resource management is carried out by the Global Resource Manager (GRM) and Local Resource Manager (LRM), respectively. In this paper, we focus on the components related to the functionality of on-line virtual cluster provision.

The GRM manages underlying resources in granularity of a resource pool. GRM is responsible for processing incoming provision requests, and maintaining a local representation of state information about each resource pool. For a provision request, the GRM tries to allocate resources to the request at the global-level. It accounts for required resources of the request and then makes a resource pool selection, according to the information of each resource pool and the geographic location of the user. Once selecting a resource pool, the GRM delegates a scheduling request attached with resource constraints to the LRM. Additionally, the GRM periodically synchronizes VM image list with the image repository. In this way, the lower level LRMs are provided with a consistent view of VM image list, instead of individually maintaining redundant lists of available images.

LRMs are the core components of Sulcata, as they take over most of resource management workloads. The main structure of a LRM is depicted in Fig. 2. The *Resource Monitor* module keeps track of resource usage and image cache list of each VMC. It periodically synchronizes local representation of resource information with each VMC and actively updates internal resource data structures once a virtual cluster has been successfully deployed, or destroyed. The crucial responsibility of the *Resource Scheduler* (the *scheduler*) module is scheduling resources for scheduling requests, i.e., mapping virtual nodes into physical machines of local resource pool, and conducting the workflow for a virtual cluster deploying. Whenever receiving a scheduling request, a LRM inserts the request into the inner scheduling queue. The *scheduler* periodically selects a request from the queue according to the given strategy. For a selected request, the *scheduler* makes a resource mapping from virtual nodes to VMCs, according to the *Load-Aware Mapping* strategy. Then, the LRM starts the workflow for a virtual cluster deploying, according to the result of resource mapping.

As illustrated in Fig. 2, there are the other two execution modules: the *Deployment Execution* and the *Image Transfer*, which are responsible for carrying out deploying operations dictated by the schedule. The *Image Transfer* takes over the operation that dictating local image repository to transfer specified image to individual VMCs; while the *Deployment Execution* carries out operations that deploying VMs into corresponding VMCs. For example, once getting a

resource mapping result, the *Image Transfer* dictates corresponding VMCs to locate and connect to the local repository (in fact, VMCs join in the multicast group), and then it indicates the local repository to start multicast image transferring.

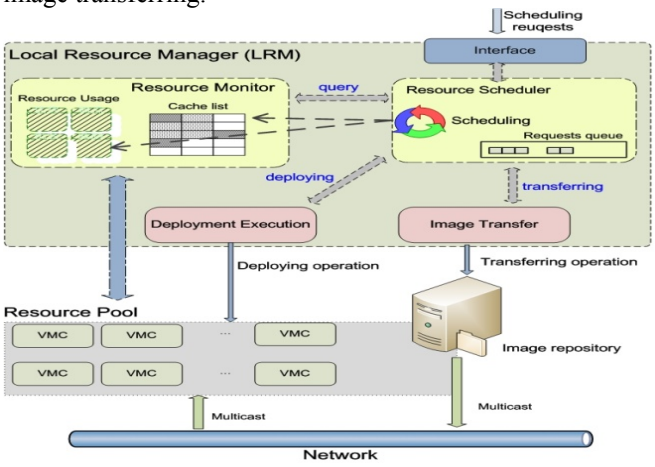


Fig. 2. Structure of a LRM

5. Performance evaluation

5.1. Evaluation Design

The emphasis of performance evaluation is focused on image caching/reusing and our resource mapping strategy, both of which reside in the LRM. Thus, we focus the extensions for corresponding modules of the LRM. Meanwhile, we omit the performance evaluation of multicast image transferring considering both space limitations and our preliminary experiments for image transferring described in Section 2.

We extended the *Resource Scheduler* module in the LRM for implementing the other two strategies. We also implemented the other local image management approach, denoted as *non-cache/unreusing* (we denote our image management approach as *caching/reusing*), which means that each virtual cluster deployment requires image transferring; and each virtual node has fully read/write access to an independent local image copy. For this approach, we omitted the overhead of local image copying (it's impossible in real scenarios), as this overhead makes the whole deploying overhead worse compared to *caching/reusing*. Thus, the results of *non-cache/unreusing* can be considered as an optimized situation. In order to accurately trace mapping results and account for deploying overhead for each request, we plugged two logging modules into two execution modules in the LRM. In addition, we replaced receiving requests from iVIC portal with reading them from a workload log, in order to use real stress workloads.

5.2. Evaluation environment

According to our prototype testbed, we modeled a simulated resource pool which is comprised of 30 VMCs, each with a Core2 2.00GHz CPU, 2GB RAM, and 40GB hard disk partition, a Gigabit Ethernet switch network connection. Based on our preliminary experiments, we conservatively set the maximum VM load at a VMC 28 units (i.e., the attribute *Overlapping*), and the bandwidth of the image transferring network be 30 MB/s. In addition, we assumed that one configuration overhead for a virtual node took 5 seconds, and the configurations for two VMs hosted on a VMC should be performed on after another, while the configurations for two distributed VMs could be performed simultaneously. We calculate the configuration overhead of a virtual cluster instance by multiplying the maximum overlapping value of its virtual nodes and the configuration overhead per node. We also assume that the network used for image transferring is independent of the network for applications on the virtual clusters and it has a dedicated bandwidth B_d. For a VM image in the size of S_{img}, the multicast transferring overhead is assumed to be S_{img} / B_d. In future work, we will relax these assumptions by replacing them with more accurate image transferring overhead related to network bandwidth of repository, the number of virtual nodes based on our measurement profiles.

We chose three typical workload logs as our stress workloads (available at the Parallel Workloads Archive [23]): two stretches of Lawrence Livermore National Lab Thunder (*LLNL-Thunder*), one (*LLNL-Thunder-518*) from submission 496 to 826, 305 valid submissions with average 39 seconds running time, the other (*LLNL-Thunder-60758*) from submission 60758 to 62148, 1249 valid submissions with average 810 seconds running time; one stretch of High-Performance Computing Center North (*HPC2N-60*) from submission 60 to 3965 (about 31 days), 1591 valid submissions with average 8746 seconds (about 2.4 hours). The two workloads from *LLNL-Thunder* are considered as scenarios in which short-lived computing requests account for the majority. On the other hand, the workload *HPC2N-60* is considered as a scenario in which survival durations of each request are erratic, mixing with long and short living requests.

We dropped an entry from workload logs with the running time unknown, and considered it as an invalid submission; while we translated a valid entry as a virtual cluster provision request as following: the number and the memory size of virtual nodes was equal to the number of requested processors and the size of requested memory respectively; we set the arrival time and survival duration of a virtual cluster as the submitting time and running time of the entry. We used 30 VM images with the size distributed uniformly between 1024 MB and 2048 MB; furthermore, we added VM image reference into each provision request according to two typical distributions:

Pareto principle (or 80/20 Rule): add one of images to a request according to the distribution in which 80% requests uniformly referred 6 images (20%), and the remaining 20% requests uniformly referred 24 images (80%).

Uniform distribution: add one of images to a request randomly according to uniform distribution.

Table 1. Evaluation cases

random_unreuse	*Random Mapping* strategy, and *non-cache/unreusing* approach.
random_reuse	*Random Mapping* strategy, and *caching/reusing* approach.
load-aware_reuse	*Load-Aware Mapping* strategy, and *caching/reusing* approach.
greedy_unreuse	*Greedy Mapping* strategy, and *non-cache/unreusing* approach.
greedy_reuse	*Greedy Mapping* strategy, and *caching/reusing* approach.

In the following of paper, each workload is named with prefix *pareto-* or *random-*, representing the image distribution combined with it. In order to minimize the random error, we conducted 15 times evaluations for the Random Mapping with each workload, and adopted the average values. Meanwhile, for the *Load-Aware Mapping*, we set two coefficients α, β be 0.3, 0.8 respectively, unless otherwise specified. We conducted evaluation cases for each workload as described in Table 1.

5.3. Evaluation results

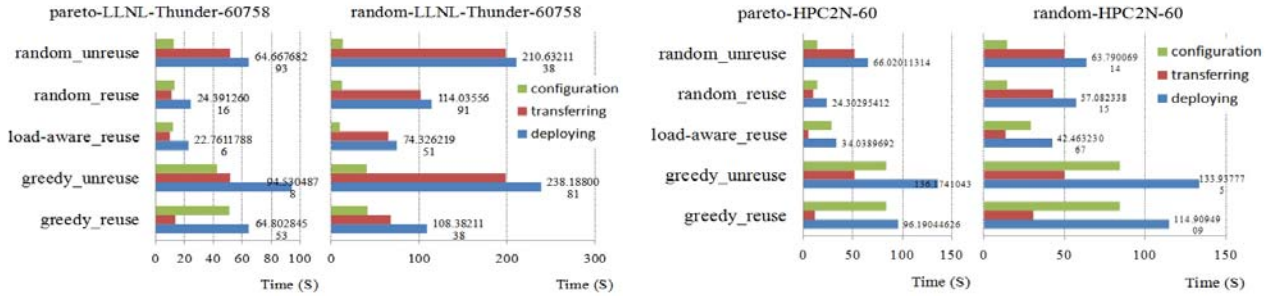
5.3.1 Deploying overhead. In Fig. 3, we show quantitative comparisons of three substantial overheads during a virtual cluster deploying process: configuration, image transferring, and the whole deploying overhead. The Fig. 3.a and Fig. 3.b respectively show the overheads in each case under *LLN-Thunder-60758* and *HPC2N-60* workload combining with two image distributions. It is worth noting that the cases that exploit image *caching/reusing* approach have evident advantage over *non-caching/unreusing* cases. Both in Fig. 3.a and Fig. 3.b, we observe that the *Load-Aware Mapping* achieves an enhancement in reducing deploying overhead over other strategies under each workload. We attribute this enhancement of performance to its awareness of image cache information. On the contrary, although both *greedy_reuse* and *random_reuse* cases exploit the image *caching/reusing* approach, they are not aware of the cache information. We argue that this unawareness results in the probability of "cache fit" decreasing and reaching to worst-case when image references distribute uniformly. Further examination shows that the average configuration overhead for both *greedy_reuse* and *greedy_unreuse* cases are larger than other cases. We attribute this observation to the greedily assigning behavior of the *Greedy Mapping* strategy, which may result in that the virtual nodes from a virtual cluster are aggregated on a few VMCs (we denote this mapping result as *node clustering mapping*). Since the configurations for virtual nodes from a virtual cluster on a hosting machine should be performed one after another. Thus, the *Greedy Mapping* causes a greater configuration overhead comparing the other two mapping strategies due to its high probability of *node clustering mapping*. Additionally, in *HPC2N-60* workload, the *random_reuse* case has shorter total deploying overhead than the *load-aware_reuse* case. The reason for this result is that the *Load-Aware Mapping* more probably results in *node clustering mapping* comparing to the *Random Mapping*; and within *HPC2N-60* workload, the requests that demand for a large virtual cluster (about *25%* requests require over 32 virtual nodes) account for the majority. These two factors conduce the *Load-Aware Mapping* taking more time for virtual nodes configurations.

5.3.2 Impact from tow coefficients. In order to further examine the impact of two coefficients: α and β, on the effectiveness of the *Load-Aware Mapping*, we arranged 9 additional evaluations for each workload. For comparisons, we also added the results from both the *random_reuse* and *greedy_reuse* cases. Fig. 4 shows comparisons with three substantial overheads under two workloads: pareto-LLNL-60758 and pareto-HPC2N-60, respectively. We denote combinations of two coefficients using the notion *α/X_β/Y* (e.g., α/3_β/8 represents a combination in which α=0.3 and β=0.8).

We observe that different combinations of α and β have a limited effect on the image transferring overhead; while the configuration overhead varies as the gap of α and β increases. For example, under pareto-LLNL-Thunder-60758, when α=0.3, as β varying from 0.7 to 0.9, the transferring overhead keeps nearly stable; while the configuration overhead increases. We attribute this effect to the image cache-awareness and partial *node clustering mapping* of the *Load-Aware Mapping*: as the gap of two thresholds increases, it maps many more virtual nodes (e.g., a $(thres_{high} - thres_{low})/2$ load increment in *Step 2*) to a VMC, which results in much more time taken for configurations of virtual nodes. In fact, we can observe the similar comparison results under other workloads. For space limitations, we do not illustrate the results from other evaluation cases.

5.3.3. Resource utilization. We denoted a VMC as *high_load* if its VM load exceeded 80% of its maximum load threshold, while a VMC as *under_load* if its load was under of 20% of the maximum load threshold. Because of space limitations, we are unable to present results for utilization of hosting machines from all the cases; instead, we focus on the three cases: *load-aware_reuse*, *random_reuse* and *greedy_reuse*, under the

pareto-LLNL-Thunder-60758 workload, which are representative of trends that we observe across all cases.



(a). Workload: LLNL-Thunder-60758 **(b). Workload: HPC2N-60**

Fig. 3. Three substantial overheads: configuration, transferring, and deploying

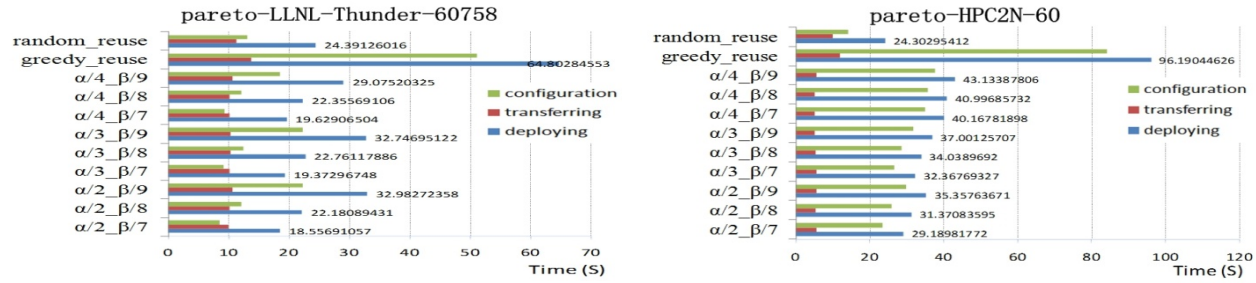


Fig. 4. Effect of two coefficients: α and β

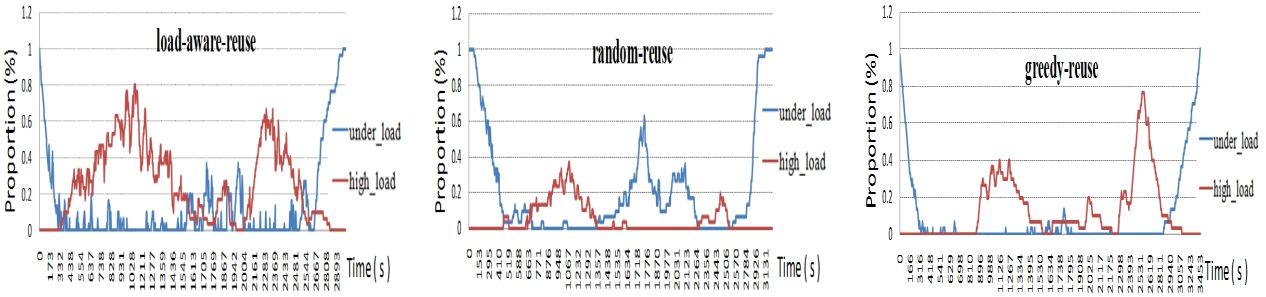


Fig. 5. Resource utilization under workload pareto-LLNL-Thunder-60758

Fig. 5 shows the VMC percentage curves of two load states in three cases. If we look only at the *high_load* curves, the *random_reuse* case performs the worst, due to that it tries to uniformly distribute VM load among all the VMCs; while the *load-aware_reuse* case has an evident advantage, due to its awareness of VM load distribution. On the other hand, if we look only at the *under_load* curves, the *greedy_reuse* case achieves an evident enhancement of eliminating resource under-utilization, since it always prefers choosing a VMC with maximum available resource (i.e., with minimum VM load) as a mapping target; while the *random_reuse* case performs the worst due to the same reason as its *high_load* curve.

6. Related Work

In Grid environments, great efforts [15, 22] have been taken towards standards for precise resource provision and building up virtual computing environments over geographical distributed resources. Most of them focus on interoperability with existing Grid infrastructures. Some projects take into account the overhead for virtual cluster creation. Yamasaki et al. [14] propose a model-based resource selection strategy that intelligently chooses a node combination for building a virtual cluster with an optimized provision overhead. To reduce deploying overhead, they pre-stage the VM images on each hosting machine, and install software packages on the fly into VM images. They also cache software packages on the hosting machines for reducing transferring burden on the central repository. However, this approach would be promising only when its software installation time is short enough compared to image transferring overhead. Although their empirical model based on multiple regressions can accurately predict the overheads of each creation step, the model tightly couples with its system implementation. Emeneker et al. [12] proposed several promising image caching strategies for faster VM image

preparation and discussed deficiencies introduced by the image cache, in their system DVC [9]. However, they only support unicast image transferring over LAN; moreover, their resource mapping strategy does not take into account image cache and load distribution information. Sotomayor et al. [13] proposed a resource provisioning model for virtual computing environments, by leveraging promising features provided by VM technologies, such as suspend, migrate and resume. They presented a resource accounting method by estimating the time to deploy VMs on a single machine. Similar to us, they examined the influence of VM image caching. They focus on the batch execution scenarios in which computing requests contain various time constraints of resource availability; whereas we are concerned about VM load distribution and resource utilization, besides immediately-available time constraints from requests.

7. Conclusion

We have presented our work towards an efficient resource management system for on-line virtual cluster provision. In particular, we focused on two crucial problems: efficient VM image management and intelligent resource mapping. We have proposed how to simplify VM image management and reduce image preparation overhead by multicast file transferring and image caching/reusing. Additionally, we proposed an intelligent resource mapping strategy, named *Load-Aware Mapping*, in order to further reducing deploying overhead and balance resource utilizations. We have conducted system evaluations under various real stress workloads combining with two typical popular distributions of VM image. The evaluation results show that, our VM image management approach can evidently reduce the deploying overhead. The results also show that, our strategy can facilitate further reducing VM image preparation overhead and achieve more balanced resource utilization over the other two common used strategies.

In the future work, further investigations into the behavior of multicast image transferring process based on fine-grained analysis of deploying experiments will be conducted; furthermore, request traces from the iVIC platform will be used for performance evaluations.

8. Acknowledgements

This work is partially supported by grants from China 863 High-tech Program (No. 2007AA01Z120), China 973 Fundamental R&D Program (No. 2005CB321803) and National Natural Science Funds for Distinguished Young Scholar (No. 60525209, 90818028). We would also like to thank members in Network Computing Research team in Institute of Advanced Computing Technology of Beihang University for their helpful suggestions.

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.
[2] D. Nurmi, R. Wolski, et al. Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems. Tech. Rep. 2008-10, University of California, Santa Barbara, October 2008.
[3] Google AppEngine. http://code.google.com/appengine/.
[4] P. Barham, B. Dragovic, et al. Xen and the Art of Virtualization. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, Oct. 2003, NY, USA.
[5] Kernel Based Virtual Machine (KVM). http://www.linux-kvm.org/.
[6] J. Huai, et al. CIVIC: a Hypervisor based Virtual Computing Environment, In International Conference on Parallel Processing Workshops 2007, 10-14 Sept. 2007 Page(s):51 - 51
[7] Enomaly's ECP. http://www.enomaly.com/.
[8] UDPcast. http://udpcast.linux.lu/.
[9] W. Emeneker, D. Jackson, et al. Dynamic Virtual Clustering with Xen and Moab. In Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), 2006.
[10] The QCOW2 Image Format.
http://www.gnome.org/~markmc/qcow-image-format.html.
[11] oVirt. http://ovirt.org/.
[12] W. Emeneker, D. Stanzione. Efficient Virtual Machine Caching in Dynamic Virtual Clusters. In SRMPDS Workshop, ICAPDS 2007 Conference, December 2007.
[13] Sotomayor Borja, et al. Combining Batch Execution and Leasing Using Virtual Machines. HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing. Boston, MA, USA
[14] S. Yamasaki, N. Maruyama, and S. Matsuoka. Model-based resource selection for efficient virtual cluster deployment. In VTDC '07: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, 2007.
[15] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life on the grid. Scientific Programming, 13(4):265–276, 2005.
[16] OpenNebula Project. http://www.opennebula.org/.
[17] M. Rosenblum, T. Garfinkel. Virtual machine monitors: Current technology and future trends. IEEE Computer, 2005,38(5):39−47.
[18] I. Foster, Yong Zhao, I Raicu, S LuCloud.Computing and Grid Computing 360-Degree Compared.GCE '08 In Grid Computing Environments Workshop, 2008..
[19] C. Clark, K. Fraser, S. Hand, J. G. Hanseny, E. July,C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In NSDI, May 2005.
[20] F. Berman, G. Fox, and T. Hey. Grid Computing: Making the Global Infrastructure a Reality. Wiley and Sons, 2003.
[21] I. Foster and C. Kesselman, editors. The Grid – Blueprint for a New Computing Infrastructure. Morgan Kaufmann,1998.
[22] Yang-Suk Kee, et al. Efficient Resource Description and High Quality Selection for Virtual Grids. CCGrid, pp. 598-606, Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 1, 2005.
[23] Parallel Workloads Archive.
http://www.cs.huji.ac.il/labs/parallel/workload/.