On-demand Mode of Legacy Desktop Software and Its Automatic Deployment for Cloud-Computing Environment*

Youhui ZHANG, Gelin SU, Weimin ZHENG
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science, Tsinghua University, Beijing 100084, China
zyh02@tsinghua.edu.cn

Abstract

Owing to some Cloud Computing offerings (like EC2), now IT infrastructure can be deployed on demand quickly and elastically. Besides the virtual hardware and system software, in many cases, it is necessary to deploy application software in a similar way; then users can get a fully-functional work environment with required software conveniently. This paper proposes such a solution that a user can customize which application software is required as well as virtual machines' configurations, and shortly (usually in seconds) all of he/she required can be accessed. This solution is based on application virtualization technologies, which can convert the enormous existing desktop software into on-demand software without any modification of source code. Moreover, because of the commonality of frequently-used applications, a central deployment mechanism, as well as the Copy-on-Write technology, is used to improve the efficiency of storage and management without data conflicts. Besides, some key issues related to Cloud Computing, such as pay-as-you-go accounting and prevention of illegal copy are considered. At last, extensive testing results are provided.

1. Introduction

Cloud Computing [1] [2] refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. Then, customers do not generally own the physical infrastructure; instead, they avoid capital expenditure by renting usage from a third-party provider and they consume resources as a service and pay only for resources that they use.

*Supported by Chinese National 973 Basic Research Program under Grant 2007CB310900, Chinese National 863 High Technology Programs under Grant 2006AA01Z111 and 2008AA01A201.

As [1] said, there are three types of Cloud Computing, which are classified based on the level of abstraction presented to the programmer and the level of management of the resources. This paper is focused on the first type that provides services of virtual hardware and system software, like EC2 (Elastic Compute Cloud) [3] [4]. Owing to EC2, the customer can deploy many customized computers with required system software and hardware configurations quickly. Besides virtual hardware and system software, in many cases, it is necessary to deploy application software in a similar way; therefore end users can get a fully-functional work environment with the required application software conveniently.

In fact, there are several providers of virtual-machine technologies have implemented some products to simplify application deployments, such as VMWARE's ThinApp [5] and Citrix's XenAPP.

This paper proposes such a solution that a user can select which application software is needed (only if they can be provided by the Cloud Computing operator) when he/she customizes the VMs' configurations in a Cloud Computing environment. And then, all he/she needed will be deployed in seconds without manual intervention.

To reach this target, two key challenges should be conquered:

1) On-demand mode of legacy desktop software should be provided, which is the baseline of automatic deployment.

On-demand software is a model of software deployment whereby a provider licenses an application to customers for use as a service on demand. Now most on-demand software belongs to Web-based applications (such as salesforce.com). In this mode, a web browser is employed as a running platform for applications with the collaboration from a remote server. However, applications running on the platform are not compatible with the mainstream desktop environment: they should be rewritten for the web situation.

As mentioned in the paragraphs above, we focus on such Cloud Computing environments that provide VM-based IT infrastructure on demand. Therefore, it

would be best to design a transparent method to convert legacy software into on-demand version.

2) Secondly, a flexible and fast delivery mechanism for on-demand software is necessary.

After the user has customized software, they should be delivered instantly (without manual intervention) and efficiently. Moreover, pay-as-you-go pricing is one of the key features of Cloud Computing. For the naturally-network-based software, it is not a trouble. However, for the legacy desktop software, it is an open problem.

To overcome these problems, we make the following contributions towards the automatic deployment of application software on the Cloud-Computing environment:

1) Converting legacy desktop applications into on-demand software transparently:

One model of on-demand software is proposed. Based on this model and application virtualization technology, we develop a method to provide a virtual running environment (including the virtual file system, etc.) for software, so that the latter can run on any compatible system on demand (without installation).

2) Deployment the on-demand software in a central mode with Copy-on-Write (COW) technology

One or more central data servers (depending on overloads) are used to provide software on demand for the deployed virtual machines, rather than place the required software on the VMs in advance. Because of the commonality of frequently-used applications in the Cloud Computing environment, this technology can decrease the storage consumption significantly. On the other side, different instances share the central data will cause access conflicts; therefore the COW technology is employed.

Moreover, some access optimizations, including the local cache, metadata pre-fetch and update, are presented, too.

3) Functions of accounting and prevention of illegal copy of on-demand software

Pay-as-you-go pricing is one of the key features of Cloud Computing. We design an access control protocol to implement such accounting and prevention of illegal copy functions based on the file virtualization technology.

4) The prototype and extensive performance tests are implemented

Combined with the VM technology, such a prototype is presented. In addition, extensive testes with diverse configurations (different server/client numbers, cache sizes, run turns, etc.) are given to measure the startup time of the on-demand software in this system, which is helpful to construct the outline of this system's performance and scalability.

In this paper, we first present the model of on-demand software and the design of its runtime system. Its deployment in the Cloud Computing environment is given in Section 3, as well as the functions of accounting and prevention of illegal copy. The prototype, combined with the real usage scenario, is introduced in Section 4. Then, the performance tests are presented. Finally, we present related works and the conclusion.

2. On-demand Software

2.1. The model of on-demand software

Most Windows applications need to be installed before they can run normally. And most do not write their implementation from stem to stern; instead a lot of Windows components provided by the OS are employed.

Then software can be regarded as containing three parts: Part 1 includes all resources provided by the OS; Part 2 contains what are created/modified/deleted by the installation process; and Part 3 is the data created/modified/deleted during the run time. For Windows OS, the resources here mainly refer to files/folders, the related system registry keys/values and environment variables. The formal definitions of software parts are presented as follows:

Definition of Part 2: In the initial stage, Part 2 is the set of all resources created/modified/deleted by the installation process; during the runtime, any modification of Part 2 will be moved into Part 3.

Definition of Part 3: In the initial stage, Part 3 is empty; during the runtime, any modified resource will be added into this part.

The content of Part 1 is unknown. However, it is not a problem, because our solution only makes software run on compatible hosts, which implies that all resources of Part 1 are available on the local system.

During the runtime, the running instance will access resources of all parts on the fly: some resources are read-only while some may be modified/added/deleted. Therefore, no part is fixed: the resource modified by the application instance at run time will be moved into Part 3.

Then, the design principles are drawn as follows:

1） All related resource accesses should be intercepted dynamically

2） Part 2 and 3 can be accessed on demand.

These parts are deployed in some central server(s) described in Section 3.1; and the following section presents how to design a runtime system to intercept and redirect these accesses of Part 2 and 3 to the real storage positions.

2.2. The runtime system of on-demand software

The destination of the runtime system is to make all parts accessible by the application's executable file transparently. Therefore, from the viewpoint of the software, it looks like that it has been installed on the host.

In our design, the install procedure of a target application is monitored. Then, Part 2 can be captured. During the runtime, APIs accessing files and registry entries are intercepted and redirected to the real storage position as needed.

The whole runtime procedure is presented as follows:

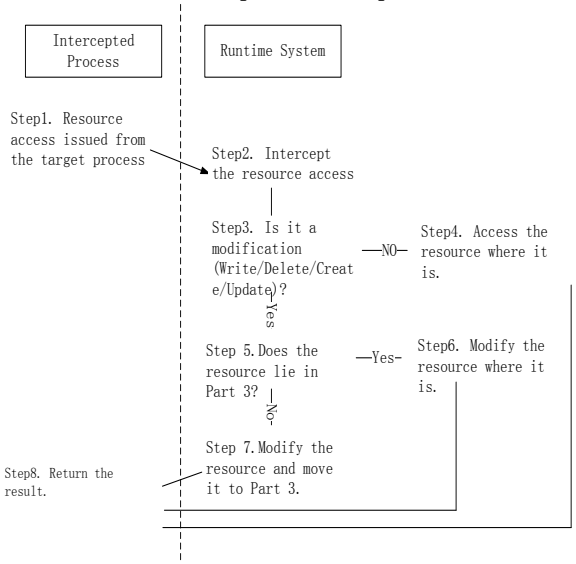

Figure.1 Work flow of the runtime system

In our previous work of on-demand software [6], user-level API Interception is employed to complete such a virtualization environment. In the current version, redirection of file access is enhanced and the others remain the same: we use a kernel-level interception mechanism, which can deal with more file access operations. For the previous work, we found that some file accesses issued by the intercepted process cannot be watched by the user-level mechanism. It looks like that they are executed by some kernel module (of the process), or from some undocumented APIs (?). Anyway, the kernel version works more completely.

This paper is focused on the difference, and other details can be found in [6]. We use Dokan framework [7], which can intercept accesses to assigned files/folders in the kernel space and redirect them to the user level (it can be regarded as FUSE [8] for Windows). Then, some callback functions in the user level that implement real data accesses are called.

Therefore, we complete the flow of Figure 1 (limited to the file accesses) in the user space to implement a virtual file system and the real data can be placed anywhere depending on the concrete usage scenario.

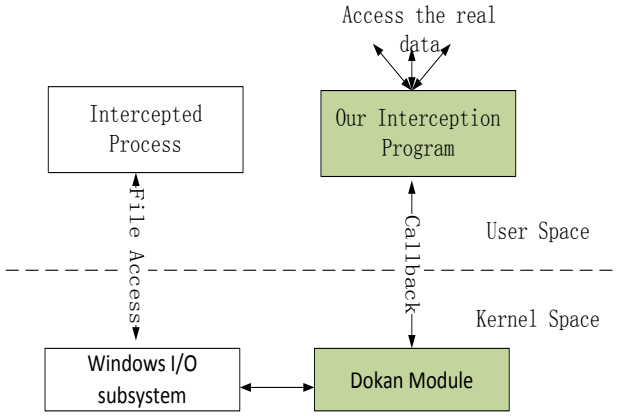

Figure.2 Access flow of the user-space file system

3. The Elastic Deployment

3.1. Central deployment

The key observation is that Cloud Computing's ability to add or remove resources at a fine grain (for example, one server at a time with EC2) and with a lead time of seconds or minutes allows matching resources to workload much more closely, which is called *Elasticity* [1].

Therefore, we should provide *Elasticity* of on-demand software. One direct method is to store the on-demand software in the VM. For example, after customization, the required software can be placed on a separated virtual disk and then attached to each VM instance.

However, it brings about management complexities: on-demand software is distributed to every running instance; then how to complete software update and access control will be a challenge. Moreover, the storage efficiency is low: many copies of same software may exist in different instances.

Considering the commonality of frequently-used applications, a central deployment mechanism is employed. Files of on-demand software are located on central server(s); as mentioned in Section 2.2, we have created a virtual drive using Dokan. When a file on this drive is visited, Dokan will direct accesses to the central server(s). Thus, only one copy is stored on the server and shared by multiple VM instances.

However, this method also has some weaknesses:

1） Access of remote files will introduce more delays;

2） Part 3 is modified during the run time, which causes write-conflicts among different users.

For the first issue, local cache on the VM instance is enabled: some frequently-accessed files will be cached locally and its replace strategy is also based on the usage frequency. The detailed control flow of cache is presented as follows:

Initial stage:

The Cache is empty, as well as the ordered-list of usage frequency of files;

Run time:

When on file is accessed, its usage number is increased by 1, and the ordered-list is updated accordingly.

Idle time:

The top n files of the ordered-list, if they have not been cached locally and if there is still free space in the cache, are downloaded on the background.

This method improves users' feeling remarkably, because we find that most frequently-used files belong to those accessed during the startup process while the startup time is an important measure of what it feels like to use the system for everyday work.

Besides these files, metadata of on-demand software are cached on VM ends, too, which include folder structures and other folders/files information; the metadata is transferred in the deployment procedure. During the normal runtime, when any software is being launched, its metadata version will be compared with the one on the server; if out-of-date, the startup procedure will pause till the new has been downloaded. Owing to the pre-fetch, VM ends can browser the file hierarchy smoothly.

For the second, a COW mechanism is employed: when one file is to be modified, it will be copied completely to the local cache firstly and all following accesses will be redirected to the local version. It means that each VM owns all Part 3 of on-demand software. Fortunately, for most software, the number of modified files is very limited.

In addition, Files in Part 3 are cache in the same way that the common files are managed; the only difference lies in that their reference numbers are set very large purposely to avoid to be replaced.

Moreover, load balance of accesses to multiple servers is also considered: each server owns copies of all software; therefore, when one server is overloaded, VM ends can visit others to get same data.

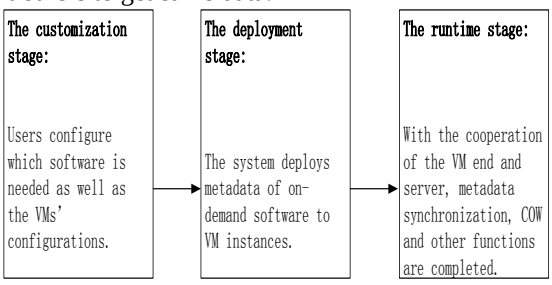

Figure.3 Three stages of the deployment for on-demand software

3.2. Usage accounting

Pay-as-you-go pricing is one of the key features of Cloud Computing. As mentioned in the previous section, when an application is being launched, the server will be contacted. Besides version number, account information is sent to the server, so that the latter can judge whether it is legal usage and begin accounting if the answer is yes.

When an application quits (from the viewpoint of our virtual file system, its main executable is closed), the accounting ends.

3.3. Prevention of illegal copy

As mentioned before, we implement a virtual file system to contain files of on-demand software, and then users can access application files just like they are using the local file system. Therefore how to prevent the illegal download is a key consideration. Otherwise, a user can copy applications to his/her local disk and use them without permission.

In this solution, an access control mechanism is implemented to protect some essential files (such as executable and DLL files of an application): we provide our own shell program to browse and launch on-demand software; and only this program and its descendants are permitted to access on-demand software files.

If the user attempts to use another program (for example, explorer.exe) to copy one essential file out of the virtual drive, the Dokan framework will identify whether they are issued from a legal process or not. Because explorer.exe is a program outside of the virtualization environment, its access will be denied.

One exception is cached files: they are stored locally; therefore it is possible that a user may locate and copy them out without permission. The current solution is that we store these files in a deformed mode, for example, file data is XORed with some predefined string and its layout is reorganized. In the next version, more sophisticated cryptography method will be deployed.

As a summary, Figure 4 presents the illustration of logic organization of the whole design: the virtual file system shoulders multiple tasks of file-access interception, access control, cache & metadata management, and real data access; and other accesses (to registry resources, etc.) are handled by another module.

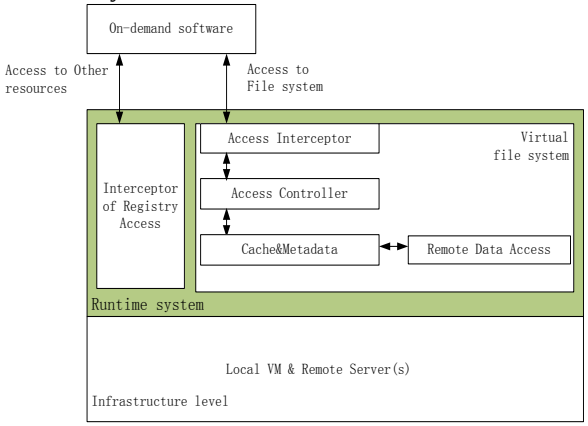

Figure.4 Logic organization of the whole system

4. The Prototype

4.1. Background

Tsinghua University intends to deploy a Cloud Computing platform for course experiments. Many courses ask students to complete study assignments and / or software experiments, so that some computer systems with specific software are required. We plan to construct such a Cloud Computing platform, which would provide lecturers and students with required systems instantly; and deployed resources will be revoked when the course finishes.

Now, such a prototype using XEN [9] is under development; we also implement the automatic software deployment for this prototype. Because this paper is focused on software deployment, more details of this Cloud Computing platform are skipped here.

4.2. Implementation

Based on the design and technologies above, a lot of desktop applications are transferred into the on-demand mode, including MS Word 2003, MS Excel 2003, MS PowerPoint 2003, Lotus Notes, Photoshop, Internet Explorer 6.0, Outlook Express 6, 7zip, UltraEdit, FlashGet, Bittorrent and so on.

A browser-based GUI is presented for users to customized the required software and VMs' configuration. As described in Figure 5, 4 virtual computers (in one virtual cluster) are being created; each node is equipped with 512M RAM and Windows XP OS, and the user can select required software through the bottom list. After about twenty seconds, all four virtual computers can be accessed through the RDP protocol, while VMs' network works within the network-bridge mode.

On the background, besides some PC servers as VMs' hosts, storage servers are deployed to store all on-demand software. After VM instances have been created, each of them will be assigned a unique IP address, which is now used as the accounting information for storage servers to judge whether the access is legal or not. In the next version, more sophisticated mechanism will be developed.

4.3. The shell program

The shell program is a bridge between the guest OS and storage server(s). It is placed on every VM instance in advance and will be launched automatically after system startup. It is also the interception process to complete data accesses, as described in Section 2.2 and 3.1.

As launched, the shell program initializes Dokan module and contacts the server to check the version of metadata. And the latter will retransfer metadata of all permitted on-demand software to it, depending on the guest's IP.

During the runtime, the shell maintains the local cache as well as the COW mechanism. Then the frequently-accessed files are saved locally only if the cache is available; moreover, as any file is modified, it will be saved in the cache as well and will never be replaced.

5. Performance Tests

5.1. Test environment

Four PC servers are used as VMs' hosts. All are Linux-XEN PCs, equipped with 2 GBytes DDR2 SDRAM and one Intel Core Duo CPU. The hard disk is one 160 GBytes SATA drive. One Linux server, equipped with one Intel Core 2 Duo E4500 CPU (2200MHz), 2

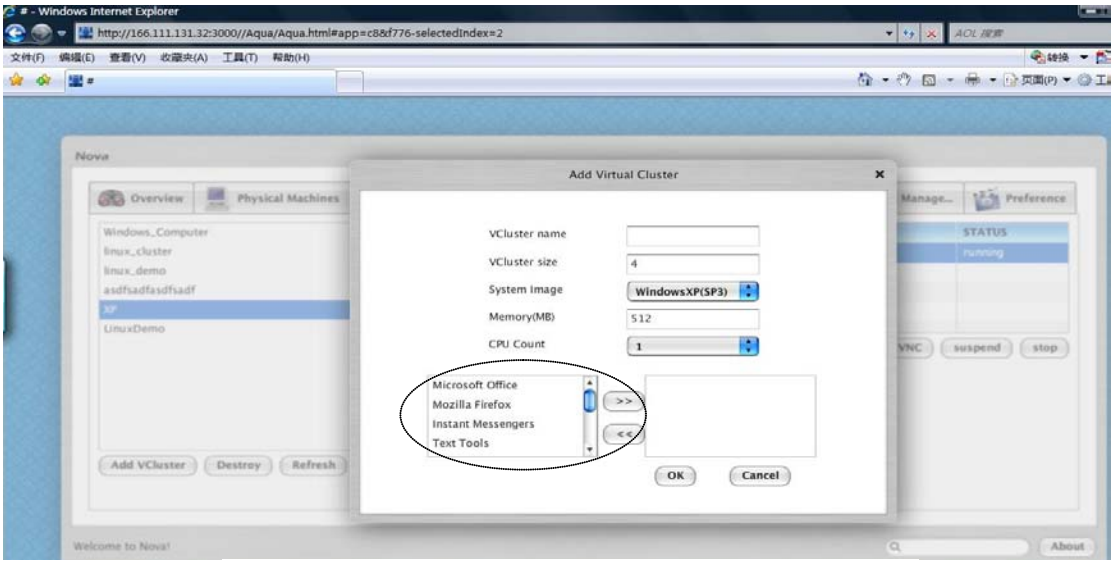

Figure.5 The browser GUI of system configurations

GBytes DDR2 SDRAM, and one 240GBytes SATA II disk, is used as storage server. All machines are connected with the 100M Ethernet.

Moreover, up to two Windows XP VMs are installed on every host; for each VM, one local cache with 200M bytes is reserved.

5.2. Test methods

Startup-time metric is tested.

The application start-up time is the key metric of our prototypes' usability. We launch eleven on-demand software applications at the same time through our shell program and record their startup time respectively.

One issue here is about how to test the startup time. Fortunately, Microsoft gives a special API, *WaitForInputIdle*, to judge whether the new process has finished its initialization and is ready to response user's input or not. As *WaitForInputIdle* returns, the elapsed time is recorded as the startup overhead.

The eleven applications are numbered in Table 1.

Table.1 Application List

Application Name	Number
putty	1
vlc	2
Acrobat Reader	3
ZM	4
Storm Codec	5
AbiWord	6
Gtalk	7
7-Zip	8
filezilla	9
UltraEdit-32	10
FlashFXP	11

5.3. Test cases

1) One VM, one storage server

Here, one VM is used as the client of the eleven software applications. Results of the first and second runs are presented in Figure 6 respectively.

During the first run, the local cache is empty so that all data are downloaded from the server on-fly. Before the second turn, some frequently-used files have been cached locally based on the usage information of the first.

In our cases, when the 200M cache is filled up, about 75% of accesses during the startup process from these applications can be satisfied locally.

The baseline startup time, which means all software are installed locally, are also given for comparison.

Based on results, we find that, compared with the local installed version, on-demand introduces 252% ~ 270% extra startup-overhead. The exception is *ZM* (148%), which is a game application with many graphic initialization works, so that IO delays are not the decisive issue.

Moreover, local cache is an effective optimization, which can reduce the extra overhead by about 40% ~ 58% (compared with the empty-cache version).

2) Two VMs (on one or two PCs), one storage server

In these cases, two VMs are used as clients, running on one or two hosts respectively. Their startup time is recorded in Figure 7 and 8.

From Figure 7 (two VMs on two hosts), it looks like that one more VM on another host does not introduce many overheads: when the cache is empty, compared with Case 1, the extra overhead is about 11.6% averagely.

For the case of two VMs on one host, the extra overhead becomes more: when the cache is empty, compared with Case 1, about 62% is introduced averagely. Because two VMs on one host cause some resource competition, the overall performance is impaired.

On the other side, local cache still plays an important role to decrease the startup time. For Figure 7, it reduces the extra overhead by about 42% averagely; while for the other, the average reduction is about 54%.

3) Four VMs (on two or four PCs), one storage server

In this case, four VMs are used as clients, running on two or four hosts respectively. Their startup time is recorded in Figure 9 and 10.

When the cache is empty, compared with the corresponding situation of Case 2, the case of four VMs on four hosts introduces about 8.1% extra overheads averagely. When the cache is full, it reduces the overhead by about 42% averagely.

Similarly, the case of four VMs on two hosts introduces about 3.2% extra overheads averagely, compared with the corresponding one in Case 2. When the cache is full, it reduces the overhead by about 41% averagely.

4) Eight VMs (on four PCs), one storage server

In this case, eight VMs are used as clients, running on four hosts respectively. Their startup time is recorded in Figure 11.

When the cache is empty, compared with the corresponding situation of Case 2, the case of eight VMs on four hosts introduces about 8.9% extra overheads averagely. When the cache is full, it reduces the overhead by about 44% averagely.

Based on the above results, it appears that our on-demand software will introduce fairly high overhead as the cache is empty. Moreover, some analysis (because of the limited space, detailed data is skipped here) shows that, Dokan framework itself almost doubles the file-access overhead because it introduces more context-switches between the kernel and the user space.

When the cache is used fully, the startup time can be reduced by half nearly.

On the other side, compared with the extra overhead introduced by more VM ends (on different hosts), more VMs on one host increase the startup time much more remarkably.

6. Related Works

6.1. Cloud Computing

Amazon EC2 [3][4] presents a true virtual computing environment, allowing customers to use web service interfaces to launch instances with a variety of operating systems, manage network's access permissions, and run image using as many or few systems as desired.

In contrast, Google's Cloud Computing platforms, like BigTable [10], MapReduce [11], and AppEngine [12], are different. For example, AppEngine is an application-domain specific platform, which just hosts traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. As we know, this mode is incompatible with the current desktop software, which asks developers to write new applications using Python or Java language.

In addition to the commercial Cloud Computing offerings mentioned above, which maintains a proprietary infrastructure with open interfaces, there are open-source projects aimed at resource provisioning with the help of virtualization. Usher [13] is a modular open-source virtual machine management framework from academia. Enomalism [14] is an open-source cloud software infrastructure from a start-up company. Similarly, Eucalyptus [15] is an open-source software infrastructure for implementing Cloud Computing on clusters; the interface to Eucalyptus is compatible with Amazon's EC2 interfaces.

Based on the practical requirement (in Section 4.1), we are developing an EC2-like platform so that a compatible software deployment solution is needed.

6.2. Fast software deployment

Virtualization has been deployed for fast software deployment. One solution is Progressive Deployment System (PDS) [16], which is a virtual execution environment and infrastructure designed specifically for deploying software on demand. PDS intercepts a selected subset of system calls on the target machine to provide a partial virtualization at the operating system level. This enables software's install-time environment to be reproduced virtually while otherwise not isolating the software from peer applications on the target machine.

Another similar and practical solution is Microsoft's SoftGrid [17]. SoftGrid can convert applications into virtual services that are managed and hosted centrally but run on demand locally. Application virtualization reduces the complexity and labor involved in deploying, updating, and managing applications.

VMWARE also provides such a solution, ThinApp [5], which packages an entire application and its settings into a single executable that is isolated from the OS through application virtualization. Administrators can plug ThinApp into the existing management infrastructure to accelerate application development and desktop deployment.

Compared with these solutions, ours is also based on the application virtualization while some unique designs are considered: a virtual file system is used for file access redirection and for central deployment with diverse optimizations; some Cloud Computing-specific issues are also solved. In addition, all above projects belong to commercial products, while our solution is the only one academic project, as we know.

7. Conclusions

This paper analyzes the on-demand mode for legacy desktop software, and converts them into services. Combined with the VM technology, we design and implement a fast deployment system for a Cloud Computing environment.

In this system, on-demand software is managed and hosted centrally but run on demand locally, which can also improve the storage efficiency. Moreover, based on a virtual file system, transparent application access, accounting, and prevention of illegal copy are completed. Besides, some optimizations, including the local cache and pre-fetch of metadata, are implemented.

References

1. Michael Armbrust, Armando Fox, etc. Above the Clouds: A Berkeley View of Cloud Computing Export. Technical Report. 10 February 2009. Available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.
2. GARFINKEL, S. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS . Technical Report. TR-08-07, Harvard University, August 2007.
3. Amazon Elastic Compute Cloud. Developer Guide, August 2009. Available at http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/.
4. DECANDIA, G., HASTORUN, D., JAMPANI, etc. Dynamo: Amazon's highly available key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (2007),

ACM Press New York, NY, USA, pp. 205–220.
5. VMWARE ThinApp——Agentless Application Virtualization Overview. White Paper. Available at http://www.vmware.com/files/pdf/thinapp_intro_whitepaper.pdf.
6. Youhui Zhang, Xiaoling Wang, and Liang Hong, Portable Desktop Applications Based on P2P Transportation and Virtualization. Proceedings of the 22nd Large Installation System Administration Conference (LISA '08) San Diego, CA. USENIX Association, November 9–14, 2008. Pp. 133–144.
7. Dokan: User Mode File System for Windows. Available at http://dokan-dev.net/.
8. Filesystem in Userspace. Available at http://fuse.sourceforge.net/.
9. Paul Barham, Boris Dragovic, Keir Fraser, etc. Xen and the Art of Virtualization. Proceedings of the nineteenth ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA. 2003. Pages: 164 - 177.
10. CHANG, F., DEAN, J., GHEMAWAT, S., etc. Bigtable: A distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06) (2006).
11. DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. In OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (Berkeley, CA, USA, 2004), USENIX Association.
12. Google App Engine: Run your web apps on Google's infrastructure. Available at http://code.google.com/intl/en/appengine/.
13. M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In Proceedings of the 21st Large Installation System Administration Conference (LISA), November 2007.
14. Enomalism elastic computing infrastructure. http://www.enomaly.com.
15. Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System, in Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid, Shanghai, China.
16. Bowen Alpern , Joshua Auerbach, et al., PDS: a virtual execution environment for software deployment, Proceedings of the First ACM/USENIX international conference on Virtual execution environments, March, 2005.
17. Microsoft Application Virtualization. Available at http://www.microsoft.com/systemcenter/softgrid/default.mspx.

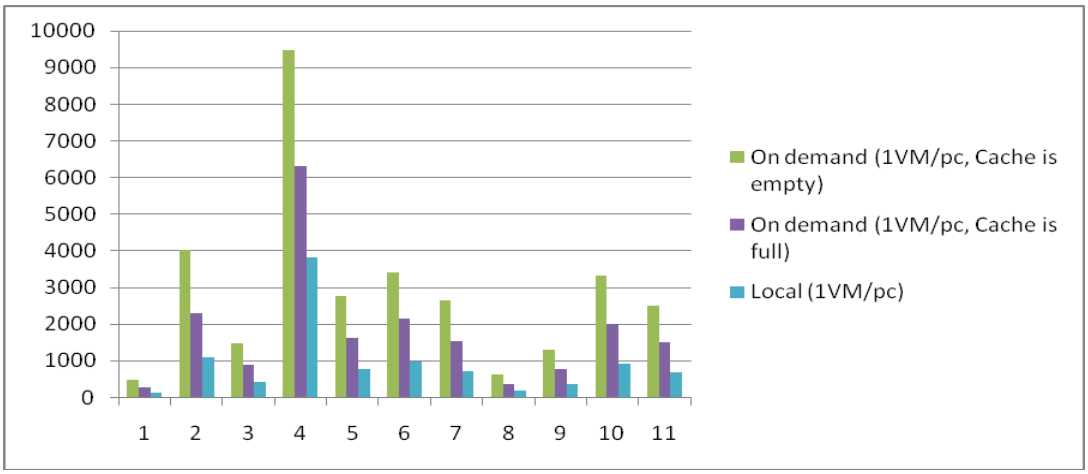

Figure.6 Startup time of Case 1

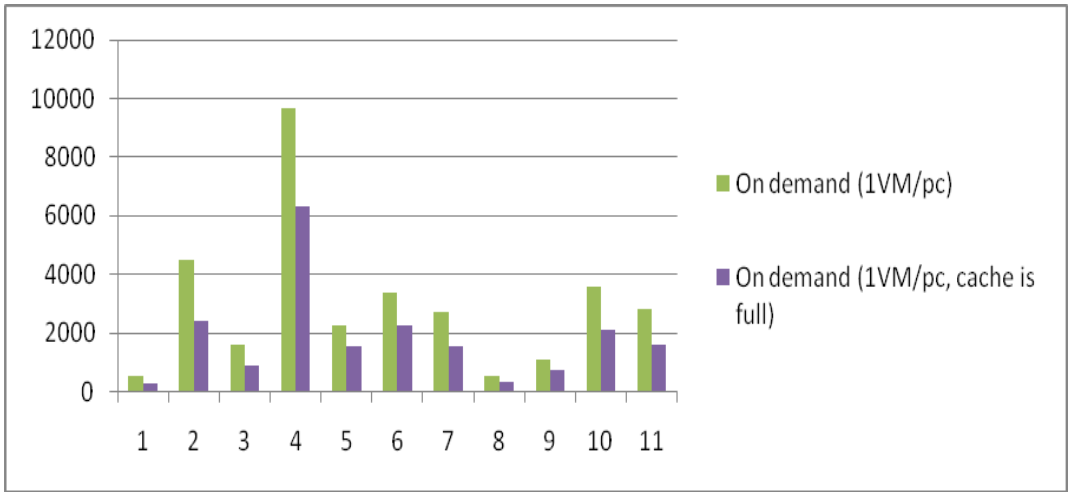

Figure.7 Startup time of Case 2 (Two VMs on two hosts)

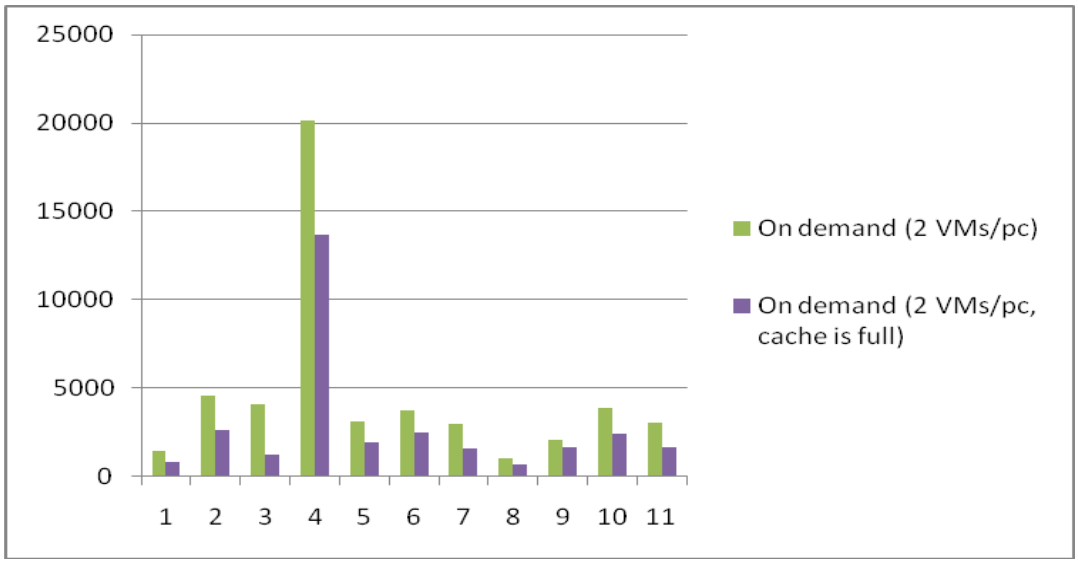

Figure.8 Startup time of Case 2 (Two VMs on one host)

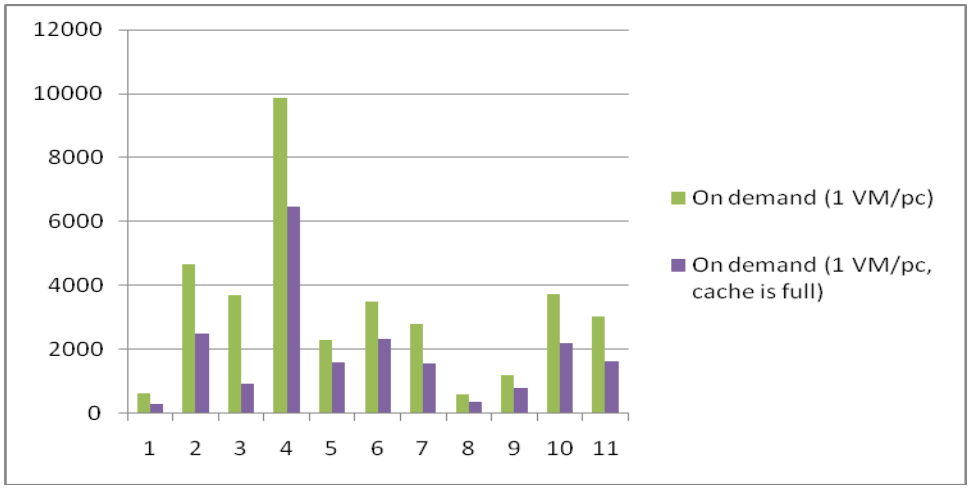

Figure.9 Startup time of Case 3 (Four VMs on four hosts)

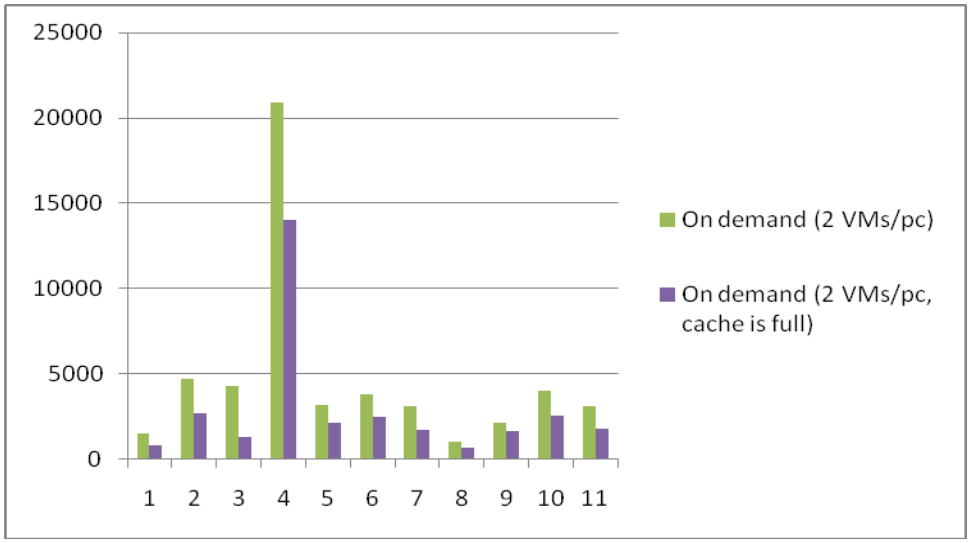

Figure.10 Startup time of Case 3 (Four VMs on two hosts)

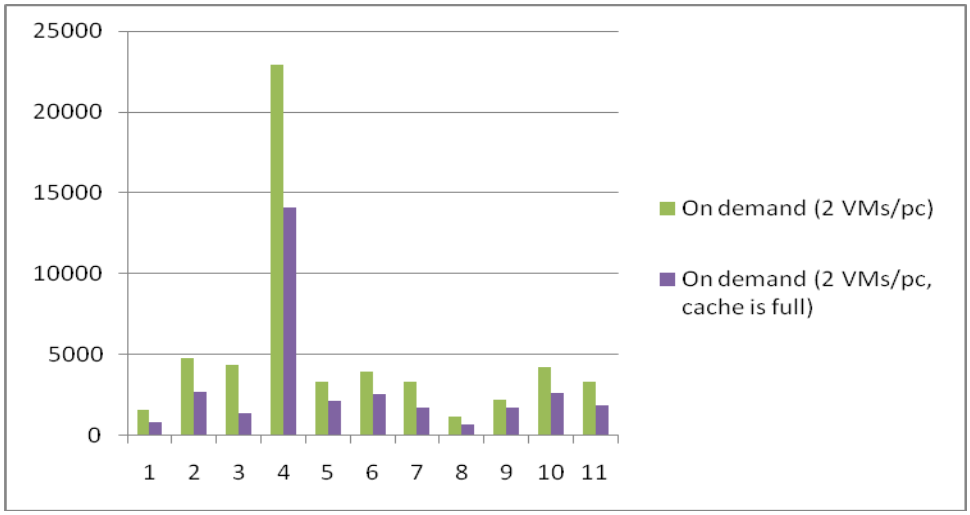

Figure.11 Startup time of Case 4 (Eight VMs on four hosts)