Portable Desktop Applications Based on P2P Transportation and Virtualization

Youhui Zhang, Xiaoling Wang, and Liang Hong – Tsinghua University, Beijing, China

ABSTRACT

Play-on-demand is usually regarded as a feasible access mode for web content (including streaming video, web pages and so on), web services and some Software-As-A-Service (SaaS) applications, but not for common desktop applications. This paper presents such a solution for Windows desktop-applications based on lightweight virtualization and network transportation technologies which allows a user to run her personalized software on any compatible computer across the Internet even though they do not exist on local disks of the host.

In our approach, the user's data and their configurations are stored on a portable USB device. At run time, the desktop applications are downloaded from the Internet and run in a lightweight virtualization environment in which some resource-accessing APIs, such as registry, files/directories, environment variables, and the like, are intercepted and redirected to the portable device or network as needed. Because applications are ''played'' without installation, like streaming media, they can be called ''streaming software.'' Moreover, to protect software vendors' rights, access control technologies are used to block any illegal access. In the current implementation, P2P transportation is used as the transport method. However, our design actually does not rely on P2P, and another data delivery mechanism, like a dedicated file server, could be employed instead to make the system more predictable.

This paper describes the design and technical details for this system, presents a demo application and evaluates it performance. The proposed solution is shown to be more efficient in performance and storage capacity than some of the existing solutions based on VM techniques.

Introduction

Today, a desktop PC is usually available at home, at work and at some public places. However, the diversified PCs, although running the same OS, cannot provide the user with her unique personalized desktop environment, which includes personal documents, frequently-used applications and their customizations.

Several solutions to this dilemma have been proposed. The first approach is thin-client computing [1]. CITRIX [2] is such a framework that allows a variety of remote computers to connect to a Windows NT terminal server to access a powerful desktop and its applications. And the remote computers only execute the graphical interface of the applications. For this solution, users' feeling would be bad when it is employed across the Internet, because of the long network latency.

Web-based application [3] is the second solution. 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.

The third approach is the virtual machine-based solution. For example, IBM's SoulPad [4] carries an auto-configuring OS with a virtual machine monitor on a small USB device. The computer boots from the device and launches the virtual machine, thus giving the user access to her personal desktop. Moka5 [5], a commercial product, provides a similar solution. It uses the local machine's installed operating system (Windows XP or Vista) and runs the virtual machine (under VMPlayer) there. Recently, Moka5 also provided a network-based solution that users can download the VM image and use it on-the-fly. Collective [6] also propose a virtual-machine-based solution. It runs virtual machines on a PC and downloads OS images from Internet, rather than from the portable device.

The VM-based solution is promising because it is fully compatible with the mainstream desktop environment. However, virtual machine will introduce fairly substantial performance overheads. On one hand, as mentioned in [4], the VMWare-based configuration of SoulPad incurred a 26-29% increase in response time for Office Productivity and Internet Content Creation applications. And on the other, although the Xen virtualization environment is claimed to incurred only a 2-8% [7] slowdown in general owing to its paravirtualization architecture, it is necessary to modify the guest OS to take advantage of this feature. Now, some hardware-based virtualization acceleration can be used to deploy VM transparently, but this acceleration can introduce more overhead compared with the full software-based solution [8], especially for workloads that

perform I/O, create processes, or switch contexts rapidly.

Moreover, carrying or downloading a whole VM-based OS is not economical. For example, a complete Windows XP installation occupies more than 1.5 GB disk space, which is too bulky for a tiny USB flash disk or the current Internet access rate.

It is believed that application virtualization will be the next frontier, and Software-As-A-Service (SaaS) is a promising deployment mode for software. Therefore, owing to the prevalence of portable storage devices and the ubiquitous network access, we propose a solution that combines the two technologies for any user to "play" her personalized software without installation anytime and anywhere conveniently. In this solution, the user's data and applications configurations are stored on a portable USB device. While at run-time, the desktop applications are streamed: downloaded from the Internet on-the-fly and run in an OS-level virtualization environment without installation.

In contrast to hardware-level virtual machine technologies, OS-level technologies have the virtualization layer positioned between the operating system and application programs. Every virtualization environment shares the same execution environment as the host machine, and only retains any divergences from the host as the VM's local state. Therefore, such an environment can have very small resource requirements and thus introduce only very limited overhead.

Under our approach, only personalized data and documents are stored on the portable device. Each personalized application runs in an OS-level virtualization environment that is layered on top of the local machine's Windows OS. At run-time, the virtualization environment intercepts some resource-accessing APIs, including those that access system registry, files/directories, environment variables, and the like, from these applications, and redirects them to those resources stored on the portable device or network as needed. For example, when one application accesses *My Documents, Desktop* or some other personalized configuration, it will reach the corresponding resources on the portable device instead of the local disk of the host machine.

In user's view, she can access her personalized applications and data conveniently on any compatible computer, even though they do not exist on local disks of the host. Moreover, because of the commonality of frequently-used applications, P2P transportation and the local look-aside cache can be used to improve the access performance.

Compared with the VM-based method, the storage capacity occupied by our solution is much smaller than other VM-based approaches, and the performance overhead introduced by virtualization is small. Of course, it relies on the host computer to provide the hardware resource as well as the compatible OS environment.

In addition, protecting software vendors' rights is a necessary prerequisite for a successful deployment mode. In our solution, only those processes running in the virtualization environment can have the right to access application files, and the user has to login a remote server before she launches the environment. Then, some mature billing mechanism can be employed, and illegal access will be prohibited.

In this paper, we first present the overview of our design, followed by implementation details, including how to run a software without installation, how to make software streaming in a friendly usage-mode based on file system filter and network transportation technologies, as well as the access control implementation. Then the prototype is introduced, and performance tests are presented. Finally, we present our conclusions, including comparison to of related works.

Design Philosophy

In our solution, a user can use her applications when plugging the portable storage into a Windows PC; and her personalized configurations, including *Desktop, My Documents, Favorites. Browser History, Temporary Internet Files, Cookies*. In addition, applications customizations (e.g., for a web browser, *Default Homepage, Internet Settings, Download Directory, Toolbars' Positions, Recently Opened Documents*) are restored just as if she were using her unique desktop environment on her home computer. Moreover, no trace of her work will be left behind on the host PC.

To provide these features, the following technical challenges should be overcome:

- How to run a Windows application without installation, including how to restore applications' customizations and user's personalized configurations transparently and how to leave no trace on the host PC.
- How to implement software streaming.
- How to enforce software licensing.

We will discuss each point separately below.

How to Run an Application Without Installation

Most Windows applications need to be installed before they can run normally. Even for an application that can work without installation, most of them save their customizations into the system registry and/or into configuration files located in some system folders. Then an application can be regarded as including two parts: Part 1 is all of the files and folders and registry keys and environment variables created by its installation process, and Part 2 is the customization produced during the run time.

To conquer the challenge, we have to make Part1 portable and enable the application to run in a sandbox where it can access and store the data associated with Part 2 in an isolation mode. The OS-level virtualization technologies are used to achieve these functions.

Therefore, this work consists of two tasks:

1) An installation snapshot, and
2) A runtime system design.

which are presented as follows.

Installation Snapshot

To make Part 1 portable, the modifications made by the application's installation process must be captured. There are usually two types of modifications: registry contents and files/folders. InstallWatch [9] is used to complete the task. It is a system monitoring tool that tracks changes to the computer's hard disk, registry, and .ini files when a new application is being installed.

In our implementation, a target application is installed on one clean Windows XP system. At the same time, InstallWatch is running to log those files created or modified in this process, as well as registry additions and modifications. Then, the files/folders created or updated are copied to a separated folder, called the *private folder*, while the directory hierarchy is retained. Similarly, the contents of the added/modified registry keys are collected to be stored in a separated file, which is called the *private registry file*.

Runtime System

Now we have captured Part 1 of one application, and the second issue is how to make it accessible by the application's executable file. *API Interception* is employed to do so.

API interception means to intercept calls from the application to the underlying running system and reinterpret the calls. It is usually used to extend existing OS and application functionality without modifications of the source code. Detours [10], a library developed by Microsoft Research Institute, are used to intercept those APIs employed to access the system registry and files/folders.

Owing to Detours, we have built *Wrapper APIs* that inject a wrapper DLL into the target process virtual address space as described in [10]. For example, when an application uses an interpreted WIN32 API to access the system registry, the wrapper API will be called firstly. Our injected code deals with this request before the original API – If one of the registry keys contained in Part 1 is to be accessed, the injected code can return the corresponding value from the *private registry*; otherwise the original API will be called to access the system resource. For requests for files and folders, a similar mechanism is adopted.

This makes the portability feasible because the customizations and files of an application have been isolated from the OS. And any modification happened during the run time will be store into the private registry or the private folder instead of the system's default position, while read operations are done wherever the content exists. Therefore no trace of application execution will be left behind on the host PC.

Streaming Software

Once we have succeeded in making the user-specific application state portable, the pivotal issue becomes where to locate the installation snapshot. Putting it on the portable device with the user's private data is not a bad idea. However, this approach disrupts traditional approaches to software licensing. The dominant licensing model for PC software permits use of the software ''on a single computer,'' not ''on a single USB drive.'' However, if they are placed on a network for downloading on demand, some mature billing mechanisms, like those employed in SaaS, can be used to protect vendors' rights.

Therefore, in the current implementation, applications are stored on the Internet and downloaded on demand while users' private data is still kept on the portable device for privacy.

However, it can be difficult for ordinary users to run applications transparently and quickly when streaming software is used. Thus, we have implemented a solution based on file system filter driver [11] technology.

A file system filter driver intercepts requests targeted at a file system. By intercepting the request before it reaches its nominal target, the filter driver can extend or replace functionality provided by the original target. Owing to this feature, a file system virtualization mechanism, which we call an *anchor file*, is achieved to present users a friendly interface.

For example, when the user launches an executable file, z:\abc.exe, the shell program will send a serial of IRPs (I/O Request Packets) to the file module of OS, which is intercepted by our filter driver. Then the driver will deal with these IRPs and return results directly rather than transfer those to the target. So, while it looks like the z:\abc.exe is being accessed, in fact the filter driver has transformed and redirected these requests to an Internet location. z:\abc.exe is just an *anchor*.

Those file system visits generated while the application is running will be handled in a similar way. In other words, our filter driver will judge whether the installation snapshot should be accessed or not. If the answer is yes, the drive will redirect the requests to the remote location. Moreover, because of the commonality of frequently-used applications, in the current implementation P2P transportation and look-aside cache are used to improve the access performance. More details about the virtualization approach and4 software streaming are presented in the next section.

Access Control

Copyright violation is a real problem that hampers the software industry's progress. In this play-on-demand mode, it is especially important to prevent any illegal access; otherwise, running software without installation will be good news for illegal users.

Therefore, before the user launches our own shell program with the file system filter, she has to identify herself and login to the server.

Moreover, as mentioned previously, users can access the application files just like they are using the local file system, so how to prevent the illegal download is another key consideration. Otherwise, a user can copy applications to her local disk and use them without login.

In our solution, an access control mechanism is implemented to protect some essential files (such as the executable and DLL files of applications). It works by allowing only certain processes in the virtualization environment (like our own shell program) to access those files. If the user attempts to use another program (for example, explorer.exe) to copy one essential file out, our filter driver will intercept those IRPs to 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.

Implementation

Virtualization Running Environment

Because there are some existing similar OS-level implementations, like Progressive Deployment System (PDS) [12] and Featherweight Virtual Machine (FVM) [13], only a brief overview of our implementation is presented here.

The captured installation snapshot can be divided into six categories:

- Added registry set. It contains the entries created by the installation.
- Modified registry set. It contains the entries whose values or sub-keys have been modified or deleted.
- Deleted registry set. Those entries deleted by the installation are included. So that the entries in this set will not be accessed during the run time.
- Added file set. It is similar to the added registry set, including new files and new folders created by the installation.
- Deleted file set. It is similar to the deleted registry set.
- Modified folder set. For any file or folder in the added/deleted file set, its parent folder will be included in this set.

These six sets are not fixed. They may be modified when the application runs.

The private registry is a complete registry system that provides access APIs just as Windows OS does. It works like a small subset of WINE [14], which is an open source implementation of the Windows API on top of UNIX. In other words, our private registry provides another implementation of registry APIs.

When the target application is launched, the six sets and registry contents will be initialized. The absolute path is used to identify a single registry key and such a map structure is maintained, which can map a handle of any opened key to its full path. For example, when one program opens the registry key *"HKCR*\.doc" the interception code will map the returned handle to the path string. Then every time this handle is used, its full path can be referred to.

The registry-related APIs below have been wrapped.

Open/Create a Key (RegOpenKeyEx / RegCreateKeyEx)

Arguments of RegOpenKeyEx contain the handle of an opened key and a null-terminated string indicating the name of the subkey to open. Based on the above-mentioned map, we can identify the absolute path name for the key to open, and thus find out which set it belongs to. If the key is in the added registry set, it will be opened in the private registry; if it is in the modified registry set, it will be opened in both the private registry and the system registry. The two handles referring to both keys are stored in another handle-map structure that will be used in subsequent invocations, and the system handle is returned to the application. If the key is in the deleted registry set, this invocation fails. If the key is not present in any of the registry sets, then the original system API will be used.

Creating a key is often regarded as an open operation except that it will create a new key when the key does not exist. In this case, the new key is created in the private registry and its full path is inserted into the added registry set. In addition, its parent key is moved into the modified set.

Set Value of a Key (RegSetValueEx)

Any new value is always saved in the private registry, and its parent key will be moved to the modified set (the exception is that it has been in the added set).

Query/Enum Value of a Key (RegQueryValueEx / RegEnumKeyEx / RegEnumValue / RegQueryInfoKey)

If the key is in the added set, it will be queried only in the private space. Similarly, the query will be done in the system registry if the key does not exist in any predefined set. The most complicated case occurs when the key is in the modified set. Both of the private and the system registries should be queried, and the results will be merged before return. It means, if there is any duplicated key, the private one should be presented instead of the system one. The same method is adopted for accesses to key values.

Close Key. (RegClosekey)

When closed, the key's corresponding handles must be released and removed from both maps.

Delete Key/Value (RegDeleteKey / RegDeleteValue)

If the key is in the added or the modified set, it will be deleted from the private registry and inserted into the deleted registry set. If it exists in the system

registry, it will not be actually deleted. Instead we add it into the deleted registry set. For subsequent accesses later, we first check whether the key is in the deleted registry set; if so, nothing is done but an error code is returned. Deleting a value is handled in the analogous way.

In summary, the principle is that any modification is always saved in the private space while any query will return the combination of results from both registries. In addition, if there is any duplication, the private registry has the higher priority.

Since the *private folder* is located in a portable device whose drive letter will change on different hosts, the registry value containing such a full path is modified to represent the current position before return.

For APIs that access the file system, a similar method is adopted because folders can be regarded as registry keys and files can be regarded as values. Moreover, the copy on write (COW) mechanism is applied at the whole file level when a file is modified during run time.

For environment variables, the solution is much simpler. As we know, a process will inherit environment variables from its parent process. In our implementation, a shell program is in charge of launching any target application. Therefore, it can set any application-specific variable before launching the target.

Streaming Software Based on File System Filter Driver

In order to explain the implementation clearly, we begin with an example operation.

The user inserts a USB device into the Windows host and then our own shell program (with the filter driver) is loaded automatically by an auto-run mechanism. Thereafter, any file system I/O operation will be intercepted.

The portable application is located on the USB device, in z:\program files\abc\. But in fact the physical position of this folder is empty and the real application is stored on a remote file server.Our shell program presents a shortcut of this application to the user.

When the user clicks the shortcut to launch the application, the shell program will send some IRPs, including IRP_MJ_CREATE, IRP_MJ_DIRECTORY_CONTROL, IRP_MJ_READ, IRP_MJ_CLEANUP and IRP_MJ_CLOSE.

IRP_MJ_CREATE is used to open the folder/file and then one or more IRP_MJ_DIRECTORY_CONTROL IRPs are issued to query the folder/file information, followed by lots of IRP_MJ_READs to read the real data. The last two IRPs end the operation series.

During this process, our filter driver handles all IRPs issued to this folder/file entirely. Specifically, it simulates the target to return folder/file information (metadata) and data that are obtained from the network server in reality. So, in the shell's – and therefore the user's – view, this application can be launched as a local one.

Work Flow. During the start-up stage, our shell program connects to the remote server to download all metadata of the anchor files and folders. Because this file system does not exist physically on the portable device, these metadata must be retrieved first for presentation. Fortunately, their total size is very small, which only delays the startup time for a few seconds.

All metadata are transferred to the filter driver, which had learned previously which folders/files are to be managed and what should be returned when they are accessed.

As it runs, the portable application will issue many IRPs and which will then be intercepted by this driver. If an IRP is issued for metadata, it will be handled by the driver itself. IRPs for file data will be retransferred to another user-level module, the *client module*, which will download the data from the network.

Details of IRP handlers. Because no Fast I/O is supported in our driver, any I/O request will generate IRP operations. The IRPs below are intercepted (for more info for IRP and Fast I/O, please refer to [11]).

CREATE IRP

When any file or folder is to be accessed, this is always the first IRP. In its dispatch function, the full path of the target is obtained using the *ObQueryNameString* API at first. If it is an object belonging to the install snapshot, the target's file object address will be stored into an internal hash table. We know all subsequent IRPs will carry their object addresses, so this table can be used to check whether an IRP should be handled by our dispatch functions or by the system default route.

DIRECTORY CONTROL IRP

This IRP contains two subtypes: IRP_MN_NOTIFY_CHANGE_DIRECTORY and IRP_MN_QUERY_DIRECTORY. The first one can be skipped in our solution. The second is used to get information about all files and subfolders of the target directory. The corresponding metadata obtained at the startup stage will be returned.

QUERY INFO IRP

It is used to query file metadata and can be handled like the preceding one.

READ IRP

Its dispatch function will put the IRP into its internal waiting queue and simultaneously signal a waiting event to notify the client module that there is some request to deal with. The module then reads the request using the *DeviceIoControl* API and downloads the requested data from the network. After downloading, it will call *DeviceIoControl* API to send the data

to the driver. On receipt of the data, the filter completes that pending IRP.

In the current implementation, the driver waiting queue can contain up to 5000 requests, which is enough for most common use cases.

CLEANUP/CLOSE IRPs

This pair of IRPs always appears as the end of a series of operations to one target. They remove the responding target's entry from the hash table when present. At the same time, its dispatch function will notify the client module that the access series finished.

P2P Transportation and Optimization. The portable application can be downloaded through a single dedicated server or it can be downloaded from multiple servers in parallel to speed up the process. Since the implementation of a dedicated server is rather straightforward, we use libtorrent [15], an open source library that implements BitTorrent [16] protocol, to achieve P2P transportation. Because P2P is a mature technology, its technical details are skipped here.

In addition to speed up the access performance, the client module creates a look-aside cache on the USB device. We know that in BitTorrent a downloadable file is divided into many fixed-length pieces, with each piece indexed by its hash value. So, we use a content-based addressing scheme, similar to several previously discussed in the literature [17] to implement a nonvolatile cache. When a particular piece is accessed, the local cache will be first, avoiding a network operation if the data is already present. When the virtual environment exits, all cached data will be stored for the next time.

The content-based addressing scheme brings us three other benefits:

1. It can reduce the storage requirements because content-addressing storage is also a good compression mechanism.
2. Store files in this mode rather than storing them directly can prevent users from recovering application files from the cache.
3. Hash value-based indexing allows any tampered content to be detected easily.

In addition, some pre-fetch technology is employed. Specifically, when one piece of a file is requested, the whole file or even the whole application to which that file belongs will begin to be downloaded. Obviously this pre-fetch can be highly efficient.

Finally, the client machines are able to serve out the applications themselves once they have them cached, which is why it is called P2P.

So far, we have discussed only read operations. So how does our approach handle write operations?

We know that most snapshot files are read-only; only configuration files (including *private registry files* mentioned earlier) will ever be modified. For these files, the copy-on-write strategy is employed: when a file is modified, it will be downloaded completely to the local USB device, and any subsequent modification will happen locally. Note that there is no coherency problem to worry about since modified files belong to users' personal customizations.

Access Control Based on WMI Service

Windows Management Instrumentation (WMI) [18] is the infrastructure for management data and operations on Windows-based operating systems. It can be used to get information of all current processes (using *CreateToolhelp32Snapshot* API) and can send corresponding notifications when any process is created or terminates (using ExecNotificationQuery Async API).

Based on these features, the client module creates the process-hierarchy structure at startup. During the application's running time, any new process's ID and their parents can be detected and captured. In this way, a whole process hierarchy can be maintained in real time.

The access control policy is based on process ID. In our design, the root of the hierarchy is the process ID of our shell program, which can access all installation snapshot files while any processes outside of the hierarchy is forbidden. Furthermore, a process and its offspring can only access the files belonging to its own application. Therefore, when the client module accepts a read request from the filter driver, it will determine whether the original sender is an authorized one or not based on the process hierarchy. Only legal requests are allowed.

The Prototype

Overview

A prototype implementation of our solution has been completed using VC 7.0. And many existing applications can be made portable, including *MS Word 2003, MS Excel 2003, MS PowerPoint 2003, Lotus Notes, Photoshop, Internet Explorer 6.0, Outlook Express 6, Winzip, UltraEdit, FlashGet, Bittorrent* and so on.

Of course, there are still some applications that cannot currently be made portable using our approach. For example, applications that have their own kernel modules are not supported because our solution only supports access- redirection for user-level resources. Another problem is that the current implementation does not consider Windows Side by Side technology (SxS) [19] and so some new applications like Microsoft Office 2007 and Adobe Reader 8 are not supported. With SxS technology, applications can install DLLs to version-specific directories and tell Windows what version of the DLL should be used when they load a DLL by that name. However, we plan to support this feature in the next version.

The whole work flow of the prototype is described in Figure 1. Initially, installation snapshots of applications are captured by the server and are stored

on the data server, which also functions as the tracker server for BitTorrent P2P transportation. When the application runs, the virtualization environment is running on multiple clients that are connected with each other for P2P data sharing.

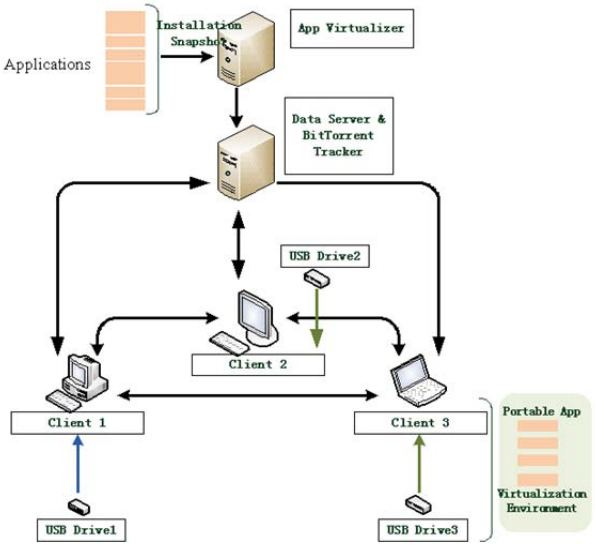

Figure 1: The work flow.

The shell program of this prototype is a *start-menu*-like GUI where the user can launch any portable application as she operates the system's Start menu. Of course, these applications are located on the portable device instead of being installed on the host.

Besides portable applications, most personal folders such as *My documents, My Desktop, Cache paths, Temporary Internet Files*, etc., are also portable. For example, when Outlook Express is launched from our prototype, its email boxes are located on the portable device, and other customizations, including the account info and the email signature, etc. are personalized. Moreover, all newly received emails, as well as modifications of its customization settings, are stored onto the portable device so that the program is enabled to be portable. In contrast, when Outlook Express starts from the system's start menu, all of its configuration settings belong to the owner of the host. Once the portable device is unplugged, no trace of work is left behind on the host PC.

In addition, there are some visual indications to users that they are running a portable application. For example, when the portable Outlook Express is launched, its title will be modified to "Outlook Express – Desktop2Go" by intercepting the *CreateWindow* API, reminding the user not to save newly created files to the local hard disk.

When the portal application exits, all modifications to the system registry and the file system are stored on the portable device. Therefore, when the application is launched again from a different computer, the user's latest modifications are present.

Additional Technical Details

Some Windows pre-installed applications, such as IE and Outlook Express, are deeply integrated into the OS, making it very difficult to separate them from the operating system. On the other hand, this also means that they are always available on a compatible host. For such applications, only their customizations and personalized data are made portable.

For example, when IE (located on the host system) is launched from our GUI, it will run in the virtualization environment and its registry APIs are intercepted. Then, when it accesses registry entries that store customizations, like *Home Page, Download Folder, Favorites, Browser History, Internet Temporary Files* and so on, the interception code will return values from the private registry so that the portable personalized customizations are implemented.

For registry entries, our principle is that any modification is always saved in the private space while any query will return the combination of results from both registries. In reality, lots of registry accesses can be skipped. Most applications do not write their implementation from stem to stern. Instead, many standard Windows components will be employed. For example, when an application shows an *open file* dialog, many registry accesses occur, but they are totally transparent to the application since they are invoked by system code. Therefore, registry accesses can be divided into two categories: application-specific ones that program developers complete intentionally and ones generated from system components. The latter can be ignored by the virtualization system.

For instance, Adobe Reader creates the key *"HKEY_LOCAL_MACHINE\SOFTWARE\Adobe\Acrobat Reader"* to save its configurations. Therefore, only those entries below this key handled in the manner described above, and all other registry accesses are left to the host system. Of course, differentiating the two types of access depends on the specific application. Fortunately, for Windows OS, some public publications, like [20], have explained which registry entries are system-related.

Our last point is about shell folders mapping. *TEMP, HOMEPATH, USERPROFILE, APPDATA*, and other system directories are called shell folders in Windows. Applications typically use them and consult the system registry in order to locate the current user's My Documents, Desktop, and other personalized folders. Because these folders are also portable, accesses to the related registry entries have to be redirected into the private space. Specifically, the following registry entries are also stored in the private registry:

- *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders*
- *HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders*
- *HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders*

- *HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders*

In this way, portable applications can transparently visit personalized folders located on the mobile device.

Performance Testing

Application response times are the key metric of our prototype's usability. The time it takes for applications to respond to user-initiated operations is a measure of what it feels like to use the system for everyday work.

The test platform is a Windows XP SP2 PC, equipped with 1.25 GB DDR SDRAM, one Intel Pentium 1.6 GHz CPU and the 100 Mb Ethernet. The hard disk is one Hitachi IC25N030ATMR04 Travelstar.

The Common Desktop Application (CDA) benchmark is used. It automates the execution of common desktop applications of the Microsoft Office suite and IE browser in the Windows XP environment. During this process, nine local documents (including three Word files, three Excel sheets and three PowerPoint presentations) are opened, edited and closed automatically, while three instances of IE are running as the background load. Finally, all applications are closed and the running time is logged.

In all test cases, the pre-fetch mechanism is enabled.

Test Case 1: Perfect Local Cache

In this case, the look-aside cache performance is assumed to be perfect: all reads are cache hits.

System Configuration	Workload	Normalization Value
Physical, IDE	39.6 s	1.000
Virtual, IDE	43.2 s	1.091
Virtual, USB	47.9 s	1.210

Table 1: Response times.

Table 1 shows the running time averages across three runs of the benchmark. The row labeled ''Physical, IDE'' represents our baseline configuration, applications running on the physical machine and from the internal IDE drive, meaning our environment is not deployed.

The row labeled ''Virtual, IDE'' corresponds to running applications in the virtualization environment, with the cache located on the internal IDE drive. It is intended to isolate the overhead of virtualization from the overhead of using an external drive.

The row labeled ''Virtual, USB'' corresponds to locating the cache on the PocketDrive connected via USB 2.0.

The results show that moving to a VM-based configuration with the cache on the IDE drive incurred a 9.1% increase in response time. Moving to a VM-based configuration with the cache on the USB drive incurred a 21.0% increase over the baseline. So, the overhead introduced by virtualization (including the user-level virtualization and filter driver) is much smaller than that due to using external storage for the cache.

Of course, perfect cache performance is not the common case. However, if the user always uses some limited portable applications and if the cache space is large enough, we believe the cache will behave very nicely.

Test Case 2: No Local Cache

For this case, our testing client is located in the China Education & Research Network (CERN) while the BitTorrent tracker server is placed in an IDC outside of CERN; and a P2P transportation environment is constructed. For more information about this infrastructure, please refer to [16]. In this case, the local cache is disabled. But this does not mean that there is no cache in the whole system: the default file cache of OS is still present.

We can identify three subsets of the runs for this test case:

- **One Remote Peer**. In this case, one remote peer (located outside of CERN) is running, which holds all portable applications. So, as the testing client starts to access files, it can download them from this peer as well as the tracker server.
- **One Local Peer**. In this case, one local peer (located in the same 100 Mb Ethernet-LAN as the testing client) is running, which holds all portable applications. So, as the testing client starts to access files, it can download them from this peer as well as the tracker server.
- **More Than One Peer**. In this case, two, four, or eight local peers (located in the same LAN as the testing client) are running, and each of them holds all portable applications. So, as the testing client starts to access files, it can download them from these peers as well as the tracker server.

The test results show that when there are only limited, remote peers, the system performance is fairly slow compared with the baseline. However, we believe in reality the situation will be not so pessimistic:

- First, in our observation, only the startup process of an application shows slowdown, and when the application's GUI is presented, the operation delay is acceptable. Because it is difficult to accurately define what the startup time is and we cannot separate it from the whole operation time, we can only present our subjective feelings here.
- Secondly, because most users access only a limited number of frequently-used applications, we can expect the local cache to exhibit a high hit ratio.

- Finally, in our tests human response time was not emulated. However, in reality, the natural delays that occur when people are working are helpful to hide the background download latency.

System Configuration	Workload	Normalization Value
Physical, IDE	39.6 s	1.000
One Remote Peer	201.9 s	5.098
One Local Peer	82.3 s	2.078
Two Local Peers	65.3 s	1.649
Four Local Peers	52.2 s	1.318
Eight Local Peers	50.9 s	1.285

Table 2: Response time.

Discussion

Software Licensing

Virtualization disrupts traditional approaches to software licensing. For instance, the arguably dominant licensing model for PC software permits use of the software ''on a single computer.'' Is it necessary to modify it to ''for one person''?

Alternatively, we can extend this solution to an Internet-based service, where the private space will be located in a network server and accessible to every connected computer. Then, any access to the applications can be logged by the service-provider, which can be used for accounting.

Security and Privacy

Our solution does not write any customization to storage on the host PC. This isolation can keep the OS pristine, helping prevent and contain security breaches and infections.

Unlike some VM-based methods, this solution works at OS-level and highly depends on the host OS, so user should trust the host before operating on it. On the other hand, our VM-based solution is safer in some ways than using an unknown PC since it does not run any software previously installed on the host and starts the host from a known power-down state. However, if the local BIOS is compromised, it is equally unsafe as using the host directly.

Another open question is whether our solution is safe or not if the host has already been infected by some virus. Because our access control mechanism can prevent any illegal access to the protected application files, it seems that the virus can not impair them.

Finally, we believe it is necessary to construct a trusted chain between the user and the host to solve this problem completely, which depends on the prevalence and availability of trusted computing.

After these problems are solved, we can expect that software-on-demand may become one important deployment model for common personal applications. Just like today's video-on-demand (VOD) services, maybe some SOD (software-on-demand) providers will spring up, and a user can play her personalized software on any compatible hosts without installation and without conflicts. Some applications can even be subscribed to and pushed to one's host, so the download latency is hidden.

Others

In the current implementation, P2P transportation is used as an architectural requirement. In fact, it is apparent that other data delivery mechanisms can be employed. For example, to deploy such a system in an enterprise, one or more central servers can be used as the application source.

Another adaptation is that, if the host has the same application and its components, our system should use these components instead of downloading them from the Internet. But in this situation, if the host application is infected with a virus/malware, other applications running inside VM are vulnerable to security issues (even if we assume that our access control mechanism is not infected). A solution to this problem can be addressed by having the component be verified based on its hash value before it is used. If the verification fails, the component is downloaded even though it is present locally.

Related Work

Portable Systems & Software

Chen and Noble [21] observed that virtual machine technology [22] can be used to migrate sessions between computers and thus be used for mobility. Internet Suspend/Resume [23] demonstrated that using commercial VM technology such as VMware Workstation, together with a networked file system such as Coda [24], makes it possible to walk up to a machine and resume a suspended session. Each ISR client has a Host OS and VMware Workstation preinstalled and has access to a networked store of VM images.

SoulPad is a new approach based on carrying an auto-configuring operating system along with a suspended virtual machine on a small portable device. With this approach, the computer boots from the device and resumes the virtual machine, thus giving the user access to his personal environment, including previously running computations. SoulPad has minimal infrastructure requirements and is therefore applicable to a wide range of conditions, particularly in developing countries.

A Soulpad-like solution is DoK (Desktop on Keychain) [25]. In contrast, it uses the local machine's installed operating system (Windows), and runs VirtualPC there, resuming the user's previous virtual machine session.

A commercialized product is Moka5 LivePC, which contains everything needed to run a virtual computer: an operating system and a set of applications.

LivePCs can be used on the desktop, or users can take them on a portable USB drive or access them through a network server. In the latter case, a central difference server can provide a VM image "diff" through streaming, so that the end user will not experience undue delays on system startup.

Unlike these approaches, our solution is based on ultralightweight virtualization. Accordingly, the storage capacity required by our solution is much smaller, and the performance overhead introduced by virtualization is almost negligible.

Some recent commercial offerings attempt to support personalization of anonymous PCs at OS-level. For example, Migo [26] allows users to carry personal settings and files on a USB flash key. One limitation of this approach is that it must be tailored for each application to be migrated. Moreover, it only saves personalized data into the USB storage, not the applications themselves.

U3 [27] presents a development specification of portable applications, which means software should be rewritten for portability. So compared with our solution, it is not a transparent one.

A similar solution is Ceedo [28], but there is no documentation of its implementation technologies.

Software As Service (SaaS)

SaaS is a software usage mode where the vendor develops a web-native software application and hosts and operates the application for use by its customers over the Internet. In the past, this mode was usually used by some enterprise-level software, including CRM, SCM, ERP and so on. Now, it has moved closer to ordinary users.

One example of this approach are web-based applications such as Google Docs & Spreadsheets [29], in which a web browser is usually employed as a running platform for word processing and spreadsheet applications. This looks like a promising software delivery model; however, the applications have to be rewritten for the Internet environment. So for the existing desktop applications, a compatible model is preferred.

Virtualization has also been deployed in on-demand software. One solution is Progressive Deployment System (PDS), which is a virtual execution environment and infrastructure designed specifically for deploying software on demand while enabling management from a central location. 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 [30], which can help IT departments to cut costs while increasing operational agility and reducing conflicts. 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.

However, both PDS and SoftGrid are designed for LAN environments and are not general solutions.

Our solution is an interesting and helpful exploration for deploying personal software on demand in a compatible way. Compared with other works, it combines application virtualization and SaaS technologies together: users can access their personalized applications over the Internet with some access control, which is speeded up by P2P transportation and the local cache.

Others

Using P2P to improve system's start-up performance has been used in some other systems, like Moobi [31]. Moobi uses BitTorrent to provide efficient distribution of the image cache of a dataless-workstation system, and it can support many more nodes to startup simultaneously compared to conventional network booting.

Another work we have referred to is OS Circular [32]. It is a framework for Internet Disk Image Distribution of software for virtual machines, those which offer a virtualized common PC environment on any PC. Especially, it used FUSE [33], a user-space file system framework, to implement a stackable virtual disk across the Internet, which inspired us to design the anchor file system.

Summary and Future Work

This paper presents a solution for portable Windows applications/customizations based on OS-level virtualization and P2P technologies. This solution can separate an application's private files/folders/registry entries into a portable device and employs *API interception* to make the application transparently access these resources at run time. From the user's viewpoint, she is able to access her personalized applications and data conveniently on any compatible computer, even though they are not present on the local disks of the host computer.

In addition, to present a friendly interface for users, file system virtualization is implemented, combined with access control to prevent any illegal software usage. P2P and local caching are used to speed up the download performance.

The design principle is presented and a prototype solution is introduced. We find in reality some practical issues, such as pre-installed applications, shell folders and application-irrelevant accesses, should be dealt with individually. Owing to the lightweight virtualization, the extra performance overhead mainly comes from the network latency, which can be decreased by P2P transportation and the local cache.

We believe this solution can be used not only for personal computing across the Internet but also for software deployment and administration in enterprises. In this mode, software installation can be converted into content distribution, which is hosted centrally but runs on demand locally. Doing so will reduce the PC total cost of ownership (TCO) significantly.

Currently we put users' data and customizations on the portable device for privacy. In the future, we plan to discard the device entirely and employ the network to provide and store everything. The challenge lies in how to keep coherence of personal data across the Internet. Moreover, we intend to address the problem of upgrading portable applications transparently, especially upgrade those cached on portable devices, in the next version. Some existing Content Distribution Network (CDN) technologies will be referred to.

Another issue with the current version is that the filter driver-based implementation requires the user have Administrator' access, which violates the principle of least privilege [34]. We plan to achieve a user-level version in the next version.

Author Biographies

Zhang, Youhui is an Associate Professor in the Department of Computer Science at the University of Tsinghua, China. His research interests include portable computing, network storage and microprocessor architecture. He received his Ph.D. degree in Computer Science from the same university in 2002.

Wang, Xiaoling is a Master's degree student in the Research Institute of Information Technologies at the University of Tsinghua, China. Her current research interests include portable computing and computer system simulation.

Hong, Liang is a Master's degree student at the Beijing University of Post and Telecommunication, China. Currently he is doing his research work at the University of Tsinghua, and his research field is portable computing.

Acknowledgement

We would like to thank Prof. Brent B. Hoon Kang, and Ms. Aeleen Frisch and our anonymous reviewers for their time and insightful comments regarding this paper.

The work is supported by the High Technology Research and Development Program of China under Grant No. 2006AA01Z111 and the National Grand Fundamental Research 973 Program of China under Grant No. 2007CB310900.

Bibliography

[1] Richardson, T., Q. Stafford-Fraser, K. R. Wood, et al., "A Virtual Network Computing," *Internet Computing, IEEE*, Vol. 2, Num. 1, January, 1998.

[2] *CITRIX*, http://www.citrix.com .

[3] http://en.wikipedia.org/wiki/Web_application .

[4] Caceres, Ramon, Casey Carter, Chandra Narayanaswami and Mandayam Raghunath, "Reincarnating PCs with Portable SoulPads," *Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005)*, June, 2005.

[5] http://www.moka5.com/ .

[6] Chandra, R., N. Zeldovich, C. Sapuntzakis, and M. S. Lam, "The Collective: A Cache-Based System Management Architecture," *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI 2005)*, May, 2005.

[7] http://www.cl.cam.ac.uk/research/srg/netos/xen/performance.html .

[8] Adams, Keith and Ole Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, March, 2006.

[9] http://tejasconsulting.com/open-testware/feature/installwatch.html .

[10] Hunt, Galen and Doug Brubacher, "Detours: Binary Interception of Win32 Functions," *Proceedings of the Third USENIX Windows NT Symposium*, July, 1999.

[11] *File System Filter Drivers*, http://www.microsoft.com/whdc/driver/filterdrv/default.mspx .

[12] Alpern, Bowen, 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.

[13] Yu, Yang, Fanglu Guo, Susanta Nanda, Lap-chung Lam and Tzi-cker Chiueh, "A Feather-weight Virtual Machine for Windows Applications," *Proceedings of the Second ACM/USENIX Conference on Virtual Execution Environments (VEE'06)*, June, 2006.

[14] http://www.winehq.org/site/docs/wineusr-guide/index .

[15] http://sourceforge.net/projects/libtorrent/ .

[16] Cohen, Bram, "Incentives Build Robustness in BitTorrent," *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, June, 2003.

[17] Morrey III, Charles B. and Dirk Grunwald, "Content-Based Block Caching," *Proceedings of 23rd IEEE Conference on Mass Storage Systems and Technologies*, May, 2006.

[18] *Windows Management Instrumentation*, http://msdn.microsoft.com/en-us/library/aa394582.aspx .

[19] http://en.wikipedia.org/wiki/Features_new_to_Windows_XP .

[20] Russinovich, Mark E. and David A. Solomon, *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000*, Microsoft Press, January, 2005.

[21] Chen, P. M. and B. D. Noble, "When Virtual is Better Than Real," *Proceedings of IEEE 8th Workshop on Hot Topics in Operating Systems*, May, 2001.

[22] Goldberg, R. P., "Survey of Virtual Machine Research," *IEEE Computer*, Vol. 7, Num. 6, June, 1974.

[23] Kozuch, M. and M. Satyanarayanan, "Internet Suspend/Resume," *Proceedings of 4th IEEE Workshop on Mobile Computing Systems and Applications*, June, 2002.

[24] Satyanarayanan, M., "The Evolution of Coda," *ACM TOCS*, Vol. 20, Num. 2, May, 2002.

[25] Annamalai, Muthukarrupan, Andrew Birrell, Dennis Fetterly and Ted Wobber, "Implementing Portable Desktops: A New Option and Comparisons," *Microsoft Research (MSR)-2006-151*, October, 2006.

[26] http://www.migosoftware.com/default.php .

[27] http://www.u3.com .

[28] http://www.ceedo.com/ .

[29] http://documents.google.com/ .

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

[31] McEniry, Chris, "Moobi: A Thin Server Management System Using BitTorrent," *Proceedings of 21st Large Installation System Administration Conference*, November, 2007.

[32] Suzaki, Kuniyasu, Toshiki Yagi, Kengo Iijima, and Nguyen Anh Quynh, "OS Circular: Internet Client for Reference," *Proceedings of 21st Large Installation System Administration Conference*, November, 2007.

[33] *Filesystem in Userspace*, http://fuse.sourceforge.net/ .

[34] http://en.wikipedia.org/wiki/Principle_of_least_privilege .