Streaming Legacy Desktop Software from the Cloud

Youhui Zhang, Gelin Su, Weimin Zheng
Department of Computer Science and Technology
Tsinghua University
Beijing, China
{zyh02, sgl08, zwm-dcs}@tsinghua.edu.cn

***Abstract*—Desktop Cloud can enhance business agility and reduce the total-cost-of -ownership, it introduces long network latencies and the power of local PCs cannot be utilized fully. This paper presents a light-weight mode for desktop cloud, which stores legacy desktop software in the cloud storage while streaming and running them on the user's PC locally. In details, based on the light-weight virtualization, software can be converted into portable counterparts and stored in the cloud. Moreover, a run-time system is implemented, including a user-space file-system for cloud, to stream and run the remote software on local machines. Local cache and data pre-fetch mechanisms are also adjusted to suit the file-access-pattern of software. This prototype has been implemented and tests show it is practical for much daily-used software.**

Keywords- Cloud computing; user-space file system; OS-level virtualization

I. INTRODUCTION

Existing software delivery model is usually based on a large number of distributed PCs executing operating system and desktop software independently. As mentioned by Gartner Group [1], for enterprises, deploying and managing personal operating systems and software in this mode are very expensive, which is the most important determinant of PC total cost of ownership (TCO). Personal users also face the similar problem.

Desktop Virtualization [2][3], combined with cloud computing allows users to run desktops on virtual machines (VM) hosted at the data center and access them as a service through some remote desktop protocol (RDP) [4][5], which is also called as "Desktop Cloud" [6]. Then, users can enhance business agility and reduce business risks, while lowering TCO.

But, for this solution, the client PC is used as a thin-client device, which executes the graphical interface of desktop to convey input and output between the user and the data center where software is really running. Therefore, there are two drawbacks: the user's feeling would not be good when it is employed across the Internet because of the long network latency [7]; secondly, processing power of the client PC cannot be utilized fully. To solve these problems, some Desktop clouds using Web applications are provided [8] [9]. Now, modern web applications are driving toward the power of fully functional desktop software such as email clients, productivity apps, etc. The user can access the personalized operating environment anywhere. But, the enormous legacy desktop software cannot be used in this model.

To fully utilize local PCs and legacy desktop software, a light-weight Desktop cloud solution is proposed here, which stores legacy Windows desktop software (rather than VM) in the cloud storage, and streams and runs them on the user's PC on-demand.

Because software is executed locally, the power of client PCs can be used efficiently; on the other side, as existing cloud storage services can be used as the backend without any modification, some key features of cloud computing, such as dynamic scaling of infrastructure, flexible usage based pricing, rapid service provisioning, can be still maintained.

To reach this target, three challenges should be conquered:

1) Legacy desktop software should be converted into portable software transparently.

OS-level virtualization is employed here to solve this problem. Every virtualization environment shares the same execution environment as the host machine. Therefore, such an environment can have very small resource requirements and thus its overhead is light-weight.

In our approach, existing desktop software is made portable: each software instance runs in an OS-level virtualization environment. This environment intercepts some resource-accessing APIs from the instance, and redirects them to the actual storage position(s) rather than the host. Then, in the user's view, he / she can launch software conveniently, although it does not exist on local disks.

2) Users should access the portable software in the cloud just like common desktop software; therefore a transparent and friendly delivery mechanism is needed.

A Windows user-space file system for cloud storage is designed. It acts as a proxy for file system accesses: file operation requests from portable software (e.g., CreateFile, ReadFile, WriteFile, etc.) to the Windows I/O subsystem (runs in kernel mode) will be forwarded to the corresponding user-space callback functions which visit real data in the cloud and send results back.

We implement such a file system based on Dokan [10], a development framework for Windows user-space file system (like fuse [11] for Linux), and the Amazon S3 interface.

3) Performance optimization

Some optimizations are adopted: all metadata is pre-fetched by client ends, which will be updated as necessary during the running time; local cache for frequently-used data is enabled to decrease the number of remote accesses, as well as a data pre-fetch mechanism which can adapt to different access patterns of diverse software.

We implement such a prototype for Windows OSes and extensive tests show that, these optimizations are efficient for much daily-used software.

This solution has its own limitation: portable software can only be used on compatible local OSes, while the traditional desktop clouds support any local OS only if a browser or some proper RDP client is available. However, we believe our proposal is practical because now Windows OSes still dominate the desktop PC market.

In this paper, we first present the model of portable software and the design of its runtime system. The user-space file system and optimizations are given in Section 3, as well as the access control mechanisms. The prototype is introduced in Section 4, as well as the performance tests. Finally, we present related works and the conclusion.

II. Portable Software

Usually, Windows 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 and the related system registry keys/values.

During the runtime, the software instance accesses resources of all parts on the fly: some resources are read-only while some may be modified/added/deleted. So, no part is fixed: those modified at run time will be moved into Part 3.

To make the existing software portable, all parts should be captured and made portable except for Part 1 while Part 2 and 3 should be accessed on demand.

A. *Installation Snapshot*

The modifications made by the software's installation process must be captured to enable Part 2 portable. Some system monitoring tool, like InstallWatch [12], is used.

In this implementation, a target application is installed on one clean Windows system, while InstallWatch is running to log those files created or modified in this process, as well as registry additions and modifications. Then, all files/folders/registry-keys created or updated are collected to be stored in a dedicated position. Till now, Part 2 is obtained.

B. *Runtime System*

Detours [13], a library developed by Microsoft Research Institute, is used to intercept those Windows APIs accessing files and registry entries during the runtime. Then, all accesses to files and registry entries are intercepted and redirected to the dedicated storage position as needed. In another word, API Interception is employed to complete a lightweight virtualization environment to make all parts accessible by the software's executable file transparently.

The strategies are:

1) Any non-modification operation is executed on site;

2) Any modification is moved to Part 3 so that the local host can be kept unchanged;

3) Any query will return the combination of results from all parts. If there is any duplication, Part 3 owns the highest priority while Part 1 is the lowest.

For details, please refer to our previous work [15][16].

III. Streaming Software from the Cloud

Now the windows software can run without installation as the runtime system provides all resources transparently.

Then, the next question is how to design a delivery solution that should own the following features or functions:

Transparent to users and software; access control; high efficiency on network access; dependent on the OS to the minimum extent

We design a user-space file system for cloud storage to reach the target. Firstly, with file system interfaces, the access method is compatible with the operation style of Windows desktops. And from the viewpoint of users, the remote software looks just like stored in a local drive

Secondly, two aspects of access control are implemented. The first is based on API interception to prevent portable software from accessing some local private information. The second works the other way round, which uses the process-hierarchy information to protect files of portable software from illegal copy.

Some optimizations are also adopted: all metadata is pre-fetched by client ends, and will be updated when necessary; local cache for frequently-used data is enabled to decrease the number of remote accesses, as well as an adaptive data pre-fetch mechanism. The user-space implementation can achieve most above functions in the user level, which is helpful to port our system to other OSes.

Finally, it is necessary to note that, the backend cloud inherently owns some features like dynamic scaling, high availability and rapid service provisioning. Therefore, this paper is focused on the client-end design, which communicates with the backend via some standard protocol.

A. *The file system framework*

This framework contains four parts: the first is the software instance accessing the user-space file system. Its related file operations are sent the Windows IO subsystem, which will be intercepted by our kernel proxy driver that redirects them to the user-level interception program that registers some callback functions to process corresponding operations respectively.

For more details, please refer to our previous work [15][16].

1) *File MetaData*

When a user launches the file system first time, the interception program contacts the remote server for login. Then it gets all metadata of his/her customized portable software and the version number based on the user ID. The metadata contains the following information:

Full paths of all files and folders; the attribute, size, creation-time, last-access-time and last-write-time of all files.

The received metadata is saved on the client permanently. When there is any file modification in the cloud, the updated metadata will be sent back during the subsequent procedure to keep data consistency.

Therefore, any metadata access can be completed locally, which speeds up the corresponding operations (like browsing directory, etc.) remarkably.

2) *File Data*

When a file is opened for read, it will be redirected to the remote position to fetch the real data (the local cache and pre-fetch are both employed, which will be described later).

If there is any write, a copy-on-write method is used, which means the whole remote file will be fetched to the client at first, and then any subsequent operation can happen locally.

As the file system is being unmounted, any new and modified files/folders will be transferred to a remote position (reserved for every user) hosted in cloud. So at the next time, the user can reach his/her latest metadata and data of all files.

3) *Remote access*

Our file system communicates with the backend through the S3 [14] interface, which is used by Amazon's notable cloud storage service. In S3, data is organized as objects in a bucket identified by unique IDs, which can be accessed through the standard HTTP protocol. Therefore, any file of portable software is regarded as an object in S3 and is identified by its full path name. Its URL looks like http://server_address/portablesoftware /full...path/filename.

For any new or modified file of a user, its naming style is different. For example, if John creates a new file (\program files\app1\file.name), the URL looks like http://server_address/portablesoftware/john/program files/app1/file.name.

In addition, for each user, the metadata info of his/her portable software, combined with the above-mentioned lists, is stored in a special position: http://server_address/ portablesoftware/username/metadata_list_version_num.

In summary, all users share portable software stored at the common place; each has the private space for any new or modified files to avoid write conflicts, as well as all metadata. Then, any whole file can be downloaded with the HTTP GET method while any part of a file can be accessed with the same method using the Range Header Field.

B. *Access control*

1) *Protect the local info*

As mentioned in Section 2, file accesses from portable software are intercepted, therefore the user can configure a white list to restrict the allowed range to protect his/her private info.

Another method is that any process of portable software is spawned with less permission rights; for example, its Access control List (ACL) is set as the Guest privilege. So the private data of the current user can be protected.

2) *Protect the portable software*

Users can access software files just like they are using the local file system, so how to prevent the illegal copy is a key consideration.

An access control based on the process-hierarchy is designed to protect essential files. The root of the hierarchy is the interception program that can access all files while any process outside of the hierarchy is forbidden.

For more details, please refer to our previous work [15][16].

C. *IO optimizations*

The user-space file system is grounded on the backend cloud. If all reads were completed remotely, our solution would be very slow. To alleviate this, two methods are adopted.

1) *Local cache*

We analyzed the file-access-pattern of the running process for some frequently-used software; it is found that, most frequently-used files belong to those accessed during the startup process, which only occupied a limited ratio of the whole capacity. For example, the following frequently-used software is converted into portable versions:

Abiword[1]*, PhotoShop, Lotus Notes, VLC (a powerful media player), 7Zip, UltraEdit, ClamWin (an anti-virus program), FileZilla, Gimp (an open source picture editor), Acrobat Reader, WarZone2100 (a real-time strategy game), On-screenkeyboard.*

Tests show that, the average ratio of the amount of data accessed during the start-up process to the whole capacity for the given software is about 21%.

Based on this observation, some frequently-accessed data is cached locally and its replace strategy is also based on the usage frequency.

At the first time, the cache is empty and then the run speed is fairly slow. During the run time, the cache is fulfilled according to the usage frequencies of data. Then for the following runs, the performance is improved because reads will be partly hit in the local cache.

2) *Data pre-fetch*

Besides the local cache, pre-fetch is another potential method to reduce the number of data access across the Internet. And its efficiency depends on the concrete access mode: for sequential accesses, it will be highly efficient.

We study the access behavior on any single file. For a given file, two arguments are defined: *a* is the ratio of the number of sequential reads to the total read-number, and *b* is the ratio between read amounts. The greater the value of *a* or *b* is, the better the effect of pre-fetch is. A file is sequential if and only if the values of *a* and *b* are both more than a threshold. Another conclusion from the analysis is that, for given software, its file-access-pattern is fixed regardless of its storage position. Then, we only adopt pre-fetch for these sequential files. In the current implementation, the threshold is set as 66% and the pre-fetch distance is 32KB.

IV. PROTOTYPE AND TESTS

We have implemented the prototype using VC 2005.

A. *Performance Tests*

1) *Test Methods*

Two types of performance metrics are measured.

- Start-up time

[1] The Microsoft Office applications can also be made portable by us, but it cannot run on the virtual file system because of some bugs of DOKAN. As stated at http://dokan-dev.net/.

The application start-up time is the key metric of the prototypes' usability: the time it takes for applications to begin to respond to user-initiated operations is a measure of what it feels like to use the system for everyday work.

In our test, CreateProcess is invoked to launch the given software, and then another API WaitForInputIdle is used 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 logged as the start-up overhead.

- Run time

A special program is used to record the user's inputs of the keyboard and mouse and replay them after start-up. Based on this tool, we design scripts to control software to complete a series of operations, which looks like triggered by a real user. For example, for the word processing software, one document is created and compiled for several seconds and saved before termination. Moreover, between any two continuous operations, some random waiting time (less than one second) is inserted to simulate the human's behavior.

The elapsed time is logged as the run time.

2) Test Environments

The client platform is a Windows Vista PC, equipped with 2 GBytes DDR2 SDRAM and one Intel Core Duo CPU (1.86GHz). The hard disk is one 160 GBytes SATA drive.

It uses one 100M Ethernet adapter to access the Internet.

The client machine should cross the Internet for data. So where to place the server is decisive for the performance. Two cases are considered.

In Case 1, it is assumed that some edge server can be found to provide the download service, therefore the web server is located in the CERNET (Chinese Education & Research Network, is the second largest network backbone in China.) as well as the client PC. This is a common case now: Content Delivery Network has been widely used for software downloading. As disclaimed by *Akamai*, the world leading CDN provider, most visit requirements can be fulfilled by some edge server(s) just a single-hop away.

The network throughput between the client and this server is about 1.89MBps and the average response time is about 6ms, which are tested by Qcheck, a free and professional network benchmark program.

In Case 2, the server is located outside the CERNET. The throughput is 998KBps and the response time is 32ms.

Two Windows 2003 servers, equipped with one Intel Core 2 Duo E4500 CPU (2200MHz), 2 GBytes DDR2 SDRAM, and one 240GBytes SATA II disk, are used for these two cases respectively.

3) Test cases

Case 1: The original start-up time / run time (portable software is saved in one local disk).

Case 2: The start-up time / run time based on the user-space file system (portable software is saved in one local disk, which is also mirrored as a virtual drive; then the software is launched through this drive.)

Case 3: The start-up time / run time based on the user-space file system for the remote server located inside the CERNET; no cache, no pre-fetch;

Case 4: The server is located inside the CERNET; the cache hit ratio is 20%, no pre-fetch;

Case 5: The server is located inside the CERNET; the cache hit ratio is 33%, no pre-fetch;

Case 6: The server is located inside the CERNET; the cache hit ratio is 50%, no pre-fetch;

Case 7: The server is located inside the CERNET; the cache hit ratio is 66%, no pre-fetch;

Case 8: The server is located inside the CERNET; the cache hit ratio is 80%, no pre-fetch;

Case 9: The server is located inside the CERNET; no cache, the pre-fetch size is 32KB;

Case 10~Case 16: The server is located out of the CERNET and other conditions are the same as those of Case 3~Case 9.

Some software cannot be controlled by our automation method; therefore the software number in the run-time tests is less.

We present the detailed results from Figure 1 to Figure 5. To present clearly, all results have been normalized, compared with the values of Case 1.

4) Test results

For the start-up time (Figure 1 ~ 3), the user-space file system itself introduces less than 96% extra overheads (comparing Case 2 with Case 1), as the file system causes more context-switch operations.

The exception is WarZone2010 (in Figure 1), whose extra overheads are much more because its access-pattern is special: it reads one sequential file many times while each time only two bytes are fetched. Similarly, when our file system is based on the remote servers, its start-up time becomes much longer: about 7720% extra overheads in Case 3 and 26590% in Case 10 (compared with Case 1). But pre-fetch is very efficient for this game, the speed-up ratios are 11 in Case 9 and 17 in Case 16, compared with Case 3 and 10 respectively.

For the other software (in Figure 2 and 3), our system introduces about 890 % extra start-up time on average in Case 3 and 1452% in Case 10 (compared with Case 1). When the cache-hit ratio is 80%, the corresponding results are 88% and 264%. So, the network performance largely determines program behaviors; and local cache is a highly-efficient method to improve the performance, except for some tiny software because their disk-IO overhead is so small that the IO sub-system affects its whole performance little. For pre-fetch, the speed-up ratios are 1.25 in Case 9 and 1.44 in Case 16 respectively.

As mentioned in Section 3.3, the most frequently-used files only occupied a limited ratio of the whole capacity. In our tests, one local cache of 140MB can reach the hit-ratio of 80%. Then, for much frequently-used software, after several runs, their performance on our system is really acceptable.

For run-time tests (in Figure 4 and 5), the results are better: because the waiting time overlaps the background transfer operations and much data required has been fetched during start-up, about 11% extra overheads in Case 8 and 18% in Case 15 are introduced. Because our scripts only complete some simple and common operations, this result is for reference only.

V. RELATED WORK

A. *Cloud Computing*

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. As [16] said, there are three types of Cloud Computing, which are classified based on the level of abstraction presented to the programmer.

Amazon EC2 [17][18] is at one end. It presents a virtual computing environment, allowing customers to launch instances with a variety of operating systems, manage network's access permissions, and run image, which is compatible with legacy desktop software completely.

Google AppEngine [19] is at the other extreme. It is an application-domain specific platform, which just hosts traditional web applications. As we know, this mode is incompatible with the desktop software, which asks developers to write new applications.

Accordingly, there are mainly two types of Desktop cloud. The first is based on the thin-client computing mode, which hosts VMs running desktop systems on the data center and users access them through some RDP. IBM Smart Business Desktop Cloud [20], VMWARE's ThinApp [21] and Citrix's XenAPP belong to this catalog.

The second refers to the Web-Application based [8] [9]. It provides a desktop-like GUI on the browser, which contains many web applications.

Another important service provided by the cloud computing is cloud storage, like Amazon's S3, Microsoft's Live Skydrive [22] and so on. Cloud Storage delivers virtualized storage on demand, over a network based on a request for a given quality of service (QoS).

B. *Software Streaming*

Virtualization has been deployed for software streaming. A solution is Progressive Deployment System (PDS) [23], which is a virtual execution environment and infrastructure designed for deploying software on demand. Another practical solution is Microsoft's SoftGrid [24]. SoftGrid can convert applications into virtual services that are managed and hosted centrally but run on demand locally.

Our previous work [15] also provides a solution for software streaming based on lightweight virtualization and p2p transportation technologies. Compared with [15], this work is based on the cloud storage and a new user-space file system is introduced. Another previous work [16] of ours is about how to fast deploy desktop software in a VM-based cloud environment (like EC2).

VI. CONCLUSION AND FUTURE WORK

This paper presented a solution to convert the existing Windows desktop software to on-demand application stored on the cloud storage, which could be regarded as a mode of SaaS. As we know, this is the first prototype of such a solution. Two main technologies were used: the first was OS-level virtualization, which made legacy software portable; and the second was a user-space file system that provided the user a transparent interface to access them. In addition, some access control mechanisms were implemented.

Owing to the local cache and pre-fetch mechanisms, tests showed that, for much frequently-used software, their performance was acceptable with a limited local cache.

ACKNOWLEDGMENT

This research is supported by Chinese National 973 Basic Research Program under Grant 2007CB310900, Chinese National 863 High Technology Programs under Grant 2008AA01A201.

REFERENCES

[1] Federica Troni and Michael A. Silver. Use Processes and Tools to Reduce TCO for PCs, 2005- 2006 Update. Gartner Group.

[2] IBM Virtual Infrastructure Access Service Product. https://www-935.ibm.com/services/au/gts/pdf/end03005usen.pdf. 07.07.2010.

[3] Windows Server 2003 Terminal Services. http://www.microsoft.com/windowsserver2003/technologies/terminalservices/default.mspx. 07.07.2010.

[4] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing, Internet Computing, IEEE, Volume.2, No.1. January, 1998, pp.33-38.

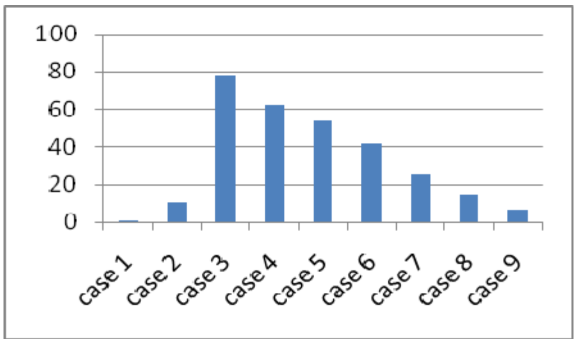
[5] CITRIX, http://www.citrix.com/lang/English/home.asp. 07.07.2010.

[6] Kirk Beaty, Andrzej Kochut, and Hidayatullah Shaikh. Desktop to cloud transformation planning. Proceedings of 2009 IEEE International Symposium on Parallel & Distributed Processing. Rome, Italy, May 23-29, 2009, pp.1-8.

[7] Albert Lai and Jason Nieh, On the Performance of Wide-Area Thin-Client Computing. ACM Transactions on Computer Systems (TOCS), Volume 24, Issue 2. May 2006, pp. 175-209.

[8] iCloud, http://icloud.com/. 07.07.2010.

[9] Lifehacker, the Full-Screen Firefox Cloud Desktop. http://lifehacker.com/5256657/the-full +screen- firefox-cloud-desktop. 07.07.2010.



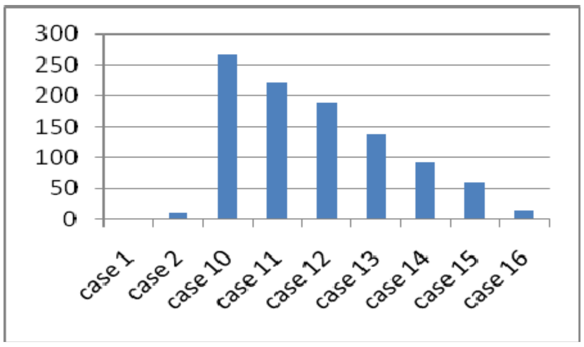


Figure 1. The start-up time of WarZone2010 (the left part is the case that the server is inside).

[10] Dokan, user mode file system for windows. http://code.google.com/p/dokan/. 07.07.2010.

[11] Filesystem in Userspace. http://fuse.sourceforge.net/. 07.07.2010.

[12] Installwatch. http://tejasconsulting.com/open-estware/feature/ installwatch.html. 07.07.2010.

[13] Galen Hunt and Doug Brubacher, Detours: Binary Interception of Win32 Functions, Proceedings of the Third USENIX Windows NT Symposium, July, 1999, pp.135-144.

[14] Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/. 07.07.2010.

[15] 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, 2008, pp. 133–144.

[16] Youhoi Zhang, Gelin Su, and Weimin Zheng: "On demand mode of legacy desktop software and its automatic deployment for Cloud-Computing Environment", Proceedings of the Sixth Workshop on Grid Technologies and Applications (WOGTA 2009), 18-19 Dec 2009, Taitung, Taiwan, pp.25-31.

[17] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, et al. Above the Clouds: A Berkeley View of Cloud Computing Export. Technical Report. 10 February 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html. 07.07.2010.

[18] Amazon Elastic Compute Cloud. Developer Guide. http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide. 07.07.2010.

[19] Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, et al. Dynamo: Amazon's highly available key-value store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, ACM Press New York, NY, USA, 2007, pp. 205–220.

[20] Google App Engine, http://www.google.com/apps/intl/en/business/ index.html. 07.07.2010.

[21] Desktop cloud computing services. http://www-935.ibm.com/services/us/index.wss/offering/eus/a1026737. 07.07.2010.

[22] VMWARE ThinApp——Agentless Application Virtualization Overview. White Paper. Available at http://www.vmware.com/files/pdf/thinapp_intro_whitepaper.pdf., 07.07.2010.

[23] Windows Live SkyDrive, http://skydrive.live.com/, 07.07.2010.

[24] Bowen Alpern, Joshua Auerbach, Vasanth Bala, Thomas Frauenhofer, Todd Mummert, and Michael Pigott. PDS: a virtual execution environment for software deployment. Proceedings of the First ACM/USENIX international conference on Virtual execution environments, March, 2005, pp. 175-185.

[25] http://www.microsoft.com/systemcenter/softgrid/default.mspx. 07.07.2010.

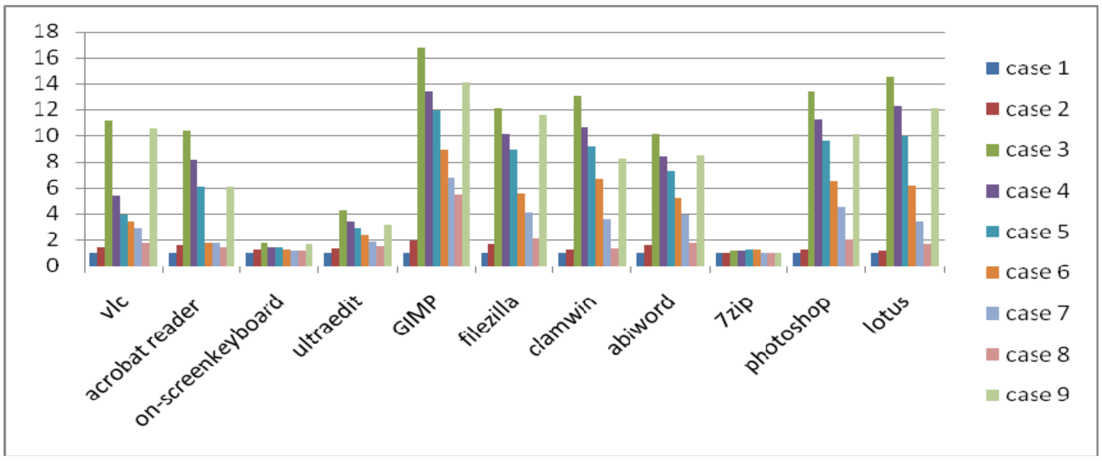


Figure 2. The start-up time (The server is inside and values have been normalized).

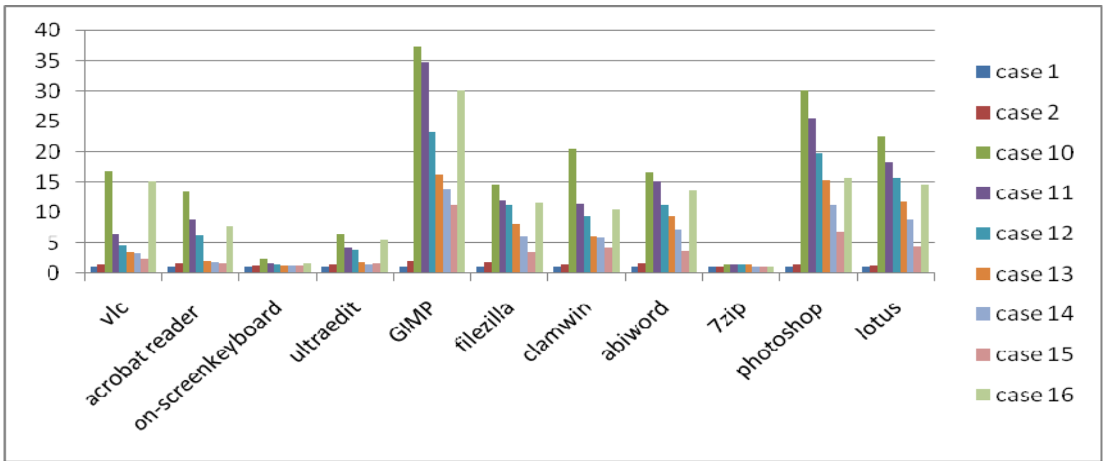


Figure 3. The start-up time (The server is outside and values have been normalized).

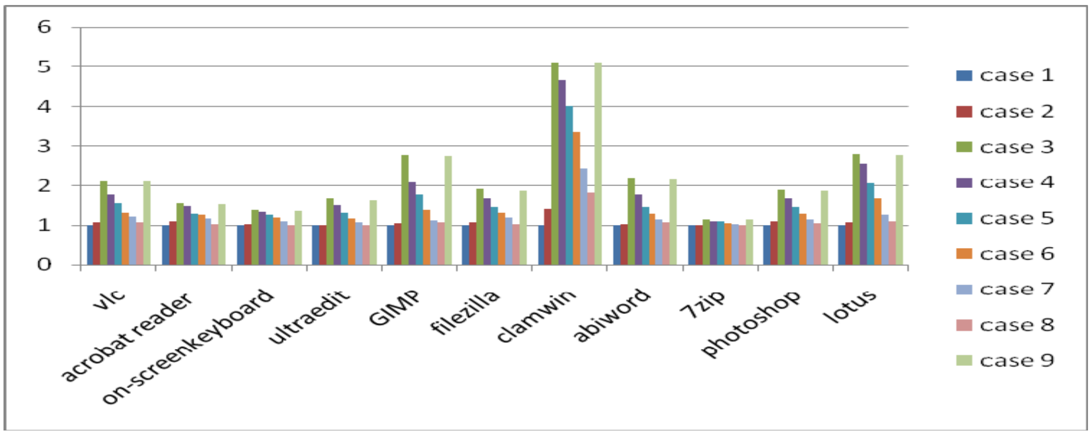


Figure 4. The run time (The server is inside and values have been normalized).

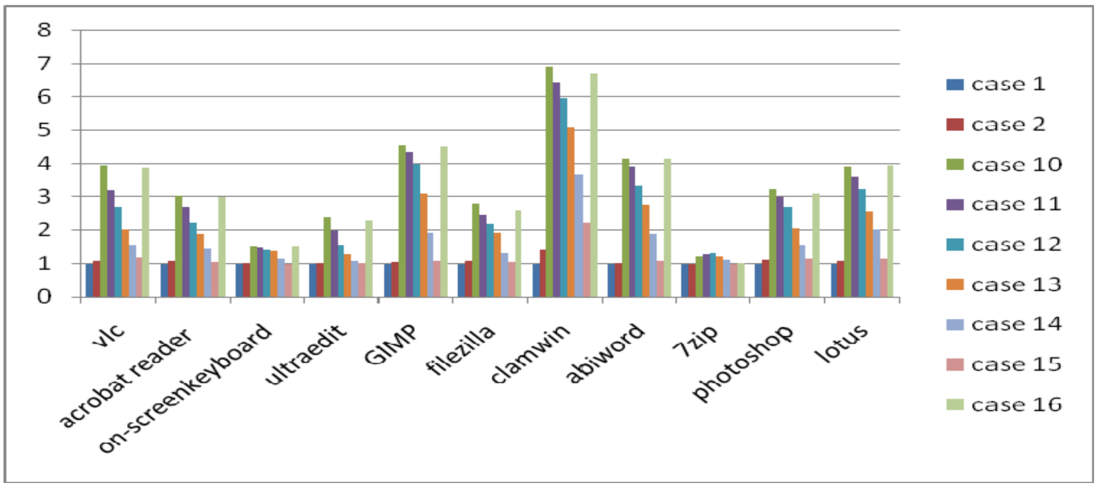


Figure 5. The run time (The server is outside and values have been normalized).