Future Generation Computer Systems 29 (2013) 330–340

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

CyberLiveApp: A secure sharing and migration approach for live virtual desktop applications in a cloud environment

Jianxin Li [a,*], Yu Jia [a], Lu Liu [b], Tianyu Wo [a]

[a] School of Computer Science & Engineering, Beihang University, Beijing, China
[b] School of Computing and Mathematics, University of Derby, Derby, UK

ARTICLE INFO

Article history:
Received 24 November 2010
Received in revised form
4 May 2011
Accepted 5 August 2011
Available online 12 August 2011

Keywords:
Cloud computing
Software as a service (SaaS)
Virtual machine
Secure accessing
Live application sharing and migration

ABSTRACT

In recent years, we have witnessed the rapid advent of cloud computing, in which remote software is delivered as a service and accessed by users using a thin client over the Internet. In particular, a traditional desktop application can execute in the remote virtual machines of clouds without re-architecture and provide a personal desktop experience to users through remote display technologies. However, existing cloud desktop applications have isolated environments with virtual machines (VMs), which cannot adequately support application-oriented collaborations between multiple users and VMs. In this paper, we propose a flexible collaboration approach, named CyberLiveApp, to enable live virtual desktop application sharing, based on a cloud and virtualization infrastructure. CyberLiveApp supports secure application sharing and on-demand migration among multiple users or equipment. To support VM desktop sharing among multiple users, we develop a secure access mechanism to distinguish their view privileges, in which window operation events are tracked to compute hidden areas of windows in real time. A proxy-based window filtering mechanism is also proposed to deliver desktops to different users. To achieve the goals of live application sharing and migration between VMs, a presentation redirection approach based on VNC protocol and a VM cloning service based on the Libvirt interface are used. These approaches have been preliminary evaluated on an extended MetaVNC. Results of evaluations have verified that these approaches are effective and useful.

1. Introduction

Nowadays, the Internet has become a data and computing center, with a large number of applications, and ubiquitous equipment. New Internet-based computing paradigms, e.g., cloud computing [1,2], have emerged, aiming to bring large-scale computing, storage, and data service resources together to build a virtual computing environment. These paradigms provide simple and transparent approaches that enable effective sharing and utilization applications over the Internet.

In the early personal computing era, to use software, users needed to install it under a granted license. This traditional method suffers several limitations as the software has increased both in amount and number of categories. First, software users need to deal with many complex tasks in terms of software installation, configuration, updating, and even troubleshooting. Besides, software which is dependent on their respective host operating systems may face compatibility issues. Normal users are thus loaded with an extra burden. In contrast, the concept of software as service (SaaS) [3] that emerged with cloud computing has been promoted by many companies, such as Amazon and Google. With SaaS, things become simpler. Software can be installed into VMs with easier encapsulation and secure isolation. Users could access software on demand through the Internet without worrying about maintenance issues. There are two approaches to achieve these goals. One is to redevelop software (e.g., GoogleDoc) based on Web technologies. This not only requires much extra work, but also leads to compatibility problems on various browsers. The other approach is based on desktop virtualization, which separates the presentation and execution of applications, and provides a transparent way to deliver an application-based remote virtual desktop. Currently, a virtual desktop can be delivered based on remote display protocols, such as VNC (Virtual Network Computing) [4,5], and RDP (Remote Desktop Protocol) [6]. These protocols generally provide methods for remote virtual desktop accessing, so that users can log into a VM and operate on the desktop.

Over the Internet, secure collaboration among users and equipment is an important requirement [7], and, in particular, user terminal equipment shows increasing mobility. In a cloud computing environment, applications are executed in VMs. Therefore, to provide a novel flexible collaboration service among

* Corresponding author.
E-mail addresses: lijx@act.buaa.edu.cn, lijianxin@gmail.com (J. Li), jiayu@act.buaa.edu.cn (Y. Jia), l.liu@derby.ac.uk (L. Liu), woty@act.buaa.edu.cn (T. Wo).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.08.001

live virtual desktop applications (a *live application* in short) is the focus of our work. There are some new scenarios for live application sharing in a single VM or among multiple VMs. In a single VM, the collaboration scenarios can be supported based on shared desktops. For example, in a remote teaching system, desktop sharing enables the instructor and students to operate on the same view. However, a traditional way for remote desktop sharing is to simply share the desktop login account and password, which may induce inconvenience and insecurity. In a cloud, a virtual machine monitor (e.g., KVM) only provides a coarse-grained access control in VM-level granularity (e.g., Qemu VNC implementation). This brings a disadvantage that authorization only has two possible results: success or failure, and it means that users may either all have complete operation rights with poor security and privacy protection, or cannot concurrently access the desktop at all. In particular, it is impractical to share the whole desktop including users' private windows. Thus, a fine-grained access control mechanism at window-level granularity is required to provide security and privacy protection.

Example 1. Three applications (document editing, image viewing, and chatting) are running on a VM owned by user *A*. *A* wants to share the desktop with all these applications with user *B*, but wants to block input from *B*'s keyboard and mouse. *A* also gives the authority of viewing the pictures on his virtual desktop to user *C*, but does not want to show *C* the document editing window and the chatting window.

In the traditional approach for application sharing among multiple VMs, when a user needs to share an application and related data with other users, binary copies of the application are generated. Users may have to endure a long-time installation and configuration process, which may terminate due to system incompatibility failures. In a cloud computing environment, this can be easily handled through presentation redirection technologies among application desktops or an application data clone in the cloud.

Example 2. In a virtual machine environment, user *A* is operating a desktop with a drawing application window on it. After finishing his/her part of the drawing, *A* intends to let user *B* continue with it, which means that *A* needs to migrate the application window to *B*'s desktop. While user *C* is interested with this drawing application and the part drawing that user *A* has finished, *C* hopes to have a complete copy of this application and continue the drawing, which means that user *A* has to give a clone of the application and user data to user *C*.

To meet the above requirements, we have designed a live virtual desktop application sharing and migration system, named CyberLiveApp. The major contributions are as follows.

(1) *Multi-user secure application accessing in a single VM.* To enable secure live application sharing and collaboration among different users, we design a multi-user secure accessing mechanism at application-window-level granularity. A VM owner can configure desktop sharing policies and use a proxy to filter useless desktop windows.

- *Multi-user secure desktop accessing approach.* Considering the VM security sharing and collaboration needs, we design and implement a multi-user secure accessing approach based on MetaVNC, extend the RFB protocol to support multi-user authentication, and provide the functionality of user customized view.
- *Window operation event tracking and real-time hidden area computation approach.* As the access control objects, windows generally overlap each other and may change dynamically upon events such as resizing, minimizing, maximizing, moving, and so on. We first track the events which affect the layout of multiple windows, and design a real-time hidden area computation approach to extract a permitted desktop area on a virtual desktop.
- *Proxy-based window filtering mechanism.* In the existing remote control software, access control mechanisms are generally in a desktop level, which is a coarse-grained mode, and cannot specify the detailed windows. We provide a desktop view proxy for desktop delivery among different users. This proxy calculates permitted desktop areas and clips hidden areas based on the VM owner's specific security policy.

(2) *Application sharing and migration among multiple VMs.* To enable collaborations among multiple VMs, we design a live application sharing and migration mechanism. Through presentation streaming redirection and VM cloning technology, an application can be easily shared or migrated. We also design an application state maintaining mechanism during application sharing and migration.

- *Presentation streaming redirection approach.* Via live application presentation streaming, users can subscribe to software services and access the virtual desktop. In a cloud environment, all VMs are located in the same network. Due to the encapsulation and migration abilities of the VM, and the favorable communication bandwidth among the hosts, we achieve the sharing and migration of live applications though redirection of application presentation streaming among different clients. During the migration and sharing process, a VNC connection on a certain client side can be easily created or closed.
- *Application state maintaining mechanism.* Ensuring the consistency of all application states during application sharing and migration is a critical problem. It means all users involved should get the same view on a shared application, including the application configuration and all user data (stored in both disks memories). During application sharing, a target user should get a complete copy of the application. We achieve this by cloning the VM in which the target application is running, and design a state maintaining mechanism to coordinate the communication sequences among users.
- *Virtual desktop application sharing and migration protocols.* Considering the inflexible application copy and desktop sharing under the traditional desktop mode, we design two protocols for virtual desktop application sharing and migration. In the protocols, we design a desktop session management approach based on inter-process communication. With the protocols, clients and servers can communicate via SOAP messages, and application sharing is achieved based on VM cloning.

(3) All approaches mentioned above have been implemented in CyberLiveApp based on extended MetaVNC and the virtual machine monitor KVM. Furthermore, we have performed simulations to verify that the approaches are effective and useful.

The rest of this paper is organized as follows. We discuss the requirements for live applications in Section 2. Section 3 presents the design of CyberLiveApp, the architecture, and protocols. We elaborate implementation experiences in Section 4. The related work is discussed in Section 5. Finally, we conclude the paper in Section 6.

2. Requirement analysis

In a VM, when a user wants to share applications on his/her virtual desktop with other users, the basic requirements for CyberLiveApp are summarized as follows.

Requirement 1: Providing a multi-user accessing mechanism, and controlling viewable desktop areas at application window granularity.

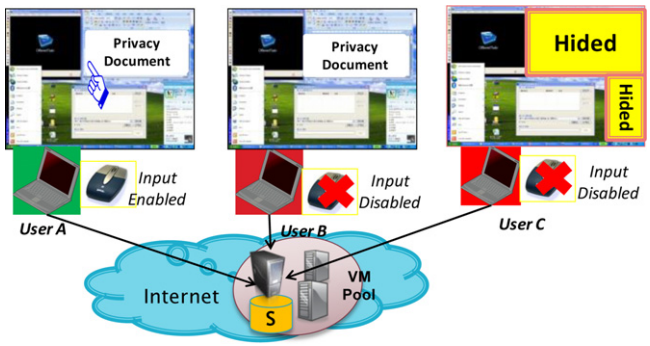


Fig. 1. A scenario of secure remote access to a single VM from multiple users.

Table 1
Permission matrix for multiple users.

	ACD see picture browser	Office word	MSN	Mouse & keyboard input
Owner *A*	Y	Y	Y	Y
User *B*	Y	Y	Y	N
User *C*	Y	N	N	N

Requirement 2: Providing the capability to enable/disable input devices such as the keyboard and mouse. For instance, some users are allowed to browse some windows but not to operate or edit them.

Requirement 3: Preventing a third party from eavesdropping on the content of desktop during transmission, i.e., to ensure confidentiality of transmitted context though an encrypted tunnel.

As shown in Fig. 1, there are three users connecting to the same VM to view its virtual desktop, but their permissions are different. Table 1 is an example of a permission matrix. User *A* shares all the windows with user *B*, but does not permit *B* to operate the windows via his/her keyboard or mouse. User *A* also shares the desktop with user *C* after hiding the window areas of the text editing and chatting programs. User *C* is also not allowed to operate *A*'s desktop.

CyberLiveApp achieves the display of remote desktop based on MetaVNC. Some key technologies are as follows.

- *Secure access control for multiple users*. The virtual desktop can be accessed by multiple users. Every user has a customized view, and can connect to the virtual desktop in a secure way.
- *Configuration of sharing policy among users*. The virtual desktop owner can specify policies for its applications, and configure the permitted users and their permissions as shown in Table 1.
- *The extraction of application windows*. For the convenience of administrators to assign browse permissions for different users, CyberLiveApp extracts all the windows from the operating system (OS), and stores them in a list. It first gets the handlers of all the windows on the desktop, then obtains the processes they belong to, and finally extracts the names of the processes.
- *Hiding specific application windows*. CyberLiveApp selects private windows and calculates the hidden area related to these windows in accordance with the sharing policies.
- *The blocking of keyboard and mouse events*. Through intercepting keyboard and mouse events of different users, the server side of CyberLiveApp decides whether or not to block the keyboard and mouse input events.

Among multiple VMs, when a user wants to copy or migrate applications on a virtual desktop, the source and the destination of application sharing or migration should be specified. As shown in Fig. 2, in the VM pool, user *A* has two VMs running *App*1 and *App*2, respectively. User *A* can connect to the virtual desktop to view *App*1 and *App*2 through a notebook computer (Client1). When user *A* leaves, he/she can migrate *App*1 to his/her mobile terminal (Client2). User *B* is very interested in *App*2, and user *A* can easily send this application to user *B*'s client (Client3).

The key technologies of live application sharing and migration among multiple VMs include the following.

- *Redirection of application presentation streaming among multiple client terminals*. In a cloud environment, all applications are maintained in a VM pool, where the virtual desktops are delivered to client terminals through VNC or RDP protocols. In the case of application migration or sharing, application windows can be redirected to a new client terminal.
- *Management of VNC connections on the client side*. CyberLiveApp achieves the goal of remote desktop access based on MetaVNC. During the phase of application sharing and migration, an interprocess communication technology is applied on each client to automatically create/close a specified VNC connection, thus achieving the redirection of application presentation streaming.
- *Application states maintaining*. CyberLiveApp regulates the communication sequence and maintain the consistency of the applications. Two protocols have been designed for virtual application sharing and migration, which will be introduced in Section 3.2.1.

3. Design of CyberLiveApp

3.1. Secure sharing of a single VM desktop among multiple users

Fig. 3 shows the architecture of secure desktop sharing for multiple users. For every VM owned by a user, the display

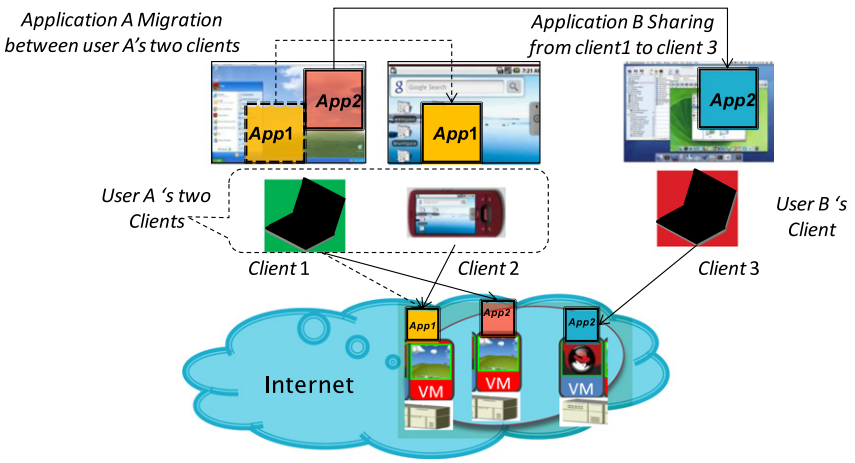


Fig. 2. A scenario of live application sharing and migration between deferent VMs.

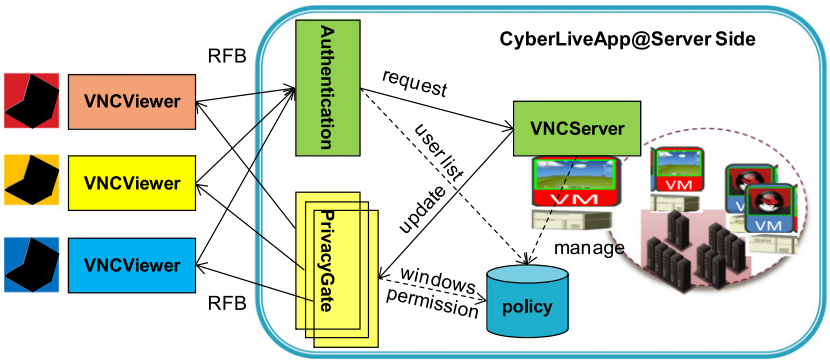


Fig. 3. Architecture of secure desktop sharing for multiple users.

of running applications is delivered to the `vncViewer` on the client sides through a `vncServer` installed in the VM. The basic procedure of secure desktop sharing among multiple users is as follows.

1. *Extraction of application windows in the operating system*. To enable administrators to assign application permissions to users at window granularity, CyberLiveApp needs to extract all related windows from the operating system. First, CyberLiveApp gets all of the window handlers on a desktop. Second, CyberLiveApp uses the window handlers to obtain related process handlers which the windows belong to, by which we can get the application process names. Finally, CyberLiveApp creates an available application list to users for making security policies.
2. *Policy configuration on CyberLiveApp@Server side*. The owner of the virtual desktop creates a list of those who are allowed to access his/her desktop and assigns permissions. The policy is de facto an access control list. The policies include two types — one is a *permit policy* which defines the windows that a user can access, and the other is a *forbid policy* which defines the windows that a user cannot access. When a `vncServer` runs in the OS, you can right-click the MetaVNC icon to pop-up a menu. When one chooses `Properties menu`, some configuration options are shown on the tabs, and we add two extra tabs on the MetaVNC options dialog to support such configuration.
3. *Authentication of requested users.* When a user wants to access the remote virtual desktop, He/She will first launch the `vncViewer` to send a connection request to the `vncServer` though the RFB protocol. The *authentication* module will authenticate the identity of the requesting user based on the local *policy* database. If the user is authenticated and acquires the permissions, the `vncServer` will build a connection with the `vncViewer`.
4. *The PrivacyGate module functions*. In current MetaVNC functions, a `vncClient` object is created when the `vncServer` connection is created. Once some events occur on the server desktop, the `vncServer` will deliver the updated desktop to every `vncClient`. Therefore, each client will get the exact same desktop view. In order to allow each user to have a customized view, the `vncServer` works on the proxy mode to hide some specified windows according to the policies. The significance of our design is a breakthrough in the current MetaVNC structure. A PrivacyGate is used to send the desktop to each `vncClient`, which means that, when the server desktop is changed, it will update the desktop view though a `PrivacyGate` module based on the *permit policy* or *deny policy*. The `PrivacyGate` module will hide the window area for which the corresponding user has no browsing permission, and then deliver the filtered desktop view to every `vncViewer`.
5. *Hiding specific application windows of PrivacyGate.* In order to allow every user to have an independent view, the `vncServer` needs to hide the specified windows according to the policy. First, CyberLiveApp gets the names of the invisible windows which the administrator has configured for the user. Through this name, CyberLiveApp calls a query function to obtain the handlers of the application window and then gets rectangle area of this window. Second, this rectangle area is added into the hidden area, which could be a simple rectangle or a complex polygon. When setting the hidden area, some overlapped areas with the hidden windows belonging to the permitted top-level windows should be shown.
6. *Blocking input from keyboard and mouse at PrivacyGate.* Since some users are permitted to only browse, not operate, a desktop, CyberLiveApp blocks input of keyboard and mouse from these users. If the *input variable* state is "*DISABLE*", the server will block the client's inputs, and the user can only view the desktop windows, but cannot operate the server's windows. If the *input variable* is "*ENABLE*", the user's inputs from the remote client will be handled and responded normally. We control the input operations through the Windows `hook` function, and it intercepts and processes the requests to decide whether to forward the input events.

3.2. Application sharing and migration among multiple VMs

To realize remote application access in a cloud, we use the VNC protocol to transfer the virtual desktop of a remote VM. The VNC protocol works at the buffer frame layer and supports the remote access to graphical user interfaces, and the mouse or keyboard inputs can be transferred to the remote application, thus achieving a transparent access to the applications. In such a presentation streaming-based software delivery mode, when a client wants to migrate or share an application to another client, the presentation streaming of this application should be redirected, and the corresponding VM will be cloned in the case of application sharing. Based on these considerations, we have designed the architecture for live application sharing and migration (shown in Fig. 4). The key components are as follows.

- *Client Controller:* With a modified `vncViewer` to display application windows from multiple VM machines, the Client Controller manages multiple VNC connections between the `vncViewer` and `vncServers`. Using an inter-process communication technology, the Client Controller can close or create a specified VNC connection.
- *Server Controller:* This maintains information of all live users and applications in the VM pool. This component provides a unified management and processing of all clients' requests, which includes application presentation streaming, and application sharing and migration service.
- *VM Manager:* This provides functionalities of monitoring, stopping, cloning, or restarting a specified VM with running applications. The VM Manager receives notifications from the Server Controller to clone a VM or manage the VM pool.

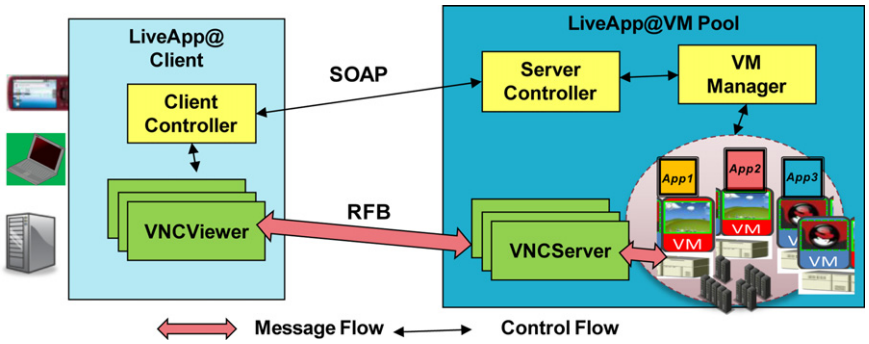


Fig. 4. Architecture of application sharing and migration among multiple VMs.

Table 2
Definitions of all the state variables.

	App state	Meaning
	$v1$	NORMAL
	$v2$	SHARE
	$v3$	DISCONNECT
	$v4$	MIGRATE
	$v5$	CLONE

Table 3
Application sate transition during migration.

No	$s = \langle v_1, v_2 \rangle$	Description
1	$\langle v1, v3 \rangle$	Client1 has no request
2	$\langle v4, v3 \rangle$	Client1 sends a migration request to Client2
3	$\langle v4, v4 \rangle$	Client2 accepts the migration request
5	$\langle v3, v1 \rangle$	Client1 closes the application connection, and Client2 connects to it

3.2.1. Application state

In CyberLiveApp, the Server Controller is permitted to identify all the clients' requests and capable of responding to them by sending control messages to every Client Controller, while the VM Manager manages multiple VM instances in a VM pool. During the phases of application migration and sharing, application states or messages may change frequently. Therefore, we use a state tuple to ensure the consistency of various application and VM information.

The Server Controller maintains a table of states of all virtual applications. When a client sends a request to the application, the Server Controller updates its state. We use the state symbol v to denote a requested state to the application. All the state variables are defined in Table 2.

The v field value of an application is one of the five types at one time. When an application's op = 'NORMAL', that means there is no shared or migration process for this live application. During the procedure of application sharing and migration, we use a tuple s to describe application states between two clients:

$$s = \langle v_1, v_2 \rangle,$$

where v_1 denotes the application state that a client requests, and v_2 is the application state that another client receives. During the procedure of application migration and sharing, an application's state will be transferred according to order as shown in Tables 3 and 4.

We have implemented a SOAPServer in the Server Controller, which receives SOAP messages from the Client Controller and updates the state of application.

3.2.2. Virtual desktop application sharing and migration protocols

When a Client Controller requests to migrate or share a live application, two corresponding protocols are designed to achieve these goals. Several steps are taken to complete the migration or sharing, as shown in Figs. 5 and 6.

Table 4
Application sate transition during sharing.

No	$s = \langle v_1, v_2 \rangle$	Description
1	$\langle v1, v3 \rangle$	Client1 has no request
2	$\langle v2, v3 \rangle$	Client1 sends a sharing request to Client2
3	$\langle v2, v2 \rangle$	Client2 accepts the sharing request
4	$\langle v2, v5 \rangle$	VM Manager clones this application, and another instance will be started
5	$\langle v1, v3 \rangle$	Client1 returns its initial state as order 1

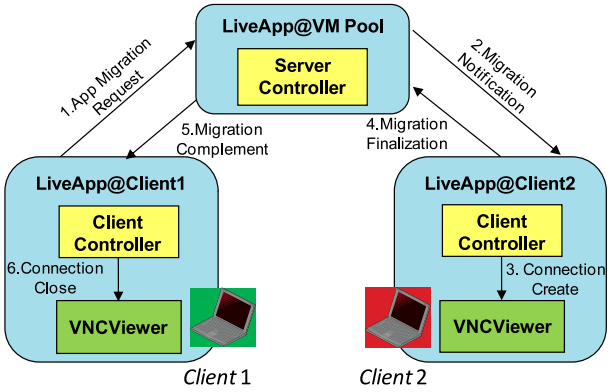


Fig. 5. Application migration protocol.

The steps of application migration are as follows.

Step 1: Client1 → Sever Controller: {*AppName*, *MigrationSource*, *MigrationDes*}. Client1 sends a migration request to the Server Controller with the *AppName*, *MigrationSource*, and *MigrationDes*. In CyberLiveApp, all connected clients first register in the Server Controller, so Client1 can choose the destination from the client list on the LiveApp@Client side.

Step 2: Sever Controller → Client2: {*AppName*, *VMInfo*, *MigrationSource*, *MigrationDes*}. Client2 gets a migration notification from the LiveApp@VM Pool side, and the VM IP address and VNCServer port are enclosed in VMInfo, so the VNCViewer can connect to a remote VNCServer to view the application. Sometimes, Client2 does not have a public IP address (e.g., it is located in a local network), so the Server Controller cannot initialize a connection to Client2. In CyberLiveApp, we can change the mode of Step 2 through actively querying the Server Controller at a certain interval.

Step 3: Client Controller@Client2 → VNCViewer@Client2 : {*VMInfo*}. Client2's Client Controller creates a new Client2's VNCViewer connection to the remote VNCServer running on the VM, and the VNCServer will also close the connection with Client1. Therefore, the presentation streaming of Client1 can be redirected to Client2,

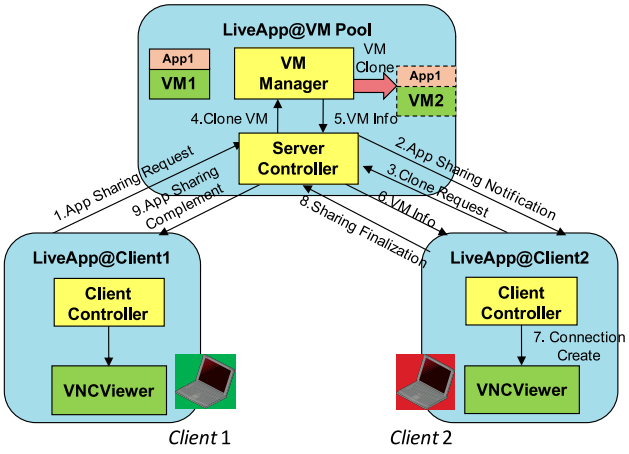


Fig. 6. Application sharing protocol among multiple VMs.

and the application view and data are exactly the same due to there being no change in its running environment.

Step 4: Client2 → Server Controller: {*MigrationFinal*}. Client2 sends a *MigrationFinal* message to the Server Controller.

Step 5: Server Controller → Client1: {*MigrationComplement*}. The Server Controller updates the meta information of VMPool, and sends a *MigrationComplement* message to Client1.

Step 6: Client Controller@Client1 → VNCViewer@Client1: {*VMInfo*}. Client1's Client Controller clears the connection record of Client1's VNCViewer, and shows a migration success dialog on Client1.

The steps of application sharing are as follows.

Step 1: Client1 → Server Controller: {*AppName*, *SharingSource*, *SharingDes*}. Client1 sends an application sharing request to Server Controller with the AppName, SharingSource, and SharingDes. In CyberLiveApp, all connected clients first register in the Server Controller, so Client1 can choose the destination from the client list on the LiveApp@Client side.

Step 2: Server Controller → Client2: {*AppName*, *SharingSource*, *SharingDes*}. Client2 gets a sharing notification from the LiveApp@VM Pool side. Sometimes, Client2 does not have a public IP address (e.g., it is located in a local network), so the Server Controller cannot initialize a connection to Client2. In CyberLiveApp, we can change the mode of Step 2 through actively querying the Server Controller at an interval.

Step 3: Client2 → Server Controller: {*CloneRequest*}. Client2 sends a clone request to the Server Controller after it accepts the sharing message from Client1.

Step 4 & *Step* 5: Server Controller → VMManager: {*VMClone*, *VMInfo*}. The Server Controller sends a VM clone request to the VMManager, and the VMManager clones a VM with the `Libvirt` command.

Step 6: Server Controller → Client2: {*VMInfo*}. The cloned VM IP address and VNCServer port are enclosed in *VMInfo*, so the VNCViewer can connect to a remote VNCServer to view the application.

Step 7: Client Controller@Client2 → VNCViewer@Client2 : {*VMInfo*}. Client2's Client Controller creates a new Client2's VNCViewer connection to the remote VNCServer running on the cloned VM. Therefore, Client2 can view the duplicated application like Client1, and the initial application view and data are exactly the same.

Step 8: Client2 → Server Controller: {*SharingFinal*}. Client2 sends a *SharingFinal* message to the Server Controller.

Step 9: Server Controller → Client1: {*SharingComplement*}. The Server Controller updates some meta information of VMPool, and sends a *SharingComplement* message to Client1, and Client1 will unlock its interaction.

In terms of application sharing, we intend to provide a completely duplicated application for the two clients. In CyberLiveApp, this requirement is met by cloning the VM in which the application runs. We will discuss the design and implementation of VM cloning in a later section. When a Client Controller sends a VM cloning request to the Server Controller, the latter will send the VM id to the VM Manager, who will return the new VM IP address after the cloning completes. In this new cloned VM, all the disk and memory data are the same as the original one, thus providing a duplicated application view.

4. System examples and simulations

Currently, we have implemented a CyberLiveApp prototype system, and verified its viability through some evaluations. The core remote display protocol is based on the RFB implementation of MetaVNC 0.6.6.

4.1. Implementation experience

4.1.1. Secure sharing of a single VM desktop among multiple users

The MetaVNC allocates an unoccupied slot as the `ClientID` for the arrived client request according to the array `vncClient *m_clientmap [MAX_CLIENTS]`. Then, a `vncClient` object will be initialized for this connection, and its attributes are set according to the passed parameters. After that, MetaVNC calls the `vncClient::Init` method to pass the `clientID` instance pointer. Finally, this user will be added into the unauthorized user list of `vncServer` through `m_clientmap[clientid]=client`.

(1) *Multi-user authentication*

In the authentication entry function `vncServer:: Authenticated (vncClientId clientid)` of MetaVNC, it first gets the `vncClient` object from the unauthorized list, and removes it from the `unauth list`. If the user is the first `vncServer` client, a `vncDesktop` object is created by calling `m_desktop->Init(this)`. Next, it allocates a `vncBuffer *buffer = new vncBuffer (m_desktop)`, and sets this buffer by calling `vncClient::SetBuffer` to add this user into `auth list`.

The multi-user authentication feature is implemented in the `vncClient. cpp` file of the MetaVNC project. There are two classes named `vncClient` and `vncClientThread` in `vncClient.cpp`. `vncClient` is used to send an updated image to the client while `vncClientThread` is the server's service thread which is used to receive messages from the client. The authentication function is implemented in `vncClientThread` because there are lots of interactions with the client. CyberLiveApp implements the authentication function in `BOOL vncClientThread::InitAuthenticate()`. We added a new security type named `rfbSecTypeMultiUser`, and it will be used if the client supports this security type; otherwise, the default authentication method will be used. In the new authentication process, the client returns the challenges and `username` to the server. When the server receives this `username`, it checks whether this user exists. If so, the server compares the received `password hash` with the hash value stored locally. If the two `usernames` and `passwords` match, the authentication is successful, then the client and the server will further negotiate to prepare for sending the remote desktop.

(2) *Proxy-based window filtering*

In the protocol of MetaVNC, the client thread in `vncServer` is used to receive requests from the clients and send update information to them, and this is one of the most predominant functionalities of `vncServer`. The `vncClient` class is responsible for sending data, which is implemented in the function `vncClient::SendUpdate`. The basic procedure is like this:

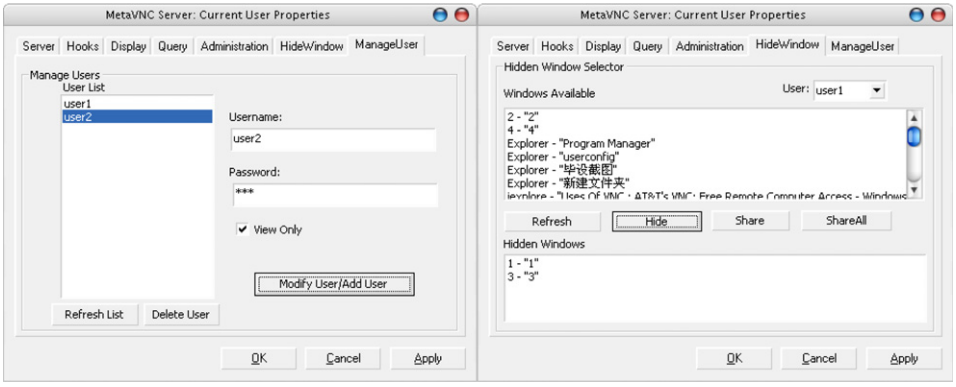


Fig. 7. The snapshot of MangerUser and HideWindows Tab.

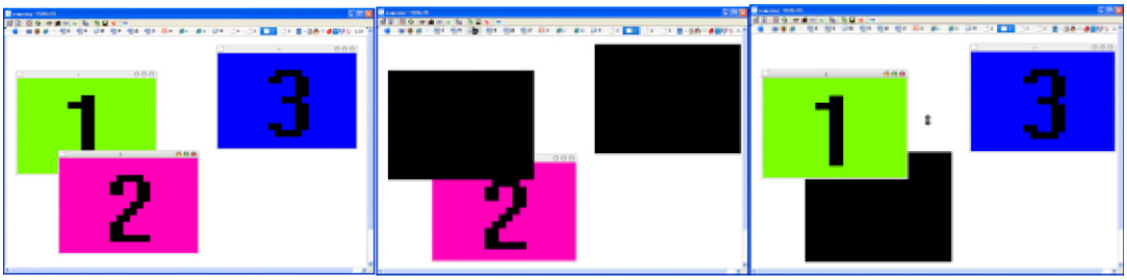


Fig. 8. The snapshot of desktop access by three clients.

first, `vncClient` calls `SendRFBMsg`, then it calls the function `SendCursorShapeUpdate` to send the mouse shape update, and calls function `SendCursorPosUpdate` to send the mouse position update, and then it calls function `SendTpDecRgn` to reset the region. Finally, vncClient calls `SendRectangles` to send the related data for updated rectangle.

The existing service threads can only request a display update or a send display update, which cannot meet the requirements of multiple user accesses. The tasks undertaken by CyberLiveApp include the followings.

- First, the access control module reads some configuration information, such as username, password, and windows list. The information will be sent to other modules, and we define the functions of `BOOL WritePrivateProfileString` and `DWORD GetPrivateProfileString` to modify the policy file.
- We can only get the window's name from the authentication module, but we ultimately need to get the permitted window area. Therefore, we implement a transformation function in `SendUpdate` of the `vncClient` object. First, we get the application path name of the current windows on the desktop, and further get the windows' rectangle area based on the application windows.

Two dialogs shown in Fig. 7 are extended on metaVNC to set user permissions and window permissions.

(3) *Windows region hide*

We need to deal with layer and overlap relations of multiple windows, so as to calculate the hidden areas. We implement this in the function `SendUpdate` of the `vncClient` object. First, we use `GetForegroundWindow` function to get the handler of the top window, and we set a flag variable. If the top window is a hidden window, this will be easy. If the top window is not a hidden window, we should get the window's size and remove this rectangle area from the hidden area with `tpRegion.SubtractRect`, so that the user can view the top window which he/she is using.

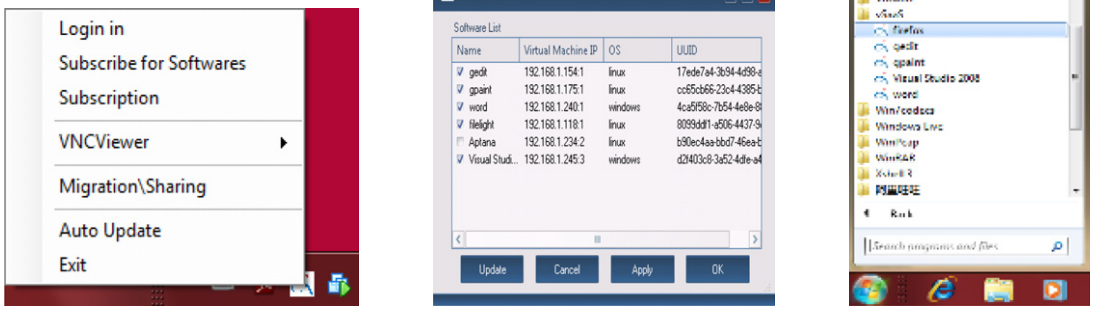
There is a `vncRegion` object named `toBeSent` in the function of `SendUpdate()` in `vncClient.cpp`. We call the `addRect` function in `vncRegion.cpp` to add the updated area into `toBeSent`. Then, we call the `Rectangles()` function to get the `rectList` which includes the information of the rectangle area to be sent. Finally, we call the `SendRectangles(rectList)` function to send the update rectangle.

Fig. 8 shows an example in which three clients connect to the same virtual desktop and they get different desktop views due to various permissions.

4.1.2. Application sharing for multiple VMs

In CyberLiveApp, the Client Controller manages multiple `vncViewer` processes, and the inter-process communication is based on Windows `COPYDATA` messages. We have used some Windows predefined messages such as `VM_CLOSE`, `VM_USER`, `VM_COPY-DATA`, as well as messages `VM_NEWCONNECTION`, `VM_CON_END`, `VM_KILL_ALL`, `VM_SEND_SUBS_HWND` defined by ourselves. For instance, `VM_NEWCONNECTION` is a message to create a new VNC connection, while `VM_CON_END` is a message to close a VNC connection. Since the original `vncViewer` project does not provide a function for closing a specified connection, we add a function `KillConnection(TCHAR *host, int port)` in the file `VNCviewerApp32.cpp`, and it calls function `kill()` to close a specified connection, while on the side of Client Controller, we use `DllImport(''user32.dll'')` to send a `COPYDATA` message to the `vncViewer` process though `SendMessage(int hWnd, int Msg,int wParam,int lParam)`.

The VM Manager manages all VMs in the VM pool, and its main functions are to create, start, or clone a specified VM with the support of the `Libvirt` library. `Libvirt` library is a Linux API realizing the virtualization functions of Linux; it supports a variety of virtual machine monitors including Xen and KVM [8] (we use KVM in CyberLiveApp). `Libvirt` provides a mechanism of saving the memory mirroring of a live VM and starting a VM based on the memory mirroring. When the VM Manager wants to clone a VM, it will first save the memory mirroring and disk mirroring of this VM, which is set to *Suspend* state during the cloning, and it will restore it after the cloning. The procedure of VM cloning is as follows.



(a) The menu of the Client Controller.

(b) The software list subscription.

(c) The subscripted software shortcut on the Start menu.

(d) The client and terminal list connected to the Server Controller.

(e) The application migration request.

Fig. 9. Snapshots of a Client Controller.

- Copying the original VM disk mirroring file. The user can also configure a copy-on-write (COW) to reduce the image copying time.
- Saving the VM memory mirroring.
- Restarting the original VM based on the memory mirroring.
- Modifying the network configuration and the disk mirroring path in the memory mirroring file.
- Restoring a new VM based on the modified memory mirroring and copied disk mirroring file.

Fig. 9 shows some snapshots of CyberLiveApp for application migration and sharing. When we start a Client Controller, it will display an icon on the task bar, and provide an easy way to subscribe to applications. All of these applications are maintained in the VM pool. After subscription, the Client Controller will automatically create shortcuts for these subscribed applications in the Windows Start menu. When a user wants to migrate or share a running application with another client, he/she first selects the target client from the online client list registered on the Server Controller. After migration, the application presentation streaming is closed in the sponsor client and redirected to the migration target client. However, after sharing, the presentation streaming will not be closed in the sponsor client, and the sharing target client will get an application streaming from the cloned VM. Therefore, they get the same application and user data at the same time within different clients, but then they can run independently.

4.2. Simulation results

In this section, we report on some simulations that were conducted to evaluate the CyberLiveApp prototype. We examine four performance aspects: (a) the overhead of multiple users' authentication in CyberLiveApp, (b) the network traffic of multiple users' access for a VM, (c) the cost of live application migration, and (d) the cost of a VM clone for application sharing. The simulation environment is shown in Fig. 10.

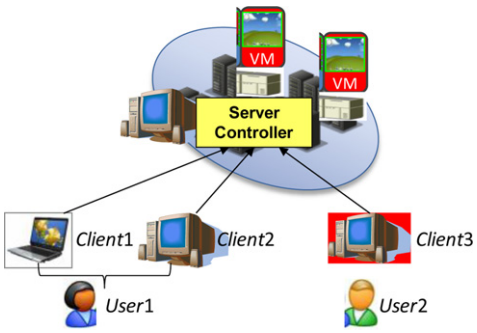


Fig. 10. Simulation environment.

Simulation environment setup: The Server Controller, Client2, and Client3 running on machines with Intel® Core™2 Duo CPU E6400@2.13 GHz, 4 GB RAM, Ubuntu 9.04 (Linux 2.6.28) operating system. Client1 is an IBM T61 computer with an Intel(R) Core(TM)2Duo CPU 2.2 GHz, 4 GB RAM, Windows 7 operating system. These machines have a 100 Mbps Internet connection. Client1 and Client2 are owned by User1. Unless specified separately, each experiment is executed five times and the average value is chosen.

4.2.1. Desktop sharing for a VM

Simulation 1: *In this experiment, we measure the total time for the VNCServer update for multi-user access control in CyberLiveApp. We simulate multiple clients on the three computers to connect to the VNCServer on the side of the Server Controller. The total VNCServer update time is recorded by increasing the number of concurrent clients from 1 to 5.*

The experimental results are shown in Fig. 11, which shows that the overall VNCServer update time almost increases linearly with the number of concurrent clients, and multi-user desktop

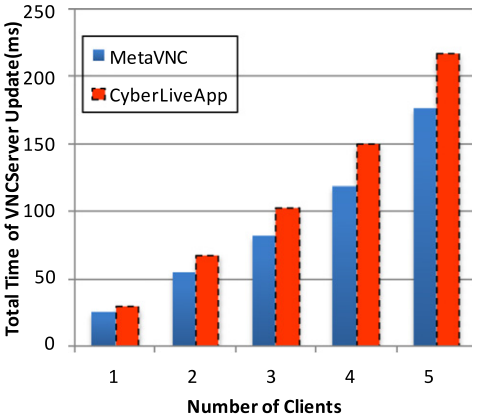


Fig. 11. The number of clients versus the total update time of the VNCServer.

Table 5
Network traffic comparison between MetaVNC and CyberLiveApp.

System	Operation	
	Moving windows (KB)	Maximizing/minimizing windows (KB)
MetaVNC	15 781	1151
CyberLiveApp	15 990	1173

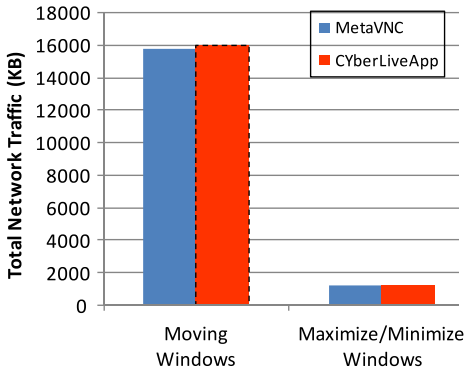


Fig. 12. Network traffic comparison between MetaVNC and CyberLiveApp.

access for a VM in CyberLiveApp is scalable. The bar chart also indicates that the VNCServer update time in CyberLiveApp is higher than in MetaVNC, but the overhead incurred by the multi-user authentication mechanism is less than 20%.

Simulation 2: *In this simulation, the network traffic impact due to Windows actions is measured. We run two types of window operation: one is moving the windows quickly, and the other is maximizing/minimizing the windows. We respectively record the total traffic in a period of time on MetaVNC and CyberLiveApp, where the total network traffic is recorded with Wireshark.*

The experimental results are shown in Table 5 and Fig. 12. We can see that these two systems have almost the same network traffic under the two different types of operation. Moreover, the frequent updating of a window will dramatically increase the network traffic. In all, the traffic overhead incurred by multi-user authentication in CyberLiveApp is small.

4.2.2. Application migration and cloning for multiple VMs

The two simulations below mainly focus on the time cost for application migration or sharing. We analyzed and compared the time consumed for application migration or sharing under different configurations.

Simulation 3: *In this experiment, we use two clients to simulate a scenario of live application migration. If a client has a public IP address,*

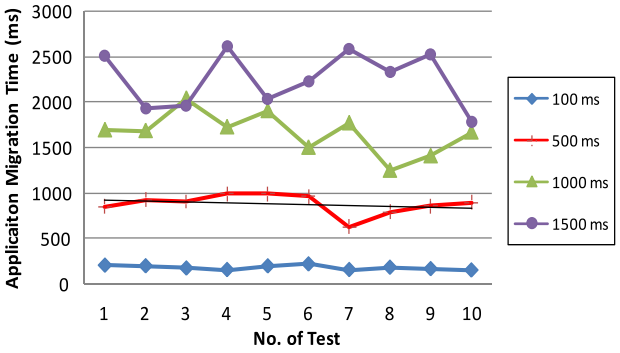


Fig. 13. Application migration time for different notification interval times.

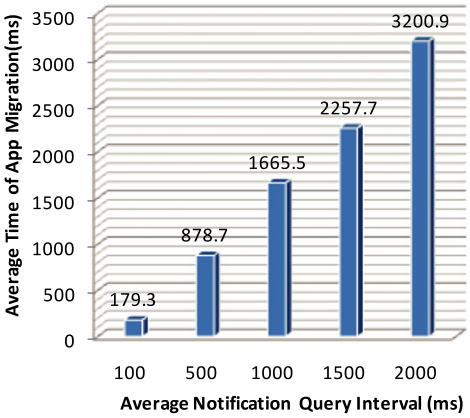


Fig. 14. Average application migration time for different notification interval times.

the total migration time is similar to that for Step 2 *in* Fig. 5. *However, in a cloud computing environment, the client generally does not have a public IP address, so the client should use a query mechanism to get the notification of Server Controller. We test the application migration time for this scenario under different query intervals.*

The experimental results are shown in Figs. 13 and 14. Fig. 13 shows the migration time for different notification interval times tested 10 times: the migration time is less than 200 ms if the query interval time is 100 ms. Fig. 14 shows the average migration time for different query intervals tested 10 times: the application migration time increases almost linearly with the query interval time.

Simulation 4: *In this experiment, we also use two clients to simulate a scenario of live application sharing. The environment is the same, and the client does not have a public IP address. Based on our designing, the sharing time will add an extra VM cloning time compared with the application migration scenario. Therefore, we mainly test the VM clone time under different sizes of VM image.*

When a VM is cloned, the VM Manager needs to save the VM's memory mirroring, copy the disk mirroring, and restart the VM. Therefore, the clone time consists of three parts.

- $\boldsymbol{T}_1$: Time for saving the memory mirroring file.
- $\boldsymbol{T}_2$: Time for copying VM disk COW (copy-on-write) file.
- $\boldsymbol{T}_3$: Time for restarting the VM.

The selected Windows XP VM memory is 512 M in our experiment. After 20 repeated tests, we get the average $\boldsymbol{T}_1$ is 9.308 s and $\boldsymbol{T}_3$ is 0.879 s. $\boldsymbol{T}_2$ is related to the size of VM disk mirroring file. We tested the time cost for copying COW files of size 64 M, 128 M, 256 M, 512 M, and 1024 M, respectively; the average time cost is shown in Table 6 and Fig. 15.

As shown in Fig. 15, $\boldsymbol{T}_2$ increases linearly with the increasing size of the COW disk mirroring file. In summary, the virtualized

Table 6
Average time cost for copying disk mirroring files of different size.

Size	64 M	128 M	256 M	512 M	1 G	2 G
T2b	0.122 s	1.611 s	3.805 s	8.973 s	17.391 s	42.713 s

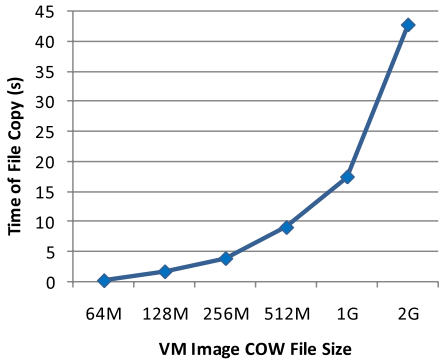


Fig. 15. Time cost for copying disk mirroring files with different image sizes.

application's sharing time is mainly related to the following two factors: notification query time interval in the Client Controller and the size of disk COW image of the VM.

5. Related work

Desktop virtualization technology provides access to cloud computing environments from anywhere in the world, on whatever operating systems. It has become an irresistible trend, and is ranked second among the ten hottest technologies selected by InfoWorld in 2010 [9].

Microsoft has launched *Client-Hosted Desktop Virtualization* and the *Server-Based Desktop Virtualization*. The latter virtualization technology allows the separation of software execution and presentation by adopting some remote desktop protocols, such as RDP. *Virtual Desktop Infrastructure* (*VDI*), a desktop delivery model developed by Microsoft, allows users to access desktops running in the datacenter. *VMware View* is developed by VMware to achieve isolation of the operating system, applications, and user data, which avoids problems brought by the tightly coupled architecture. Citrix [6] also launched *XenDesktop* to achieve desktop virtualization. A *FlexCast* technology is used to meet the different demands of desktop environments for different users in an enterprise. These products break the traditional tightly coupled software execution environment, and provide flexible desktop access approaches. However, they do not meet the demands for live application sharing and migration or secure sharing of a virtual desktop. Besides, these products also face a compatibility problem under different operation systems.

The representative projects on desktop virtualization include THINC [9,8,10], Citrix XenDesktop [6,11], Microsoft Terminal Service [6] and some VNC systems [5,12]. THINC is a remote display system architecture for high-performance thin-client computing in both LAN and WAN environments. THINC enables higher-level graphics primitives used by applications to be transparently mapped to a few simple low-level primitives that can be implemented easily and efficiently. Citrix provides full VDC (Virtual Desktop Computing) using their ICA protocol in parallel with the Ardence image and provisioning manager and desktop server hypervisor. Recently, XenClient has extended the benefits of desktop virtualization to mobile users, offering improved control for IT with increased flexibility for users. RDP enhancements in Windows Server 2008 and in recent MS client Operating Systems will also address some of the problems identified in relation to video and other graphics-intensive applications over RDP. VNC [4,13–15] is based on the PRB protocol, which is a simple and powerful remote display protocol. Unlike other remote display protocols such as the *X* Window System and Citrix's ICA, the VNC protocol is totally independent of operating system, windowing system, and applications. RealVNC [12] proposes different remote display solutions for client access: the software is executed at remote servers; the user's client just gets the presentation desktop. This solution only focuses on the separation of execution and presentation, and does not involve software deployment and execution-related fields. MetaVNC [5] pursues a remote desktop environment on which users can control applications on different hosts seamlessly. MetaVNC is a window-aware VNC, and it merges windows from multiple remote desktops into a single desktop screen.

In addition, some products and research work have emerged to address the software service requirements for mobile equipment in recent years. Microsoft Application Virtualization (App-*V*, previously named SoftGrid) is a core component of the Microsoft desktop optimization pack for software assurance: it transforms applications into centrally managed virtual services that are never installed and do not conflict with other applications. The Progressive Deployment System (PDS) [16], Yang's work [17], and FVM [18] employ OS-level virtualization technology to reduce the deploying, updating, and management labor cost of IT as well as the execution environment isolation. All the virtual software packages are managed at central server sites. When a user wants to use some software, the software package will be delivered to the local machine in a streaming way. MobiDesk [19] is a mobile virtual desktop computing hosting infrastructure, and it transparently virtualizes a user's computing session by abstracting underlying system resources in three key areas: display, operating system, and network. It provides a thin virtualization layer that decouples a user's computing session from any particular end-user device, and moves all application logic to hosting providers.

In summary, there are some desktop virtualization approaches to providing remote access to a cloud computing environment. However, these approaches only focus on displaying the remote desktop, and do not consider flexible collaboration for live application sharing and migration.

6. Conclusion

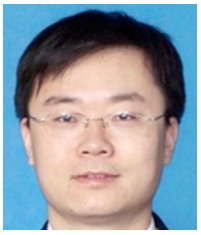
In a cloud computing environment, users can get SaaS subscriptions instead of traditional perpetual-use licenses from software vendors. We have developed a dynamic prototype system named CyberLiveApp to support application sharing and migration on demand among multiple clients. CyberLiveApp provides two key services: a secure multi-user sharing service for the virtual desktop of a VM and multi-VM application sharing and migration. We designed a mechanism for tracking window operation events, and the hidden window areas can be computed quickly. To filter various windows, a proxy-based filtering mechanism is used to deliver a desktop to different users. To achieve the goals of live application sharing and migration between VMs, a presentation redirection approach based on VNC protocol and a VM cloning service based on the Libvirt interface are used. All these methods have been implemented in the CyberLiveApp prototype based on extended MetaVNC and the virtual machine monitor KVM. We experimentally verified that these approaches are effective and useful. Several extensions will be made for future work. We are currently developing virtualization-based software as a service platform, and we are exploring how to integrate CyberLiveApp into some cloud-based computing environments for flexible collaboration.

Acknowledgments

The authors gratefully acknowledge the anonymous reviewers for their helpful suggestions and comments, and thank Shuang Yang, Yanmin Zhu, Liang Zhong, and Jin Li for their help with this work. This work is partially supported by Program for New Century Excellent Talents in University 2010 and the Fundamental Research Funds for the Central Universities, National Nature Science Foundation of China (No. 60903149, 91018004), and China 973 Fundamental R&D Program (No. 2011CB302602).

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, 2009.
[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona brandic: cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (6) (2009) 599–616.
[3] M. Turner, D. Budgen, P. BreretonTristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper, Turning software into a service, IEEE Computer 36 (10) (2003) 38–44. http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/docs/att/tr.98.1.pdf.
[4] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, Andy Hopper, Virtual network computing, IEEE Internet Computing 2 (1) (1998) 33–38. http://www.cl.cam.ac.uk/Research/DTG/attarchive/pub/docs/att/tr.98.1.pdf.
[5] MetaVNC, a part of the Collective at Stanford University http://metavnc.sourceforge.net/.
[6] T.W. Mathers, S.P. Genoway, Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame, Macmillan Technical Publishing, Indianapolis, IN, 1998.
[7] Jianxin Li, Jinpeng Huai, Chunming Hu, Yanming Zhu, A secure collaboration service for dynamic virtual organizations, Information Sciences 180 (17) (2010) 3086–3107.
[8] Albert Lai, Jason Nieh, Bhagyashree Bohra, Vijayarka Nandikonda, Abhishek P. Surana, Suchita Varshneya, Improving web browsing on wireless PDAs using thin-client computing, in: Proceedings of the 13th International World Wide Web Conference, WWW 2004, New York, NY, May 17–22, 2004, pp. 143–154.
[9] Info World Test Center staff. InfoWorld's 2010 Technology of the Year Awards[Z]. Info world,2010.
[10] Albert Lai, Jason Nieh, On the performance of wide-area thin-client computing, ACM Transactions on Computer Systems (TOCS) 24 (2) (2006) 175–209.
[11] Citrix Systems—Virtualization, Networking and Cloud. http://www.citrix.com/.
[12] RealVNC – VNC® remote control software, http://www.realvnc.com/.
[13] Tom Wall, Virtualisation and thin client: a survey of virtual desktop environments, Technical Report, Dublin Institute of Technology, 2009, http://arrow.dit.ie/ahfrcart/5/.
[14] C. Taylor, J. Pasquale, Improving video performance in VNC under high latency conditions, 2010, in: International Symposium on Collaborative Technologies and Systems, CTS, 17–21 May, 2010, pp. 26–35.
[15] Oren Laadan, Ricardo Baratto, Dan Phung, Shaya Potter, Jason Nieh, DejaView: a personal virtual computer recorder, in: Proceedings of the 21st ACM Symposium on Operating Systems Principles, SOSP 2007, Stevenson, WA, October 14–17, 2007, pp. 279–292.
[16] Alpern, Bowen, Joshua Auerbach, et al. PDS: a virtual execution environment for software deployment, in: Proceedings of the First ACM/USENIX International Conference on Virtual Execution Environment, March, 2005.
[17] Yu, Yang, Fanglu Guo, Susanta Nanda, Lapchung Lam, Tzi-cker Chiueh, A Feather-weight virtual machine for windows applications, in: Proceedings of the Second ACM/USENIX Conference on Virtual Execution Environments, VEE'06, June, 2006.
[18] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, J. Norris, M.S. Lam, M. Rosenblum, Virtual appliances for deploying and maintaining software, in: Proceedings of Seventeenth USENIX Large Installation System Administration Conference, October, 2003.
[19] Ricardo A. Baratto, Shaya Potter, Gong Su, Jason Nieh, MobiDesk: mobile virtual desktop computing, in: Proceedings of the 10th annual international conference on Mobile computing and networking, MobiCom '04, ACM, New York, NY, USA, 2004, pp. 1–15.



Jianxin Li is an associate professor of Beihang University. He received his Doctor's degree in Computer Software and Theory from Beihang University in 2008. His research interests include information security, virtualization, and Web Service. He is an editorial board member of the Journal of Cloud Computing. He has published over 30 papers in journals including Elsevier Information Sciences, Journal of Peer-to-Peer Networking and Applications, HASE 2008, SRDS 2007, and eScience 2006. He also is a program committee member of international conference IEEE Cloud 2009–2011, IEEE SCC 2008/2010, SNPD 2010, and others. He is member of IEEE.

Yu Jia received his Bachelor's degree in computer science from Beihang University, China, in 2010. She is currently an M.S. student in the Department of Computer Science, Beihang University, China. Her research interests include a broad range of topics related to cloud computing, including virtualization, SaaS, and virtual desktops.

Lu Liu is a Senior Lecturer in the School of Computing and Mathematics, University of Derby (UK). Before joining the University of Derby, he was Lecturer in the School of Engineering and Information Sciences at Middlesex University (UK). Previously, he was a Research Fellow in the School of Computing at the University of Leeds (UK), working on the NECTISE Project, which was a UK EPSRC/BAE Systems funded research project involving ten UK Universities, and on the CoLaB Project, which was funded by UK EPSRC and China 863 Program. He received his Ph.D. degree (funded by UK DIF DTC) from the University of Surrey (UK) and his M.Sc. degree from Brunel University (UK). His research interests are in areas of service-oriented computing, software engineering, Grid computing and peer-to-peer computing. Dr Liu has over 50 scientific publications in reputable journals, academic books, and international conference proceedings. He won the Best Paper Award at the Realising Network Enabled Capability Conference in 2008. He is member of IEEE.

Tianyu Wo received his B.Eng. and Ph.D. degrees, both in Computer Science, from Beihang University, China, in 2001 and 2008, respectively. Heis currently an Assistant Professor in the School of Computer Science and Engineering, Beihang University. His current research interests include large-scale distributed systems, virtual computing environments, network operation systems, and network-enabling applications. He is a member of IEEE.