Tsinghua Science and Technology

Volume 18 | Issue 1 Article 2

2013

Transparent Computing: Spatio-Temporal Extension on von Neumann Architecture for Cloud Services

Yaoxue Zhang
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China School of Information Science and Engineering, Central South University, Changsha 410083, China.

Yuezhi Zhou
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 02/12 pp10-21
Volume 18, Number 1, February 2013

Transparent Computing: Spatio-Temporal Extension on von Neumann Architecture for Cloud Services

Yaoxue Zhang and Yuezhi Zhou*

Abstract: The rapid advancements in hardware, software, and computer networks have facilitated the shift of the computing paradigm from mainframe to cloud computing, in which users can get their desired services anytime, anywhere, and by any means. However, cloud computing also presents many challenges, one of which is the difficulty in allowing users to freely obtain desired services, such as heterogeneous OSes and applications, via different light-weight devices. We have proposed a new paradigm by spatio-temporally extending the von Neumann architecture, called transparent computing, to centrally store and manage the commodity programs including OS codes, while streaming them to be run in non-state clients. This leads to a service-centric computing environment, in which users can select the desired services on demand, without concern for these services' administration, such as their installation, maintenance, management, and upgrade. In this paper, we introduce a novel concept, namely Meta OS, to support such program streaming through a distributed 4VP$^+$ platform. Based on this platform, a pilot system has been implemented, which supports Windows and Linux environments. We verify the effectiveness of the platform through both real deployments and testbed experiments. The evaluation results suggest that the 4VP$^+$ platform is a feasible and promising solution for the future computing infrastructure for cloud services.

Key words: transparent computing; extended von Neumann architecture; cloud computing; cloud services; Meta OS

1 Introduction

In the last two decades of the 20th century, with rapid advances in hardware and software, the centralized computing model of mainframe computing has shifted toward the more distributed model of desktop computing. Recent proliferation of special-purpose computing devices, such as laptops, tablet computers, smart phones, handhelds, and wearables, marks a departure from the established traditional general-purpose desktop computing toward the greatly heterogeneous and scalable cloud computing[1,2], in which diverse services can be accessed anywhere, anytime, from a variety of clients, including popular Personal Computers (PC). However, analysis of service access and support platforms from a client perspective indicates that maintaining and managing service operating environments on clients remain a challenge for end-users. It has been previously shown that the annual cost of managing a PC can be up to five times the cost of deploying it[3]. Meanwhile, all files and data are stored on the local disks of individual machines. They may be lost once the corresponding device is damaged

• Yaoxue Zhang and Yuezhi Zhou are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: zyx@moe.edu.cn; zhouyz@mail.tsinghua.edu.cn.
• Yaoxue Zhang is also with the the School of Information Science and Engineering, Central South University, Changsha 410083, China.
∗ To whom correspondence should be addressed.
Manuscript received: 2012-12-15; accepted: 2013-01-04

or compromised, requiring distributed data backup and restoration services. Furthermore, if sensitive server data are fetched and cached at local disks, they will potentially be available to the public, or to attackers with access to the machine.

The power of cloud computing has recently also been recognized to address the above challenges faced by traditional computing paradigms. While there are different types of usage, the models can be roughly classified into two categories. The first category is that in which the application software, such as Salesforce[4] and Google Docs[5], is hosted in data centers, and delivered to end-users through the web browser. This new paradigm can sharply reduce the cost of software maintenance and management, by centralizing both in the data centers. However, these application programs in cloud computing are specialized and dedicated, making it very difficult for traditional applications (e.g., MS Word) to be hosted and delivered. In addition, this only solves the maintenance and management issues of specific applications, which are not concerned with traditional OSes such as Windows.

The other category is that in which a Virtual Machine (VM) based thin-client approach emerges as virtual desktop solutions in data centers, such as Xen Desktop[6] and VMware View[7], which create virtual PCs/desktops (i.e., instances of Windows) on the server or server blade with virtualization technology. Thus, the user has a complete virtual PC in the data center or cloud, but only consumes a fraction of the server resources. The virtual desktop can be accessed from any client devices, whether normal PCs, thin clients, or mobile devices, through a Remote Desktop Protocol (RDP)[8], Independent Computing Architecture (ICA)[9], or virtual network computing (VNC)[10]. Compared with traditional thin-client systems, a virtual PC/desktop can guarantee and isolate user performance and improve security. However, as a type of thin client, it is difficult to support graphics-intensive applications, such as multimedia applications, due to the huge network bandwidth needed to transfer video display data, even in an enterprise environment.

To address these challenges, we have proposed a new computing paradigm, namely, transparent computing[11]. The core idea of this paradigm is to realize the "stored program concept" model in networking environments, in which execution and storage of programs are separated in different computers. Specifically, as illustrated in Fig. 1,

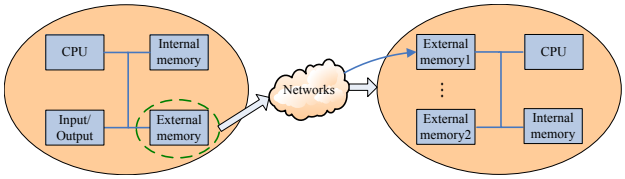

Fig. 1 An extended von Neumann architecture (I).

system/service programs are stored on central servers (like a warehouse), to be streamed on demand, and automatically initiated/executed on light-weight devices or clients with local CPU and memory resources (like a factory). Users can select any desired services, including commodity OSes and applications, via the same hardware platform and can use them according to their past experience. They do not need to undertake, or even bother about the installation, maintenance, and management of the services. In this paper, we present a realization of such a paradigm with a novel approach, namely Meta OS, mainly in local area network environments. This pilot system can then be further generalized to ubiquitous networks.

The rest of the paper is organized as follows. Section 2 introduces the basic motivation, i.e., the concept of transparent computing in comparison with other computing systems. Section 3 presents Meta OS and its architectural details. Section 4 describes the detailed design issues of 4VP$^+$, which is a distributed platform embodying Meta OS. The proposed platform is evaluated via real deployments and experiments of a pilot system in Section 5. Finally, conclusions and future work are addressed in the final section.

2 Overview of Transparent Computing

2.1 An extended von neumann architecture

The objective of transparent computing paradigm is meeting user demands for any computing services in a trouble-free way in the right place at right time. The essential idea of transparent computing is to spatio-temporally extend the traditional stored program concept in a single computer to networked computers/devices. Specifically, the storage and execution of programs are separated to be on different computers. Programs are stored on the central computers (servers) as opposed to local storage devices in traditional computers. The programs are streamed and scheduled on demand, to be executed on any computers (embedded devices or clients) with their local CPU and memory resources.

To further describe this paradigm, let us first examine the traditional concept of stored program computer. The design of a stored program concept (also known as the von Neumann architecture) computer is organized with five main components: control unit, arithmetic logic unit, memory (including internal and external parts), input, and output. The instructions and data are treated and stored in the same way in the storage devices, and will be fetched first from persistent storage (through input and output components) to memory, and then processed in its arithmetic unit[12].

A model based on a single local storage structure to hold both instructions and data (also known as programs) can achieve great flexibility, for example, easily replacing and changing programs. However, using this traditional way of storage with local devices, it is difficult to provide ubiquitous or personalized or active services to users. Several disadvantages of this approach are illustrated from different aspects below. First, with only local storage, the small, limited sized embedded devices can not hold all the necessary programs, while the relatively larger devices can not share programs beyond their contents. Second, programs on the local storage need maintenance, protection against malware (including virus, worms, and spyware), and other management routines, which are usually very difficult for non-expert users to undertake. Third, the mechanical components of local storage, such as the hard disk, are often broken and become sources of system failure. With the advent of networking, many approaches are being proposed based on network technologies to solve the problem of data sharing and partially program sharing[13, 14]. However, using these approaches, it seems difficult or impossible to fundamentally solve the program sharing problem, and thus to provide users with desired services freely. For example, it is difficult to share commodity OSes and applications (e.g., Windows) with a bare hardware platform and let users select other software services on a Linux platform.

To have fundamental solutions to the above problems, it is necessary to extend the von Neumann architecture to network environments. This extended architecture for transparent computing is further elaborated in Fig. 2. The spatio-temporal extension can be seen as follows. The storage and computing of programs are separated in different computers, as opposed to different components in a same local computer in the traditional von Neumann architecture. One computer

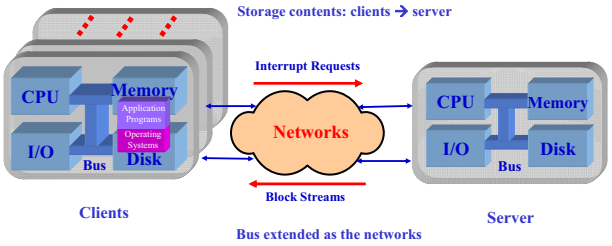

Fig. 2 An extended von Neumann architecture (II).

(client) is responsible for computing only, and shares the programs (including OSes and applications) stored on another computer (server) via the network. The servers act only as a repository of all needed programs (including OSes and applications). The programs (including OSes and applications) will be streamed back to the client in small streaming blocks on demand via the network, to be executed with the local resources, such as memory and CPU. In such a spatio-temporally extended architecture, the storage contents are extended from the local computer (client) to another computer (server) via the network, and the interrupt requests for I/O and storage, and system bus have been extended from within the local computer (client) to the network, as well as the file and user management.

Accordingly, on such spatio-temporally extended architecture, client/embedded devices require less CPU power and memory, which reduces the hardware cost significantly (easy to maintain); client devices can run different programs (including OSes and applications), which drastically reduces the software cost (easy to use); a program (including OS and application) can be shared by many users, namely a service-centric computing environment (SaaS), which reduces the software cost (easy to use); the centralized management can significantly reduce the costs of maintenance, upgrade, security, usage, and management as well. Therefore, a user can select any needed OS (e.g., Linux or Windows or an embedded OS) and applications stored on the servers via the same client machine. Also, a user can obtain the same service (e.g., applications on Windows) via different client machines. Moreover, all computing services and technologies are transparent to users. Users can select and use their desired services, without being involved in all the maintenance, upgrade, and management, which will be carried out by the system itself or centrally on the servers by administrators. The detailed description is shown in Fig. 3.

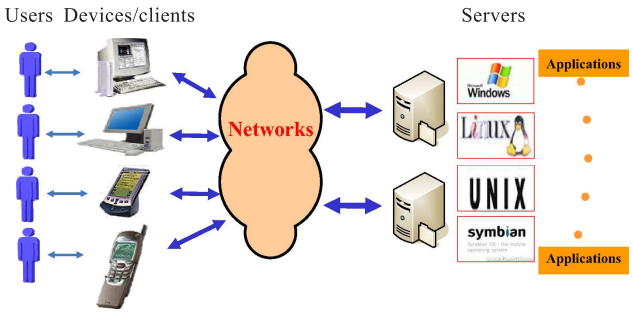

Fig. 3 Same/different services via different/same clients.

2.2 Related work and difference

There has been extensive research in this area. Our proposed transparent computing paradigm and the extended von Neumann architecture are most related to the previously proposed systems such as PCs, Network Computers (NC), thin clients, diskless computers, VMs, and cloud computing.

Because of the management challenge of PCs, NCs such as the Java Station by Sun[15] have been proposed as replacements. Such a solution supports very limited applications, such as WWW and Java applications only, and does not work with general commodity OSes or other applications, such as the Microsoft Office suite[16].

Thin-client computing systems, such as Microsoft RDP[8], Citrix ICA[9], Sun Ray 1[15], VNC[10], and MobiDesk[17], have been very popular for providing a fully featured desktop to users at low management cost. However, in such a computing paradigm, all computing tasks are performed at the central server (which is different in our proposed approach, namely that storage and computing are separated), while a client works only as a user interface by performing display, and keyboard/mouse input/output functions, but with no computing capability. Although realized by the centralized management, such systems greatly increase the server resource requirements with limited scalability. Applications with heavy computing requirements (e.g., multimedia applications) usually can not be efficiently supported by thin-client systems.

Recently developed VM-based thin-client approaches in cloud computing, such as Xen Desktop[6] and VMware View[7], also leverage the thin-client model to access the virtual PC or desktop hosted in data centers. These virtual PCs or desktops are hosted by a VM created and run within a server. Although they can guarantee and isolate user performance and achieve higher security with the help of VM technologies, it is still difficult for them to support graphics-intensive applications due to the large network bandwidth needed to transfer video display data over the Internet.

Our idea is quite similar to the concept of diskless computers such as those described in Refs. [18-20] in early years. Without local hard disks, a diskless computer usually downloads an OS kernel image from the remote server. It thus can not support OSes that do not have clear kernel images, such as Windows. It also does not support booting from heterogeneous OSes.

The concept of resource virtualization was introduced a long time ago and has recently been more popular in addressing security, flexibility, and user mobility. One evident example is that VMware[21] has extended the concept of virtual machines to support multiple commodity platforms. VM-based stateless thick client approaches, such as Internet Suspend/Resume (ISR)[22], use VM technology (e.g., VMware) together with a network file system (e.g., Coda[23]) to support user mobility. Each ISR client runs OS and applications on top of a pre-installed VMware on the host OS. The use of VMs supports heterogeneous OSes as well, but it also introduces additional performance overhead due to its virtualization of all hardware resources, including CPU and memory, while in our paradigm, client OSes are running directly on top of the CPU and memory resource. In additional, their approach still keeps the computing and storage in the same local clients. Due to the nature of running OS and applications on top of a pre-installed VMware on the host OS, their solution still can not be efficiently applied for cloud applications, since most of the clients are embedded systems or devices.

The cloud services, such as Salesforce[4], Gmail[24], Google Docs[5], centralize both computation and storage in data centers and then deliver the applications to end-users through the web browser or other special utilities. This new paradigm can sharply reduce the cost of software maintenance and management by centralizing both in the data centers. However, these application programs in cloud computing are specialized and dedicated, making it very difficult for traditional applications (e.g., MS Word) to be hosted and delivered. In addition, this just solves the maintenance and management issues of specific applications, and is not concerned with traditional OSes, such as Windows.

3 Concept and Architecture of Meta OS (4VP+)

To realize transparent computing as stated above, we introduce a novel concept of Meta OS, a super-OS control platform running beneath operating systems, but which can instantiate (startup, serve, and control) commodity OSes (Instance OSes) and their applications. The conceptual model of Meta OS, in comparison with the conceptual model of single OS, can be illustrated in Fig. 4. To implement the separation of execution and storage of programs, the Meta OS should have two basic functions. The first is to let users select which OS should be loaded and used, and then boot the needed OS remotely from server repositories. The second is to stream programs demanded by users during or after the OS startup. This is vital because of lack of local storage in the client. The concept of Meta OS can be embodied through a distributed platform with several parts residing on the client and several parts residing on the server, while other parts are integrated with the Instance OSes.

As discussed in the previous section, the conceptual description on the topology of the Meta OS has been partially illustrated in Fig. 4. As shown in this figure, the servers (transparent server) are regarded as the resource providers that supply OSes, application software, and data to bare clients who need them. The clients demand the programs (including OSes and applications) through the networks (delivery network) in a streaming way, and execute them (just-in-time computing) by using the local resources. This paradigm is different from any other computing paradigms currently used in PC, diskless workstation or conventional client/server paradigm, or embedded devices, since it enables accessing and sharing different OSes, applications, and data stored on centralized servers via the same bare client.

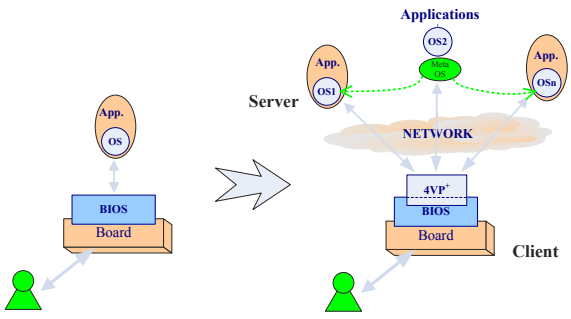

Fig. 4 General models of Meta OS and Single OS.

The layer architecture of a transparent computing system based on Meta OS is given in Fig. 5. The client must be fitted with a resident module of Multi-OS Remote Boot Protocol (MRBP). When the client is powered up, it will issue a booting request to the server and use MRBP to start up an OS (Instance OS, integrated with Meta OS) selected by a user. After the Instance OS is loaded, the Meta OS and Just-In-Time Computing (JITC) layer in the client will continue to serve the above Instant OS in a transparent way, which means that the commodity OSes can run continuously without awareness of the change.

In the server, the Meta OS layer, running on top of the server OS, holds a set of basic services and management tools, such as Virtual Disk Management System (VDMS), Network Service Accessing Protocol (NSAP), and MRBP service. The heterogeneous OS layers are used to store and support different types of OSes. The application layer refers to common applications running on the server.

The Meta OS consists of four virtual views of I/O, disk, file and users, and two protocols of MRBP and NSAP. Their detailed design and functions are presented in Section 4.

4 Program Streaming Through 4VP+

The embodiment of Meta OS is through a 4VP+ (shortly for four virtual layers and two protocols) distributed platform, which partially operates at the assembler instruction level. The structure of 4VP+ is shown in Fig. 6. This 4VP+ software platform is mostly installed in a management server, except for a part of MRBP that is burned into a BIOS EEPROM in the bare client. However, the other programs of 4VP+ platform run in clients or servers according to their functions. The parts

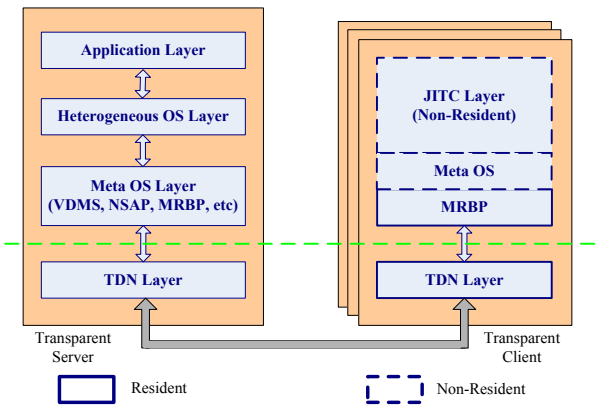

Fig. 5 Layered architecture of transparent computing.

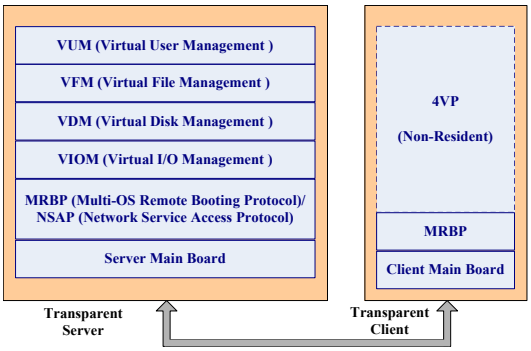

Fig. 6 Modules of 4VP+.

of 4VP+ running in the client will be also fetched from the server repository along with the Instance OSes.

The MRBP is used to boot bare clients remotely from servers. It can let users select their desired OSes and applications. The MRBP then installs an NSAP module, which is written in assembler instructions, to enable a virtual I/O for clients. Through this virtual I/O interface, clients can access the OS images located at servers and then load them as using a regular I/O device. After the dynamically downloaded OS is started up, the OS-specific in-kernel modules for virtual I/O (V-IO), virtual disk (V-disk), virtual file (V-file), and virtual users (V-user) will function to help the users to access the software and carry out their tasks seamlessly. The V-user module is used to manage users. Before users' setting up connections to a server to access their private information or data, the system has to authenticate them through a network authentication protocol, such as Kerberos[25] with the V-user manager in the server. If it succeeds, the server will serve the clients through the NSAP. Note that users can access their information via any clients in the system for convenience. In the remainder of this section, we discuss the two protocols and four virtualizations in detail.

4.1 Multi-OS remote boot protocol

In lack of local storage to persistently keep software and data during power-off, the transparent clients have to dynamically load the desired OS environment from the server. To support heterogeneous OS environments, we can not adopt traditional methods for booting client machines without local hard disks, for example, burning a specific OS kernel into a client Real-Only Memory (ROM)[15], or a more flexible way by downloading the OS kernel directly from the server[18] that can not support OSes like Windows since it does not have a clear kernel.

Our key idea is to first initialize a virtual I/O device for a client, whose requests will be sent through a network and executed by a server instead of a local hard disk. In this way, any OS can be loaded according to its normal procedure as if a real hard disk existed. Figure 7 lists the main steps involved in the remote boot process in the server and client, respectively.

After having powered on, the client interrupts the normal booting up process by starting up the MRBP client. The MRBP client will send boot requests to servers, discover the supported OSes in the system, and display them for users. After a user selects a specific OS, the MRBP client will then download and start up the NSAP client, through which the client can access the virtual disk images on the server repositories through virtual I/O devices. The client then executes the Master Boot Record (MBR) of the specific virtual disk, continuing the booting process as if there was a normal hard disk. The services, residing on server, such as MRBP and NSAP, will serve the booting process accordingly.

Before the client is powered off, the V-disk driver and V-file modules will flush out the cache contents to the server. If the power-off happened beyond schedule, the system can also startup in a clean state by clearing the Copy-On-Write (COW) disk, as shown later in the following subsections.

4.2 Network service access protocol

The core idea of 4VP+ is virtual disks. From the perspective of a user or application, there is no difference between accessing data from a V-disk and a local hard disk. The actual contents in V-disks reside at

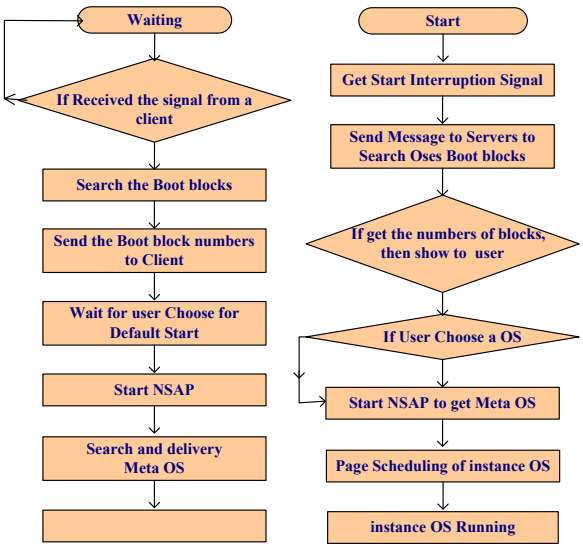

Fig. 7 Boot process of Meta OS and Instance OS.

the remote server, and will be fetched to the client on demand. The access of virtual disk images on the server is through NSAP.

A virtual disk request issued by the above OS or file system will be changed into one or more NSAP packets to be sent to the remote server and responded to from the server as well. Thus, given each virtual disk request received from the file system, the NSAP client (through an OS-specific V-disk driver) will compose one or more remote-disk requests to be sent to the server.

The first function of NSAP is to establish a unique connection between the V-disk image on the server side and the V-disk on the client side. Consequently, each client can maintain two request queues, one for the virtual disk requests received from the file system, and the other for the remote-disk requests to be sent to the server.

The second function of NSAP is to deliver the instructions and data, including OS codes from the server to clients, or the computation results from clients to the server. These transmissions occur when interruptions or I/O requisitions are made.

Figure 8 shows the format of each remote-disk request and reply in an application-level packet. Each packet starts with an identification field of four bytes to denote the unique packet sequence number. The two-byte operation mode is used to distinguish among three different types of remote-disk packet: read requests, write requests, and server acknowledgments. The logical block number and block length specify the start block number and the total number of blocks to be read or written on the corresponding V-disk. If the packet is a write request or a read response, then the actual disk data to be written or read will follow.

The NSAP uses User Datagram Protocol (UDP) to deliver data packages. Thus, to maintain reliable data transmission, the NSAP server will send an explicit acknowledgment for each remote-disk request received. If the request is a data-read request, the acknowledgment can be piggy-backed with the returned data. NSAP clients use timeouts to detect lost remote-disk requests or replies for retransmission requests. Each remote-disk request has a unique ID, so

Identification (4 bytes)	Operation mode (2 bytes)
Logical block number (4 bytes)	Block length (2 bytes)
Data (optional, maximum 32 KB)	

Fig. 8 Format of NSAP request/response packets.

that both the client and the server can detect and drop duplicate packets. For achieving good transmission performance, it is necessary to set a small but proper timeout value in case of network loss. Our empirical experience has shown that most disk accesses are involved with small amounts of data. Thus, we set the maximum size of data to be read or written by a remote-disk request to 32 KB.

To avoid out of order requests, each client uses a simple wait-and-send solution to send remote-disk requests. After a request is removed from the queue and sent to the server, the client will not send the next request until it receives the response to the last message. Hence, a read request following a lost write request will not return stale data, and repeated read/write requests are guaranteed to generate the same effects.

4.3 Virtualizations and management

After the above discussions of the two core protocols in 4VP$^+$ platform, we describe the four virtualizations in more details in this subsection.

As stated previously, the core technique of 4VP$^+$ is the virtualization of devices and users. For transparency to OSes and users, the virtual I/O is triggered with virtual disks. V-disks are flat addressable block-based virtual storage devices located beneath the file systems. Each block has a fixed length, such as 512 bytes. A transparent client can be configured to access data from one or more V-disks, with each corresponding to a V-disk image located on the server. A V-disk image consists of one or more files, depending on the required disk size. To manage different clients' V-disks, the transparent server sets up a V-disk management database to maintain the mappings between a client's IP address and the corresponding V-disk images. A system administrator can configure the quotas of client V-disks using an application tool that updates the server's V-disk management database. Note that a V-disk seen by users can be mapped to different images. This feature provides flexibility for sharing virtual images among different users.

The whole system virtualization can be illustrated in Fig. 9. The virtual disk access is implemented by the V-disk drivers and the Virtual I/O Management (VIOM) through NSAP interacting with the service modules and management database on the server. The virtual file system in the client is to enable software sharing. The components and their functions are further discussed

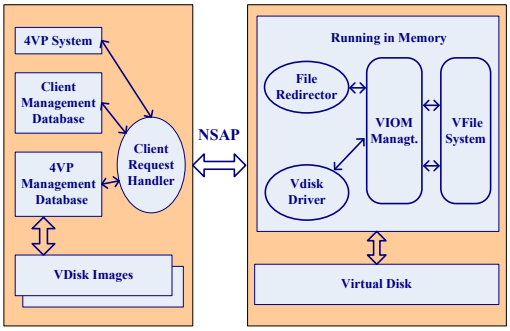

Fig. 9 Virtualizations in 4VP+.

below.

To facilitate effective management of centralized disk image files and to support heterogeneous OSes and applications with reduced complexity, the 4VP+ platform classifies client V-disks into four different categories to enable sharing and isolation. First, 4VP+ separates software from data, based on the observation that many users will use the same OS and application software and thus can share them, while data are often user-specific and can not be shared.

Second, 4VP+ maintains a "golden image" of a clean system that contains the desired OS and a common set of applications. This "golden image" is thus immutable and can be shared by all clients. However, some applications must write to the disk directories where they are installed to function properly, e.g., create temporary files. To support these applications, 4VP+ adopts a COW approach by having a COW V-disk for each client. The COW operations are implemented through a file system redirector. The file system redirector can filter the file written operations and redirect them to the COW disk. Of course, it also needs to carry out the read operations with the original or COW V-disk. Thus, there are four categories of V-disks in 4VP+:

System V-disk (S): It is used to store the "golden image" of OS and applications. The corresponding system virtual images are created by administrators, and shared by all clients. They can be modified only by the administrators.

Shadow V-disk (H): It is a user-specific COW disk of the system V-disk to enable write access to the system V-disk contents. However, the COW semantics are supported at the granularity of files through a file redirector, which is a file system level software agent. When a user needs to modify a file on the system V-disk, a COW copy of the file will be created on the shadow disk for any subsequent access. The use of shadow V-disks is transparent to the end-users.

Profile V-disk (P): Each client also has a profile V-disk to store user-specific persistent data, such as customized user settings for OS and applications. Similar to shadow V-disks, the existence of profile V-disks is also transparent to the users.

User V-disk (U): Each client has one or more user V-disks that are used to store private user data. Each user V-disk will be mapped to a user-specific image.

It is the classification of V-disks that greatly simplifies software management tasks, especially for system recovery. If a client is corrupted by accidental errors, software bugs, or malicious attacks such as viruses, worms, or spyware, system administrators can quickly clear the COW V-disk contents to return a clean system image to users.

5 System Implementation and Evaluation

We have implemented a prototype of 4VP+ that supports both Windows 2000 Professional and Red Flag 4.1 Desktop (Linux kernel 2.4.26-1)[26]. Our implementation of MRBP is based on, and compatible with, the Intel PXE protocol[27] for sending boot requests. Because device drivers are platform dependent, we have implemented two different V-disk drivers, customized for Windows and Linux, respectively.

The implementations are in C++. Since Windows 2000 is a modified microkernel, we have modified the corresponding Windows registry files for the OS to load the V-disk driver. Thus, there is no need to change or recompile the kernel. However, since Linux is a monolithic kernel, we have compiled the V-disk driver into the kernel by modifying the related kernel source code before recompilation.

5.1 Deployment environments and experiences

Our Windows-based system has been deployed across 30 clients in a university e-learning classroom for daily usage for 14 months.

In this deployment, the 4VP+ clients are Intel Celeron 1 GHz machines, each with 128 MB DDR 133 RAM and 100 Mbps onboard network card. The server is an Intel Pentium IV 2.8 GHz PC with 1 G RAM, a 1 Gbps network card, and two 80 GB 7200 rpm soft RAID0 hard disks. The clients and the server are connected by an Ethernet switch with 48 100 Mbps interfaces (used for clients) and two 1 Gbps interfaces (used for the server). The server OS is Windows 2003 Server

(SP1) edition. The 4VP+ clients use Windows 2000 professional (SP2).

During our initial deployment, the prototype system has been running stably most of the time and has achieved the following benefits:

Reduced system maintenance time: Previously, administrators spent on average 4 to 8 hours a week to regularly clear every machine, even with the help of automatic tools to fix problems caused by user faults or malicious attacks. Using 4VP+, the system cleaning and upgrading time is reduced to 30 minutes per week, due to both the reduced number of malicious attacks and the centralized operations.

Improved availability: Before using 4VP+, 4-8 hour system maintenance took place every Thursday. No class could be arranged to use the classroom during this maintenance period. After deploying 4VP+, the classroom can be used every day without weekly service interruptions.

Improved security: After deploying 4VP+, there have been no reported viruses or worms in the system. One physical theft did happen in the classroom of our deployment, where, in addition to the 30 4VP+ clients, there are also 30 other regular desktop computers. All the memory slots and hard disk drives of the 30 regular clients were stolen in this incident, resulting in significant data loss. As a contrast, all the 4VP+ transparent clients remained intact, except for one that suffered loss of a memory slot after the thief opened this single computer box and discovered no disk. No data were lost, and the transparent system resumed operating the very next day.

5.2 Testbed experiments and evaluations

In our testbed experiments, we used the same hardware configurations as in our real deployment, but with a more powerful server, an AMD Athlon64 3000+ machine. The server is configured with 2 GB Dual DDR 400 RAM, two 80 GB Seagate Barracuda 7200 rpm soft RAID0 hard disks, and 1 Gbps onboard network card. We examine the performance of V-disk in a single server and client scenario. We vary the number of clients supported by a 4VP+ server to study the application performance in concurrent cases. We also compare the performance with a regular PC (with the same hardware configuration and an additional local hard disk of 80 GB Seagate Barracuda 7200 rpm). To evaluate the system scalability, we compare it with typical thin client systems.

5.2.1 V-disk accessing performance

We first examine the throughput of accessing V-disk data. We evaluate the V-disk access throughput using the Iometer performance tool[28] to submit random disk access requests of different sizes to the client. We just show the results of the random, unbuffered case on Windows in Fig. 10 and Fig. 11. The unbuffered case means that the client's file system caches are disabled. We also compare the throughput with that of the regular PC's hard disk. For read access, V-disk throughput increases with the request size and is higher than the local disk, but decreases when the request size is larger than 32 KB, which is the maximum size delivered. Thus, this decrease may be because the network communication time dominates the latency, and a large request size will result in several service requests. At the same time, because the response in a 4VP+ system can be satisfied with the server's memory cache, the V-disk performance is higher than that of the local disk when the request size is small. The write throughput, as shown in Fig. 11, is smaller in the V-disk than the local disk because the server cache can not satisfy the requests instantly. Note that in the local disk scenario, the write throughput is bigger than the read. This may be because the embedded hardware cache of the hard disk can return the write result instantly without undertaking the real disk operations. We also evaluated the performance in the Red Flag Linux environment. The results are shown in Fig. 12 and Fig. 13. Since the lowest sector size in the Linux ext3 file system is 4 KB, Iometer can not test the 1 KB and 2 KB cases. We can see that the results are similar, but with a different less amount. This may be due to the different file systems and operating systems under measurement. In addition, we have tested the throughput in the buffer case, and the result is similar.

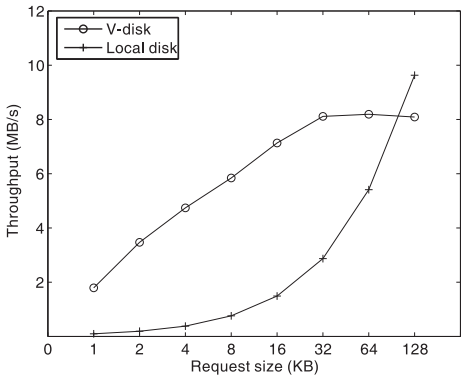

Fig. 10 Random read throughput in Windows.

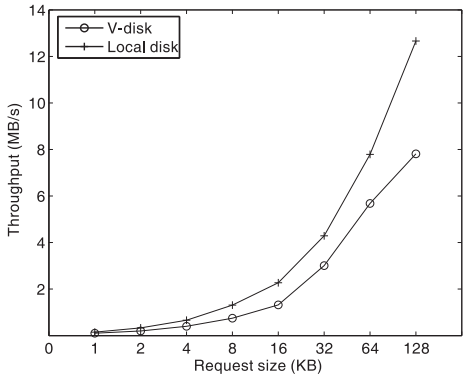

Fig. 11 Random write throughput in Windows.

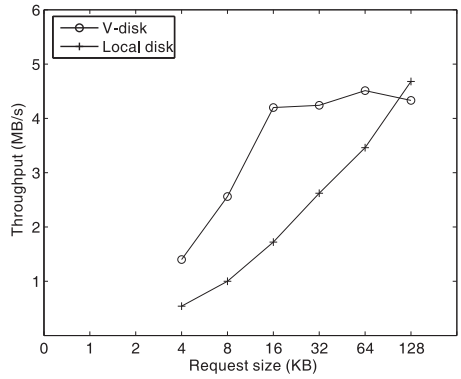

Fig. 12 Random read throughput in Linux.

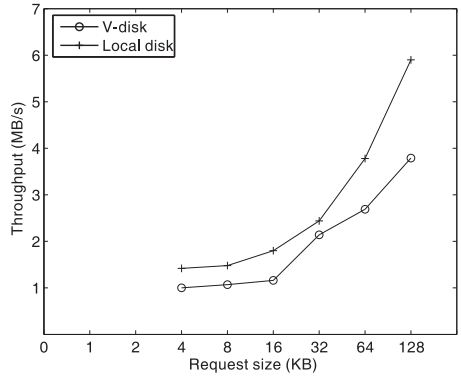

Fig. 13 Random write throughput in Linux.

5.2.2 Application performance

Tables 1 and 2 list the client access latency respectively in Windows and Linux environments, measured by concurrently running an operation on all clients in the following four categories: OS booting, launching office applications, launching other applications, and file copying. The OS boot latency refers to the time elapsed from powering on the client to the desktop windows displayed on screen. The application launch time refers to the time elapsed from starting the application till it is ready for use.

Table 1 4VP+ client access latency in Windows. (s)

Operation	PC	4VP+ clients		
		1 client	10 clients	20 clients
Booting OS	53.13	48.73	70.62	92.79
MS Word 2003 (Start up)	2.23	1.26	2.28	6.35
MS Word 2003 (Open 1 MB file)	3.07	2.13	3.57	7.27
MS PPT 2003 (Start up)	5.21	3.04	6.58	9.98
Photoshop V7.0 (Start up)	13.29	11.08	16.48	27.51
Flash V6.0 (Start up)	18.62	7.16	31.41	74.30
3D MAX V4.0 (Start up)	29.71	25.68	34.24	54.18
Copy a file (20 MB file)	11.59	8.95	19.75	37.51
Copy a file (50 MB file)	28.24	24.33	49.48	109.52

Table 2 4VP+ client access latency in Linux. (s)

Operation	PC	4VP+ clients		
		1 client	10 clients	20 clients
Booting OS	85.67	58.20	97.72	127.28
EIOffice (Start up)	2.03	1.07	2.35	6.03
EIOffice (Open 1 MB file)	3.01	2.09	3.37	7.18
Firefox (Start up)	3.67	2.55	3.89	7.99
Copy a file (20 MB file)	12.25	7.80	18.15	35.33
Copy a file (50 MB file)	33.83	25.44	53.76	115.79

For all four categories of performance shown in Table 1, we observe that in the Windows environment, 4VP+ outperforms the regular PC in the single client scenario. The latency increases with the number of clients within our range and is on the order of tens of seconds. The worst case latency is for the 20 clients to concurrently copy a 50 MB file. This is a V-disk intensive operation during which the network communication overhead was much higher and the server was more likely to become a bottleneck. However, such a worst case scenario rarely occurs in our real deployment, where it has been found by studying the traces collected from the server that there is strong locality of disk access patterns. As we can see from Table 2, the results in Linux environment are also similar only with a different amount.

5.2.3 Scalability

To study the scalability of the pilot system, we measure the web browsing performance with Microsoft IE 6.0, using the web text page load test provided by Ziff-Davis i-Bench benchmark suite 5.0[29]. We evaluated the average client latency of the entire i-Bench run by varying the number of concurrent clients in the system. We also compared the 4VP+ performance with the performance achieved by thin-client systems including Citrix ICA[9], Microsoft RDP[8]. The results are shown in the Fig. 14. We observe that when the

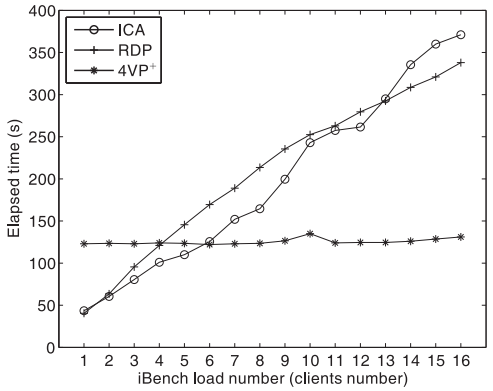

Fig. 14 Client observed i-Bench run latency.

number of clients is less than four, ICA and RDP achieve lower client latency than 4VP+. However, the latencies increase linearly with the number of clients in both ICA and RDP, whereas the client latency in 4VP+ remains constant with small deviations, suggesting that 4VP+ is more scalable than these thin-client systems. This is because in 4VP+, the server handles only V-disk access requests, while thin-client severs perform both file access and computing tasks. These results suggest that 4VP+ is a promising cost-effective solution for scalable real-world use.

6 Conclusions

We introduced a novel paradigm, transparent computing, which spatio-temporally extends the von Neumann architecture from the concept of "stored program" to the networking environments, in order to provide more active services for end-users toward cloud computing. Based on this paradigm, we have proposed that a Meta OS (4VP+) distributed platform can stream programs, including operating systems and their applications. The approach based on Meta OS can let users select and demand their desired services, like turning on and choosing different channels through TV-set, making computers more familiar and simple. Moreover, the system based on 4VP+ can achieve a lower cost than before, due to the low direct cost (hardware), and the low indirect cost, including cost of maintenance, upgrade, security, and usage.

We implemented the 4VP+ system in a client/server environment, and this system has been widely deployed in China. We have also given some evaluation results of the system. Both results from users and the evaluation results suggested that our system is a feasible and cost-effective solution for future computing infrastructure.

There are still several important aspects being investigated in our research studies. We have been extending our proposed approach to support broader embedded devices (e.g., PDA and intelligent mobile phone) to connect with ubiquitous communication networks, and to support more instance embedded OSes and applications. We will report further details in the near future, due to space limitations in the current manuscript.

Acknowledgements

This work was supported in part by the National High-Tech Research and Development (863) Program of China (No. 2011AA01A203), the National Key Basic Research and Development Program (973) in China (No. 2012BAH13F04), and the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology. The authors would like to thank the following people for their contributions to the development and experiments of the 4VP+ system: Guanfei Guo, Xiaohui Wang, Li Wei, Huajie Yang, Pengzhi Xu, and Nan Xia. We specially thank the people who have given their valuable comments and suggestions. We also thank the people and organizations who have deployed the system.

References

[1] B. Hayes, Cloud computing, *Commun. ACM*, vol. 51, no. 7, pp. 9-11, July 2008.
[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, Cloud computing and grid computing 360-degree compared, in *Proceedings of the Grid Computing Environments Workshop*, Austin, Texas, 2008, pp. 1-10.
[3] VMware Inc., Understanding full virtualization, paravirtualization, and hardware assist, White Paper, September 2007.
[4] Salesforce platform, http://www.salesforce.com, 2012.
[5] Google Docs, https://docs.google.com/, 2012.
[6] Citrix XenDesktop, http://www.citrix.com/virtualization/desktop/xendesktop.html, 2012.
[7] VMware Inc, VMware View 4.5 Modernize Desktop and Application Management V 2.0 Brochure, http://www.vmware.com/files/pdf/VMware-View-45-DS-EN.pdf, 2012.
[8] B. Cumberland, G. Carius, and A. Muir, *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Seattle: Microsoft Press, 1999.
[9] I. Boca, Citrix ICA Technology Brief, Technical White Paper, Boca Raton, 1999.
[10] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, Virtual network computing, *IEEE Internet Computing*, vol. 2, no. 1, pp. 33-38, 1998.
[11] Y. Zhang and Y. Zhou, Transparent computing: A new paradigm for pervasive computing, in *Proceeding of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC06)*, Wuhan, China, Sep. 2006, pp. 1-11.
[12] W. Aspray, The stored program concept, *IEEE Spectrum*, vol. 27, no. 9, pp. 51, Sept. 1990.

[13] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, Design and implementation of the sun network filesystem, in *USENIX Association Conference Proceedings*, Portland, USA, 1985, pp. 119-130.
[14] J. H. Howard, M. L. Kazar, and S. G. Menees, Scale and performance in a distributed file system, *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 51-81, 1988.
[15] Sun Microsystems Inc, Sun Ray Overview, White Paper, Version 2, http://www.sun.com/ sunray/whitepapers.html, 2012.
[16] R. G. Herrtwich and T. Kappner, Network computers — ubiquitous computing or dumb multimedia? in *Proceedings of the Third International Symposium on Autonomous Decentralized Systems*, Berlin, Germany, 1997, pp. 155-162.
[17] R. A. Baratto, S. Potter, G. Su, and J. Nieh, MobiDesk: Mobile virtual desktop computing, in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, New York, USA, 2004, pp. 1-15.
[18] D. R. Cheriton and W. Zwaenepoel, The distributed V kernel and its performance for diskless workstations, in *Proceeding of the 9th ACM Symposium on Operating Systems Principles*, Bretton Woods, N.H., USA, Oct. 1983, pp. 128-140.
[19] R. Linlayson, Bootstrap loading using TFTP, RFC 906, 1984.
[20] B. Croft and J. Gilmore, Bootstrap protocol (BOOTP), RFC 951, 1985.
[21] VMware GSX server, http://www.vmware.com/products/gsx, 2012.
[22] M. Kozuch and M. Satyanarayanan, Internet suspend/resume, in *Proceedings of the 4th IEEE Workshop Mobile Computing Systems and Applications*, New York, USA, 2005, pp. 40-48.
[23] M. Satyanarayanan, The evolution of coda, *ACM Transactions on Computer Systems*, vol. 20, no. 2, pp. 85-124, 2002.
[24] Gmail, http://www.gmail.com, 2012.
[25] B. C. Neuman and T. TSó, Kerberos: An authentication service for computer networks, *IEEE Communications*, vol. 32, no. 9, pp. 33-38, 1994.
[26] RedFlag Linux, http://www.redflag-linux.com/eindex.html, 2012.
[27] Preboot Execution Environment (PXE) Specification, ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf, 2012.
[28] Iometer, http://www.iometer.org, 2012.
[29] i-Bench, http://www.veritest.com/benchmarks/i-bench/, 2012.

Yaoxue Zhang received his Ph.D degree in computer networking from Tohoku University, Japan in 1989. Currently he is a professor at the Department of Computer Science and Technology at Tsinghua University and President of Central South University, China. He serves as an editorial board member of 4 international journals. He has won the National Award for Scientific and Technological Progress (2nd class) twice in 1998 and 2001, and National Award for Technological Invention (2nd class) in 2004, as well as 5 provincial or ministerial awards. He is a winner of the Prize of HLHL (Hong Kong) Foundation for Scientific and Technological Progress in 2005. He has also published over 200 technical papers in international journals and conferences, as well as 9 monographs and textbooks. His major research areas include computer networking, operating systems, ubiquitous/pervasive computing, transparent computing, and cloud computing.

Yuezhi Zhou received his PhD degree in Computer Science and Technology from Tsinghua University, China in 2004 and now works as an associate professor at the same department. His research interests include cloud computing, distributed system, mobile device and systems. He has published over 60 technical papers in international journals or conferences. He received the IEEE Best Paper Award in the 21st IEEE AINA International Conference in 2007.