www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 10 October 2017, Page No. 22616-22626
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i10.11

Hop Onset Network: Adequate Stationing Schema for Large Scale Cloud-Applications

Nagalakshmi Batchu[1], Nagi Setty A[2]

[1]NIIT Private Limited, India
[2]Neudesic Private Limited, India

Abstract

Reducing the time that a user has to occupy resources for completing cloud tasks can improve cloud efficiency and lower user cost. Such a time, called cloud time, consists of cloud deployment time and application running time. In this work we design jump-start cloud, under which an efficient cloud deployment scheme is proposed for minimizing cloud time. In particular, VM cloning based on disk image sharing has been implemented for fast VM and application deployment. For applications with heavy disk visits, the post-deployment quality of service (QoS) may suffer from image sharing and consequently, application running time will increase. To solve this problem, different image distribution schemes have been designed. We test jump-start cloud through a Hadoop based benchmark and MapReduce applications. Experiment studies show that our design saves application installation time and meanwhile, keeps application running time reasonably low, thus makes cloud time shorter.

Key Words- Jump start, KVM, QoS, VM, Deployment

1. Introduction

Cloud has been looked as a natural evolvement for data centre (DC) so that resources such as CPU, memory, storage, and IO/network in a DC can be dynamically and flexibly grouped or allocated, to serve clients with different service level agreement (SLA). Virtualization has been looked as a de facto management technology because of the speed, flexibility and agility it brings to cloud resource management. Providing resources in terms of virtual machines (VMs), it is easy to assign an application with a set of quantitatively measurable resources, e.g., a number of VMs where each VM has an assigned CPU slots (VCPU) and an amount of memory. As applications and OS are bundled into VM images, virtualization enables users to run their applications with a supporting OS of their choice, giving users the exposure to system-level

Deploying and launching a VM-based user cloud application on top of a cloud infrastructure in general involves the following steps. First, a VM image mainly consisted of the targeted OS and user applications is compiled and then up-loaded to a VM image repository within the cloud infrastructure. Then a number of VM's are launched on a set of DC physical hosts. Finally, application environments (IP, hostname, etc.) are configured and target applications are deployed.

It is desired that a cloud can jump-start, i.e., a cloud can be deployed and ready in short time. We think cloud jump-start time, in this work defined as the overall time that a cloud application is deployed and becomes functioning under desired QoS, should be fast for improving cloud infrastructure services for the following reasons:

1. One of the most important goals for cloud service providers is to make quick response to clients' resource requests and provide clients with applications running in desired states as soon as possible.

2. Even during the deployment time the resources reserved for a cloud application cannot be used by any other applications, which means a kind of resource waste. Making clouds jump-start can significantly improve cloud resource efficiency because a cloud normally serves thousands of users.

3. Users are usually billed based on the amount of cloud resource requested and the length of time to occupy such resource. The overall time that a cloud application occupies the resources needed, which we call cloud time, then is an important criteria for measuring, e.g., non-long lasting cloud services. It is to the interests of users to reduce cloud jump-start time for minimizing cloud time, unless the prompt cloud deployment results in a much longer application execution time.

4. Reducing cloud jump-start time helps improve cloud service availability. Various system software and hardware failures or software patches may require the entire cloud to shut down and restart. Faster jump-start time will help reduce system downtime and improve cloud availability.

Reducing cloud jump-start time has become a research challenge, especially for virtual cloud environment that normally involves a distributed system consisted of a large number of VMs. Having a virtual cloud environment ready requires many VMs deployed with a complex configuration setting, thus leading to challenges for both VM deployment and application environment setup. For example, it takes Amazon Elastic MapReduce about 4 minutes to deploy a Hadoop cluster with 20 VMs.

We propose using VM cloning to reduce VM installation time, thus to achieve a shorter jump-start time. A small-sized memory state file is generated from a suspending VM and distributed. Upon receiving the memory state file a VM can be resumed in a short time. The base image is shared in an image server, e.g. a NFS server, and VMs fetch extra data from the base image on-demand, e.g., through leveraging remote disk image access techniques that have been widely used in VM migrations.

A design issue not clearly addressed by previous cloning works such as how to differentiate cloned VMs. cloning from the memory state can only generate identical VMs. These VMs can't start working immediately before their memory states for MAC/IP addresses and the roles they play in the cloud application are assigned.

Another design issue not addressed previously, which we think even more important, is how to guarantee application post deployment QoS. The QoS may suffer from VM cloning when there are a large number of live VMs sharing the same disk image, and for each VM disk image access is frequent. The disk IO for accessing the image then can be crammed, which leads to a long delay for image data fetch. The delay may cause application interruption and bad user experience for real-time applications, or a longer task accomplishment time for result-oriented applications.

We design and evaluate a comprehensive jump-start cloud deployment frame work that considers all of the above issues. We implement VM cloning for fast VM launching, design specific daemon for VM differentiation and the consequent application deployment, and propose different disk image distribution methods for meeting post-deployment cloud QoS. We focus on non-long lasting VM-based Hadoop applications, and consider major QoS as how soon a cloud task is accomplished. The evaluation criteria then becomes cloud time, which is composed of jump-start time and application running time. The research methodology, however, can be applied to exploring how the proposed framework works for long lasting applications, by investigating intermediate stages of applications instead.

In summary, our major contributions are as follows:

We design jump-start cloud deployment framework that balances deployment time and post-deployment QoS. In this framework a KVM fast cloud application installation scheme is a designed.

– The scheme clones VMs upon memory state file, differentiates VMs through post VM configurations, and installs application through VM meta-data. Depending on application image access patterns, different image distribution schemes are designed to solve image access bottleneck problem.

– We identify cloud jump-start time as a critical parameter for cloud service measurement, and propose using cloud time that consists of cloud jump-start time and application running time for measuring non long-lasting applications.

– We design a Hadoop cloud application benchmark. Through running it and cloud applications, we conduct intensive experiments for understanding, evaluating, and validating the proposed schemes. Experimental data shows that the proposed framework minimizes cloud time by making the time consumed on both application installation and application execution relatively short.

2. Jump-Start Cloud Deployment Overview

The targeted usage is user-defined large-scale cloud applications where both applications and their execution environment (e.g., OS) are created and/or defined by client. This may likely be one of major cloud usage models because future cloud is able to expose clients its infrastructure and let client determine the soft- ware environment. When the image is created by client, cloud services may have

1) A better efficiency due to a close bundle between application and its execution environment and 2) a better privacy because the client owns the entire software stack and data. In this model, to request cloud resources a client submits disk image to the cloud. The cloud works on this image to deploy and run the user- defined application. The application is on-the-fly, and for each request cloud has to establish a new execution environment. After the application is completed, this environment including VMs and data will be removed.

Our goal is to design a comprehensive cloud deployment framework that can make a non long-lasting user-defined cloud application finished in shortest time, i.e., makes its cloud time minimum. We care about the cloud time for result- oriented applications because for client a shorter completion time means a better QoS and a lower cost (a client will be charged based on a shorter resource occupation time), and for cloud a shorter cloud time can improve cloud resource efficiency (the released resources from the finished application can be used by other clients).

The two major steps after the cloud has loaded user image is application deployment and application running. To make the application deployment time (including the VM deployment time) short, our proposed technology for jump-start cloud is VM cloning. VM memory state files (or snap shots) are generated first at a physical machine where the application image is loaded. They are then delivered to the physical machines where VMs are supposed to be located at. As the size for a state file is small, the time for distributing it is short. VMs can then be quickly activated at destined machines. Once application metadata is sent to these VMs accordingly, application can start to run.

Such a fast application deployment may cause negative impact on application performance. A VM may visit the image from time to time. This happens, e.g., when disk image contains a large amount of data that has to be processed by the cloud application. Generally, a number of VMs will share and visit the same image, e.g., through NFS. In case a large number of image visits from a large number of VMs over the disk access IO, an

image visit for a VM may take long time. If we consider the case that for an application the parallelism has been done perfectly at VM level so that there is only single thread in each of the VMs, the increased image access delay for a VM will result in the same increased running time for the application part running in that VM. Keeping image access delay low thus is critical.

In our efficient cloud deployment framework we add one more step before application deployment. The application disk image will be distributed to a number of physical machines within the cloud, if needed, to keep cloud time low for applications with a large number of image visits. Basically, a few image copies will serve VMs so that the number of VMs that share the same image becomes smaller. Consequently, the request rate for visiting a particular image becomes lower, which leads to a lower average image access delay. Because distributing image also takes time, whether or not to distribute images depends on applications' image visiting pattern, including application scale (number of VMs) and image visiting load per VM. In jump-start cloud images can be distributed before application is deployed or after application has already started.

3 Fast Cloud Application Deployment

Fast distributed application deployment in virtualized environment include VM installation and application deployment. In this section we present the detailed design and our solutions for implementation issues.

3.1 Fast VM Installation and Differentiation

For distributed cloud applications we deploy VMs through live cloning in a cloud that supports KVM. The memory state file of the original VM is distributed to the hosts (i.e., physical machines) where the VMs are supposed to be located. At this stage we consider all of VMs share one disk image, e.g., through NFS. Since the size of memory state file is small, distributing such a file within cloud takes a short time. Once a destined host receives the state file, the VM can be resumed at that host immediately. The state file can be either submitted by a cloud user or generated by cloud itself, through storing the memory state into a separate file.

The cloned VMs are exactly the same as their parent, including networking configurations for IP address and MAC address. The next step for VM installation then is how to enable these VMs with individual network configurations and other distinguished features if any. Ideally, if a hypervisor at the host of that VM knows which part of the VM memory state should be changed for such di- versification, it can modify the internal memory state and consequently diversify the VM. However, up-to-date there is no such technology due to the extreme complexity of memory state file.

In this work we diversify VMs by adding a daemon into each of them. The daemon will load the information needed, called VM metadata in this paper, to distinguish its host VM. The VM-metadata is obtained from cloud manager. Based on the metadata the daemon triggers a re-configuration for the VM to have its distinguished features enabled.

A remained design issue is how a daemon obtains the metadata of a VM. Note that a daemon cannot obtain the metadata from the cloud manager directly through the DC network, as the VM cannot communicate with any other cloud components until its diversification is completed. Therefore, the metadata can only be obtained through some other ways based on the virtualized hardware.

In this work we collect VM metadata by generating an ISO image, as what has been done in VM Plants. The metadata stored in the ISO image will be delivered to VM host along with the memory state file, while the daemon accesses the information from CDROM. The ISO image is customized through an interface open to cloud users. When creating a VM image the user adds additional information to VM-metadata, e.g.,

through a user script added to the ISO image. Once the ISO image is invoked after the VM cloning, customized configurations including application configurations will be accomplished.

We implement the fast VM deployment in KVM, hypervisor supports offline migration, during which memory state is stored into a file and then loaded in the destination. Two important implementation works are VM meta-data construction and self-configuration daemon design.

3.2 Cloud Application Deployment

Once VMs are installed, each of them has to act as different roles of a distributed application. If a VM template also carries application memory state file, the application can be installed along with VM. However, in some applications such state file depends on cloud settings, which cannot be obtained before it starts to run in cloud. For example, in a process of HDFS in Hadoop, the storage ID is generated according to the cloud configurations and once it is set, it cannot be modified. It is then difficult for a client to provide a storage ID in the VM template.

We extend the VM-metadata and design a framework that makes dynamic application configuration automatically. Under this framework, client adds a script to the VM-metadata that implements the client interface, and sends a script to a centralized configuration management server (CMS) that is designed for collecting information and delivering customized configurations.The CMS then can assign the VM a role through sending meta data, to initiate application deployment. After the fast installation of VMs the CMS has the system VM deployment information through collecting reports from these VMs. It then de- livers configuration commands based on client's request, through which the role of each VM is determined. For example, when deploying a Hadoop cluster in the cloud, after receiving VM installation completion reports from all VMs, the CMS decides which VM works as Master. It then tells other VMs the Master's IP address and their roles, which are Slaves. Slaves then contact the master to get application started.

The application configuration framework is implemented with Python. When a request is submitted to the cloud stack, the CMS makes a new waiting list for VMs regarding to the request. After configuring network, the daemon in a VM will invoke the script named self in the VM-metadata. The activated VM then communicates with the CMS using remote procedure call (RPC). Upon receiving a PRC, the CMS invokes the user policy to make a decision what should be returned to the VM. The returned value can be additional meta-data or a configuration command. Typically there is only one interaction between a daemon and CMS.

A daemon can trigger more RPC calls if more information is required.

4 Image Distribution for Application QoS

When applications depend on frequent, large number of disk image visits, the delay for a VM to obtain data from disk image may be large. This lowers computation efficiency and results in an increased application running time (i.e., a worse QoS). To reduce the average image access time, disk image can be distributed to a number of physical machines so that each of the images is visited by fewer VMs. In general, image distribution helps to reduce cloud time only if the time cost for the image distribution is less than the consequent time savings on disk image reading.

Since the size of an image is generally very large, it will take a long time to copy an image from one physical machine to another. In addition, for any part of a large-scale application, which runs in a VM in our case, it may only visit a small portion of the image. It is then not necessary to distribute the whole image to every host. Another reason for not distributing too many images within cloud is

for storage savings, as storage is considered as cloud cost as well.

Ideally, if client or Cloud is aware of which part of image a VM will visit, the image part that will be visited by a lot of VMs then can be properly distributed. Unfortunately, identifying roles of each part of an image and separating them is not an easy work. Therefore, as a first step, in this work we consider the case that client or Cloud does not have the detailed knowledge of the image contents.

However, they may know whether the application will visit the image frequently or not either through previous experience or Cloud monitoring. The major image distribution schemes we have implemented in jump-start cloud deployment framework are pre-deployment, background deployment, and on-demand deployment.

4.1 Pre-deployment

In the pre-deployment distribution scheme the image will be copied and distributed in cloud before application runs. Given a client request that has defined the number of VMs required for running the application, the cloud decides the number of images to be distributed so that for each image, the number of VMs it will serve is below a threshold value. The number of images to be distributed depends on image reading load from each of the VMs as well as the network and disk IO bandwidth availability.

After the number of disk images to be distributed in the Cloud has been deter- mined, a remained issue for pre-deployment is where to store these images (i.e., where to locate these image servers) and how to store them. Image server location depends on cloud system deployment and configuration. A general guideline is to store an image to make it easily accessed by its serving VMs. In our frame- work, an image is located at the same subnet as its serving VMs. After receiving an image, the image server stores it in its local disk. Because reading from disk takes longer time, to enhance image access speed, the image can be stored in the memory. This, however, is at the cost of occupying a much larger size of memory.

For applications that client or cloud is aware which part of the image may be visited by applications, only that part needs to be distributed. A typical such application can be one that processes client provided data, where data is the major disk part to be visited and distributed. Other disk parts, e.g., application execution OS, are shared by all the VMs, unless image access bottleneck problem occurs then we can use background deployment method introduced later to resolve it. Distributing partial image significantly reduces image distribution time.

4.2 Background Deployment

Usually for any submitted cloud application it is not easy for either client or cloud to predict image reading pattern. Therefore, it is hard to make decision whether to pre-deploy disk image and how many images should be deployed. In our framework, for applications without the knowledge of image visiting patterns, all VMs share a single image at the beginning. Through monitoring the image access load as well as image reading time, the Cloud manager then decides whether the image has to be copied and deployed at other physical machines. During image deployment, application may continue running. The number of images that should be finally distributed in the cloud can be reached gradually by adding images from time to time, until image access time is below a required value. This value is estimated through observing under what access latency application performance may not be seriously affected. An implementation issue regarding to background deployment is about the So-called VM re-direction. Because at the beginning all the VMs are directed to a single image, when other images are ready, some of VMs have to be re-directed to their new assigned image servers. In this work we resolve the issue by changing the image direction command in

KVM, which determines the image access for VMs.

4.3 On-Demand Deployment

Lots of image parts, e.g., OS bootstrap and drivers, may not or seldom be read by applications. Therefore, it is not efficient to distribute the entire image. However, pre-deployment requires a clear knowledge for what part of an image will be accessed during application runtime, which is difficult. To address this issue, we propose on-demand image distribution scheme.

In this scheme, at the beginning a number of images are built along with the original image in cloud. These images are actually empty, and we call them pseudo images. Like the original image (or real image), a pseudo image also serves a number of VMs. When a process running in a VM served by a pseudo image requires some image data, it will visit the pseudo image serving it. If the data requested is not available at the pseudo image, the pseudo image will ask the real image for that part. Once the pseudo image receives the data, it delivers it to the requesting VM. At the meantime, it stores a copy for serving other VMs that may require the same image part later. On-demand deployment may lead to a longer delay for some image readings, yet it can significantly reduce communication load for image distribution.

To further improve the performance of on-demand image deployment, multi- cast and data pre-fetching can be applied.

5. Experimental Results

5.1 Experimental Data

Hadoop Deployment Time. In **Fig**.1 we show when applying our fast application deployment scheme that shares a single image, the deployment time of a Hadoop with different scales, i.e., different number of VMs. The overall deployment time is around 10 seconds, which is much shorter than deploying such a Hadoop in EC 2. In addition, the deployment time does not significantly increase when Hadoop scales up, because in such application after loading the VM memory state file, a VM seldom visits the disk image. Access delay for disk reading at the image server then is short. The metadata delivered for VM role configuration has a small size too. Therefore, even if the number of VMs increases, there is a minor increase for application deployment time. In Fig. 2 we show the detailed time contribution from different stages of Hadoop deployment at each of the VMs, when 56 VMs are assigned. At this scale, the time for memory state file distribution (for VM cloning), VM resume, and application configuration are about 0.5s, 2s, 8s respectively.

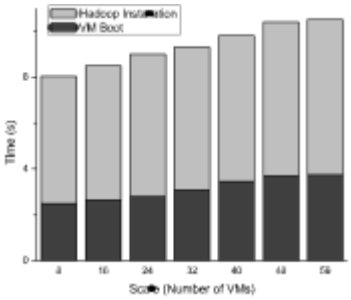


Fig. 1. Deployment time for Hadoop at different scales

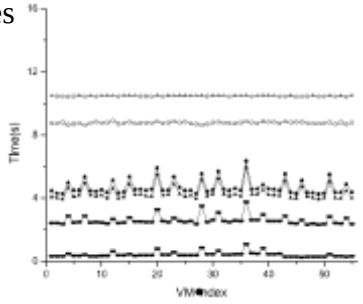


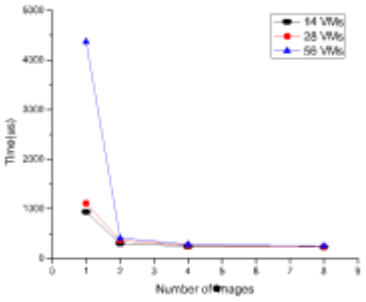
Fig. 2. Time taken at different stages and different VMs

In Table 1 we compare the time needed for deploying MapReduce with different scales in jump-start cloud and EC 2. Generally, EC 2 will take a few minutes while jump-start cloud takes less than 20 seconds. Considering a MapReduce application such as sorting may take only a few minutes, fast deployment through jump-start cloud can greatly improve user experience and cloud efficiency.

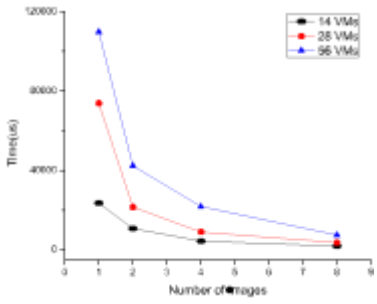
Table 1. Deployment Time: Jump-Start Cloud vs. EC 2

Application Scale(Number of VMs)	20	50	100
Jump Start cloud Launching time	13s	14s	15s
EC2 launching time	238s	291s	304s

Improvement through Image Distribution. In Fig. 3 we shows distributing disk image does help to reduce average image access delay, and consequently the overall application running time. Under our testing scenario, when the number of images changes from 1 to 2, the delay decreases significantly. When the number of images increases further, the gain on access delay becomes trivial. This implies that for reducing access delay, distributing too many images may not be necessary.



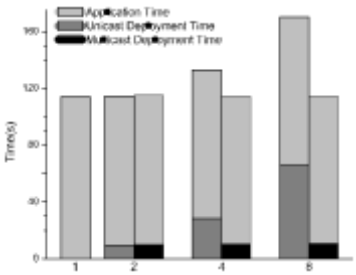
(a) Sequential Access



(b) Random Access

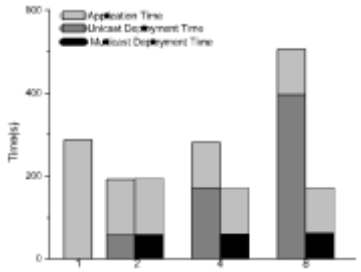
Fig. 3. Average access delay per request under different number of images

We show in Fig. 4 how image pre-deployment may help to reduce application cloud time when VM has a sequential reading pattern. Both unicast and mul- ticast are used for image distribution for comparison. It is observed that image distribution works for heavy image visiting load. Multicast has a better gain because it helps to save image distribution time. From our experimental data we found that image pre-deployment can help to cloud time if when a single image serves all the VMs, its disk IO is fully occupied. As we mentioned previously, when VMs read disk sequentially, the equivalent disk IO bandwidth in our setting is about 20MB/s or 160M b/s. For the unicast case, if we use 1 GB/s for intra-cluster connection, the maximum number of images is approximately 2. The experimental results proves the mathematical deduction. When the number of images becomes greater, pre-deployment has a longer cloud time. This can be explained by referring to Fig. 3. A greater number of images results in very minor access delay improvement at a much longer image distribution time. The overall time saving on application execution then cannot compensate the time taken on distributing images. For the multicast image distribution case there is no such issue, through which generally the a greater number of images are distributed, a shorter cloud time can be achieved. However, when the number reaches a value, the gain from multicast is minor as well. Since the number of images to be multicasted determines how much cloud network bandwidth and system storage are needed, this number should be carefully selected based on application parameters and cloud configurations.



(a) $\lambda = 50$

DOI: 10.18535/ijecs/v6i10.11



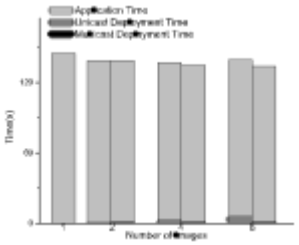
(b) $\lambda = 200$

Fig. 4. Cloud time for application with sequential access under different number of images

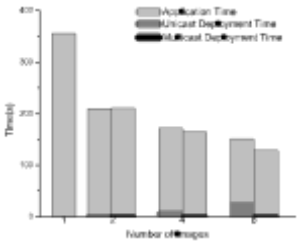
Fig. 5 shows the cloud time for random image access pattern. The overall performance trend is similar to what has been shown for the sequential disk reading case. For unicast the optimum number of image to be distributed, however, is greater than 2 because the equivalent disk IO bandwidth now is much smaller. Our test result shows the IO bandwidth is about 2MB/s, the optimum number for distributing images is about 8.

Background and On-Demand Deployment. When using background image distribution we found there was no gain on cloud time reduction. The reason is that image distribution can help to reduce overall cloud time only when sharing single image, image disk IO is fully occupied. Therefore, background distribution under heavy disk visiting load will contend disk IO with applications, making the overall data getting out from disk image greater than the case when all VMs share a single image. However, background distribution may help is some particular application cases where QoS can be degraded to keep applications from being interrupted. For example, for a streaming video application at the beginning a fewer frames per second can be played, while the saved disk bandwidth can be used for image distribution. Once the background image distribution is completed, the performance becomes normal.

In Figure 6 we show cloud time improvement when on-demand image distribution is applied. Different numbers of images (including one real image) are deployed. We suppose 50% of the image will be accessed by all of the VMs,



(a) $\lambda = 5$



(b) $\lambda = 20$

Fig. 5. Cloud time for application with random access under different number of images while other parts of the image will be accessed by VMs individually. It is observed that on-demand distribution can help to reduce cloud time. We also found that the deployment with 3 pseudo images has a little worse gain that from

1 pseudo image case. The reason is that the visiting load to the real image is the performance bottleneck for on-demand distribution. This visiting load, actually, is determined by the number of the physical machines it serves. The number of physical machines the real image serves is greater in the 3 pseudo image case in our settings (The cloud has 8 physical machines), thus it has worse performance. It is also observed that deploying one pseudo image in the on-demand scheme results in a better performance than deploying one more real image through pre-deployment, because pre-deployment has to distribute the image part that will not be accessed as well. When the number of image increases, pre-deployment works better, at the cost of communication and storage.

In Figure. 7 we show how image access pattern impacts the cloud time gain in on-demand distribution. The access pattern we care about is how much of the image will be accessed (or shared) by all of the VMs. We consider only 1 pseudo image is deployed. According to the figure, it is observed that the more image data to be shared, the more cloud time reduction can be achieved through on-demand distribution. This is because the major role of the on-demand distribution is to reduce the visiting load at the real image. This load is mainly determined by the size of the disk data to be shared. Therefore, the on-demand distribution works better when more image part has to be shared. Impact of Image Distribution on Real Application. In Fig. 8 a) we show the impact of image distribution on a MapReduce sorting application. There are 2G data to be sorted, through a MapReduce that runs over 56 VMs. The data is provided by user and originally stored in a disk image. We modified Hadoop so that data can be loaded from local storage. Each Mapper fetches the same amount data from the disk, and no data is shared. Mappers read data in turn, following a sequential reading pattern. As in our previous experiment, the tested results show the equivalent disk IO bandwidth can be up to 20MB/s. Because

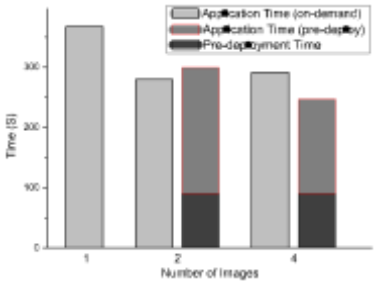


Fig. 6. Cloud time with on-demand

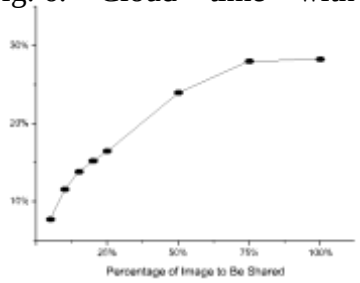
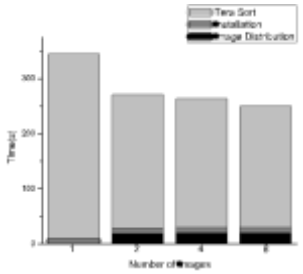
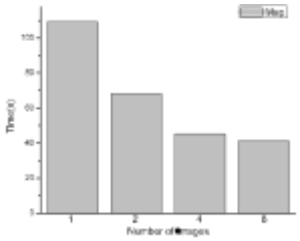


Fig. 7. On-demand distribution gain vs. image distribution image visiting pattern



(a) cloud time



(b) Map time

Fig. 8. Impact of image distribution on real application the network has a bandwidth of 1G, according to the analysis in 4.1 image distribution should be able to help to reduce cloud time. The experimental data proves it. We show in Fig. 8 b) that the major gain on application running time comes from Mapper, because that is the process involving most image access for loading data.

6 Conclusions

We enable jump-start cloud that applies an efficient deployment framework we designed to reduce cloud time for resource efficiency and service quality improvement. VM cloning is used for fast application deployment, and image distribution is used for post-deployment QoS. We test different application image access pat- terns and cloud system configurations to study cloud application characteristics and evaluate the deployment scheme.

References

1. Armbrust, M., Above the Clouds: A Berkeley View of Cloud Computing. Technical report, UC Berkeley Reliable Adaptive Distributed Systems Laboratory (2009)
2. EC2: Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/
3. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic Virtual Clusters in a Grid Site Manager. In: HPDC (2003)
4. Google app engine, http://code.google.com/appengine/
5. Windows azure platform, http://www.microsoft.com/windowsazure/
6. Amazon elastic map reduce, http://aws.amazon.com/elasticmapreduce/
7. Hadoop, http://hadoop.apache.org/
8. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M. Black-box and Gray-box