Transparent Computing: a Spatio-temporal Extension on von Neumann Architecture for Ubiquitous Services

YAOXUE ZHANG and YUEZHI ZHOU
Tsinghua University, China
and
LAURENCE TIANRUO YANG
St. Francis Xavier University, Canada

With the rapid improvements in hardware, software and networks, the computing paradigm has also shifted from mainframe computing to ubiquitous or pervasive computing, in which users can get their desired services anytime, anywhere and any means. However, the emergence of ubiquitous or pervasive computing has brought many challenges, one of which is that it is hard to allow users to freely obtain desired services, such as heterogeneous OSes and applications via different light-weight embedded devices. We have proposed a new paradigm by extending the von Neumann architecture spatio-temporally, called Transparent Computing, to store and manage the commodity programs including OS codes centrally, 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 concerning these services' administrations, such as their installation, maintenance, management, upgrade, and so on.

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 and it supports Windows and Linux environments. We verify the effectiveness of the platform through both real deployments and testbed experiments. The evaluation results suggest that 4VP$^+$ platform is a feasible and promising solution of the future computing infrastructure for ubiquitous or pervasive services.

Categories and Subject Descriptors: ... [...]: ...

General Terms: ...
Additional Key Words and Phrases: Transparent computing, an extended von Neumann architecture, ubiquitous and pervasive services, Meta OS

1. INTRODUCTION

In the last two decades of 20th century, with the rapid advances in hardware and software, the centralized computing model of mainframe computing has shifted towards the more distributed model of desktop computing. Recent proliferation of special-purpose computing devices such as laptops, embedded systems, handhelds, wearables, etc., marks a

departure from the established traditional general-purpose desktop computing to greatly heterogeneous and scalable ubiquitous or pervasive computing [Weiser 1991; Saha and Mukherjee 2003], in which every device is capable of doing computing with different intelligence, but the underlying technologies are calm, unobtrusive and/or invisible. The goal of ubiquitous or pervasive computing is to offer novel ubiquitous or pervasive services for users in right place, at right time and by right means with some kinds/levels of smart or intelligent behaviors [Ma et al. 2005]. Thus in ubiquitous or pervasive computing, users are enabled to mainly focus on their desired tasks by using appropriate computing services on their data, and have no need to care about any machine specific and management details. We believe that the future computing infrastructure will provide ubiquitous accesses to computing services globally through distributed resources. In this context, the computing systems and environments must be hassle-free for end users and much easier to maintain and manage for administrators.

There are many challenges to bring the blueprint of ubiquitous or pervasive computing to real world, for examples, effective use of smart spaces, invisibility, localized scalability and masking uneven conditioning [Satyanarayanan 1996] and finally achieving personalized or active services. In spite of these challenges and difficulties, ubiquitous or pervasive computing has stimulated many researches on embedded or wearable computers, universal electronic identification, sensors/actuators, etc., as well as associative ubiquitous network infrastructure and open services deployment and management frameworks. Lots of studies have been devoted to the ubiquitous or pervasive computing related technologies in system architecture, OS, middleware, interface model, design, deployment and management. It is highly expected, from the scalable service perspective, that a smart ubiquitous or pervasive computing platform should enable users to get different services via a single light-weight embedded device, and get the same service via different types of embedded devices. Unfortunately, all of the current technologies can not achieve such goal. In another word, users are often unable to select their desired services freely via available devices or platforms.

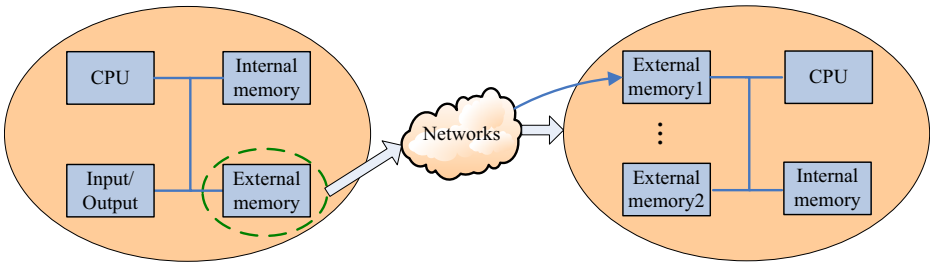

Fig. 1. An extended von Neumann architecture (I)

In order to address these two service challenges, we proposed a new computing paradigm, namely, transparent computing [Zhang and Zhou 2006]. 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, system/service programs are stored on the central servers (like warehouse), while fetched on demand in a stream way, and automatically initiated/executed on the light-weight embedded devices or clients with local CPU and memory resources (like

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 don't need to do or even care 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, in local area network environments mainly. This pilot system then can be further generalized to ubiquitous networks.

The rest of the paper is organized as follows. The next section introduces the basic motivation, the concept of transparent computing with its comparison with other existing similar 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 the 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 last section.

2. OVERVIEW OF TRANSPARENT COMPUTING

2.1 An Extended von Neumann Architecture

Transparent computing paradigm is aiming at the vision advocated by ubiquitous or pervasive computing, in which users can demand any computing services in a hassle free way in right place, at right time. The essential idea of transparent computing is to extend the traditional stored program concept in a single computer to networked/ubiquitous computers/devices spatio-temporally. 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 fetched and scheduled on demand in a stream way 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 check the traditional concept of stored program computer. As we all know, 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 [Aspray 1990].

The model that is based on a single local storage structure to hold both instructions and data (also known as programs) can achieve a great flexibility, for example, easily replacing and changing the programs. However, this traditional way of storage with local devices is hard to provide ubiquitous or personalized or active services to users. Several disadvantages of this approach are illustrated from different aspects below. Firstly, with the local storage only, the small, limited size embedded devices can not hold all the needed programs, while the relatively larger size devices can not share programs beyond their contents. Secondly, programs on the local storage need maintenance, fighting against malware (including virus, worms and spyware, etc.), and other management jobs, which are usually very hard for non-expert users. Thirdly, the mechanical character of local storage, such as hard disk, is often broken and becomes the sources of system failure. With the advent of networking, many approaches are proposed based on network technologies to solve the problem of data sharing and partially program sharing [Sandberg et al. 1985; Howard et al. 1988]. Using these approaches, however, it seems difficult or impossible to solve the pro-

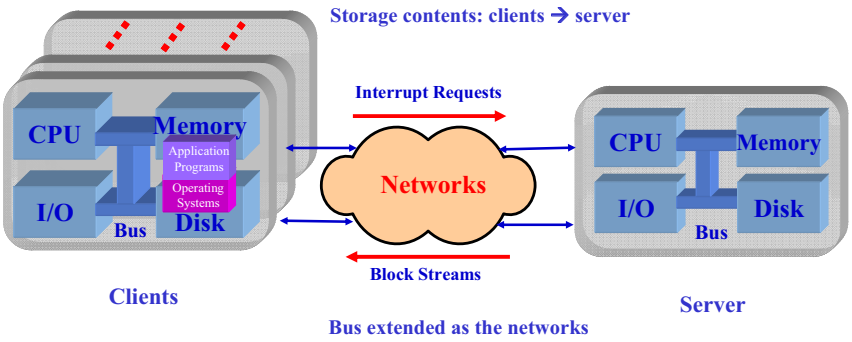

Fig. 2. An extended von Neumann architecture (II)

gram sharing problem fundamentally, thus to provide users with desired services freely. For example, it is hard to share commodity OS and applications (e.g. Windows) with a bare hardware platform and let users select other software services on Linux platform.

In order to have fundamental solutions to the above problems, it is necessary to extend the von Neumann architecture in networking/ubiquitous 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 (client) is responsible for computing only, and share the programs (including OSes and applications) stored on another computer (server) via the networks. The servers just act as a repository of all needed programs (including OSes and applications). The programs (including OSes and applications) will be streamed back the client in small streaming blocks on demand via the networks, and be executed with the local resources, such as memory and CPU. In such spatio-temporally extended architecture, the storage contents are extended from local computer (client) to another computer (server) via the networks, the interrupt requests for I/O and storage, and system bus have been extended from within local computer (client) to the networks, as well as the file and user managements.

Accordingly, on such spatio-temporally extended architecture, client/embedded devices can have less CPU power with limited size of memory which reduces the hardware cost significantly (easy to maintain); Client/embedded devices can run different programs (including OSes and applications) which reduces drastically 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 evidently (easy to use); the centralized management can reduce remarkably 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 a 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 concerning all the maintenance, upgrade and management, which will be carried out by the system itself

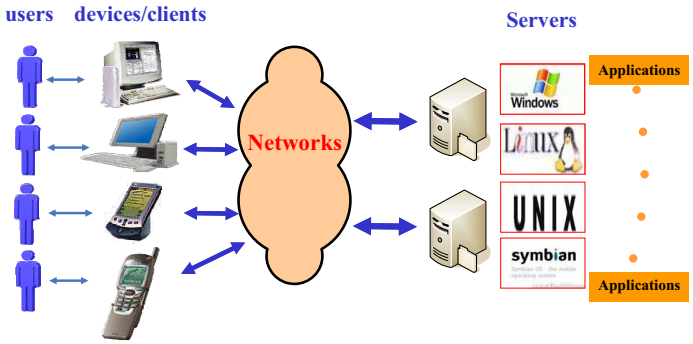

Fig. 3. Same/different services via different/same clients

or by administrators centrally on the servers. The detailed description is shown in Fig. 3.

2.2 Related Work and Difference

There has been extensive research in this area. Our proposed transparent commuting paradigm and the extended von Neumann architecture are most related to the previously proposed systems such as personal computers (PC), network computers (NC), thin-clients, diskless computers, virtual machines and clusters.

Because of the management challenge of personal computers, network computers, such as the Java Station by Sun [Herrtwich and Kappner 1997], have been proposed to replace the personal computers. Such solution supports very limited applications such as WWW & Java applications only, and does not work with general commodity OSes or other applications such as Microsoft Office suite.

Thin client computing systems, such as Microsoft RDP [Cumberland et al. 1999], Citrix ICA [Boca 1999], Sun Ray 1 [Sun 2004], VNC [Richardson et al. 1998], and MobiDesk [Baratto et al. 2004], have been very popular which provide a full featured desktop to users with low management costs. However in such computing paradigm, all computing tasks are performed at the central server (which is different with our proposed approach, namely storage and computing are separated), while a client works only as a user interface by performing display, and keyboard/mouse input/output functions, but 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 cannot be supported by thin-client systems efficiently.

Our idea is quite similar to the concept of diskless computers such as [Cheriton and Zwaenepoel 1983; Linlayson 1984; Croft and Gilmore 1985]) in early years. Without local hard disks, a diskless computer usually downloads an OS kernel image from the remote server. It thus cannot support OSes that do not have clear kernel images, such as Windows. Neither does it not support booting from heterogeneous OSes.

The concept of resource virtualization was introduced long time ago and recently has been more popular recently to address security, flexibility, and user mobility. One evident example is that VMware [VMware 2001] has extended the concept of virtual machines to support multiple commodity platforms. VM-based stateless thick client approaches, such as ISR (Internet Suspend/Resume [Kozuch and Satyanarayanan 2005]), use virtual ma-

Clients/devices	OS modes	Computing & Storage	OS free selection	Applications	Server requirement	Management	Security control
Transparent computers	no pre-installations, block streaming, multi-OSes	storage in server side, computing in client side	yes	all	normal PCs	centralized	yes
Personal computers	pre-installations, single OS	both in local computer	no			distributed	no
Network computers	pre-installations, single OS	both in server	no	very limited	high	centralized	no
Thin client computers	no pre-installations, single OS	both in server	no	limited for multimedia applications	high	centralized	no
Diskless computers	no pre-installations, single OS	storage in server side, computing in client side	no	no Window support, no heterogeneous OS booting	high	centralized	no
Virtual machines	pre-installations, multi-Oses, low performance	both in client side	yes	limited for multimedia applications		centralized	no
Clusters	pre-installations, single OS	both in client side	no	HPC applications		centralized	no

Fig. 4. Feature comparison with the exiting computing systems

chine technology (e.g., VMware) together with a network file system (e.g., Coda [Satyanarayanan 2002]) 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 virtual machines 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. Additionally, their approach still keeps the computing and storage in the same local clients. Due to the natural 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 ubiquitous or pervasive applications since most of the clients are embedded systems or devices.

In order to give a better view on these similar solutions, we summarize the comparisons with the above mentioned systems in Fig. 4.

3. CONCEPT AND ARCHITECTURE OF META OS (4VP+)

To realize transparent computing as stated above, we introduce a novel concept of Meta OS that is a super-OS control platform running beneath operating systems, but 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. 5. In order to implement the separation of execution and storage of programs, the Meta OS should have two basic functions. The one is to let users select which OS should be loaded and used, and then boot the needed OS remotely from server repositories. The other is to stream programs demanded by users during or after the OS startup. This is vital because of lacking local storage in the client. The concept of Meta OS can be embodied through a distributed platform with several parts resided on the client and several parts resided on the server, while other parts are integrated with the Instance OSes.

As discussed already 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 provide with OSes, application software and data for bare clients, which need these resources. The clients demand the needed programs (including OSes and applications) through the networks (Delivery Network) in a streaming way, and execute them (Just-In-Time computing) by using the

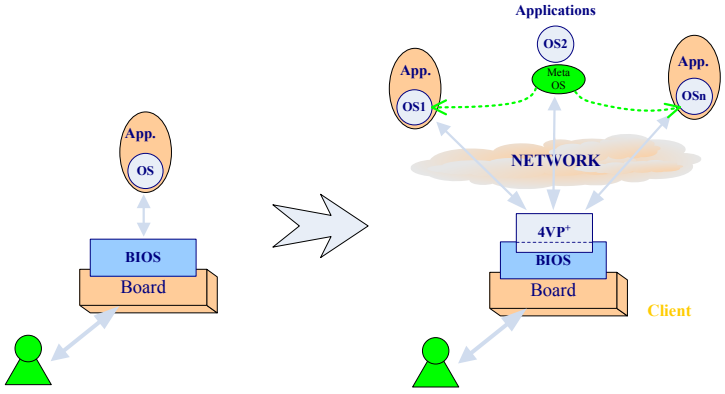

Fig. 5. General models of Meta OS and Single OS

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.

The layer architecture of a transparent computing system based on Meta OS is given in Fig. 6. The client must be with a resident module of MRBP (Multi-OS Remote Boot Protocol). 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 JITC (Just-In-Time Computing) 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 VDMS (Virtual Disk Management System), NSAP (Network Service Accessing Protocol), MRBP service, etc. The heterogeneous OS layers are used to store and support different types of OSes. The application layer means 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 the next section.

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. 7. This 4VP+ software platform is mostly installed in a management server, except 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 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 a NSAP (Network Service Access Protocol) 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

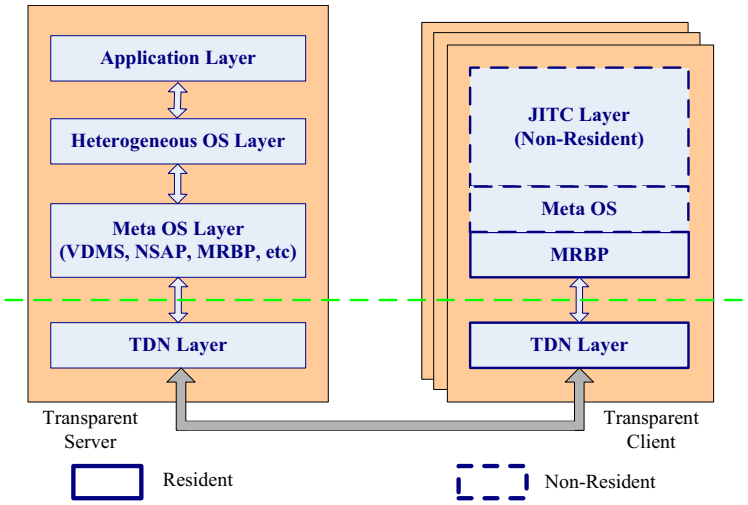

Fig. 6. Layered architecture of transparent computing

servers and then load them as if with 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 [Neuman and TSó 1994] with the V-user manager in the server. If it successes, 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 following 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. In order 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 ROM [Sun 2004], or a more flexible way by downloading the OS kernel directly from the server [Cheriton and Zwaenepoel 1983] that can not support OSes like Windows since it has not a clear kernel.

Our key idea is to first initialize a virtual I/O device for a client, whose requests will be sent through networks and executed by a server instead of a local hard disk. In such a way, any OS can be loaded according to its normal procedure as if a real hard disk existed. Fig. 8 lists the main steps involved in the remote boot process in the server and client, respectively.

After 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

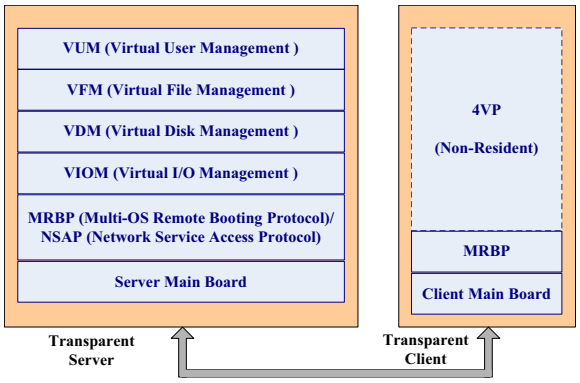

Fig. 7. The modules of 4VP+

the Virtual disks images on the server repositories through Virtual I/O devices. The client then executes the MBR (Master Boot Record) of the specific virtual disk, continuing the booting process as if there was a normal hard disk. The services, reside on server, such as MRBP and NSAP, will serve the booting process accordingly.

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

4.2 Network Service Access Protocol

The core idea of 4VP+ is the notion of 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 the remote server, and will be fetched to the client on demand. The access of virtual disk images on the server is through the Network Service Access Protocol (NSAP).

A virtual disk request issued by above OS or file system will be changed into one or more NSAP packets sent to the remote server and responded 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 send to the server.

The first function of NSAP is to establish a unique connection between V-disk image in server side and the V-disk in 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 server to clients or the computation results from clients to server. These transmissions occur when interruptions or I/O requisitions are made.

Fig. 9 shows the format of each remote disk request and reply in an application-level packet. Each packet starts with an identification field of 4 bytes to denote the unique packet sequence number. The 2 byte operation mode is used to distinguish between three different types of remote disk packets: read request, write request, and server acknowledgment. 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 a packet is a write request or a read response, then the actual disk data to be written or read will follow.

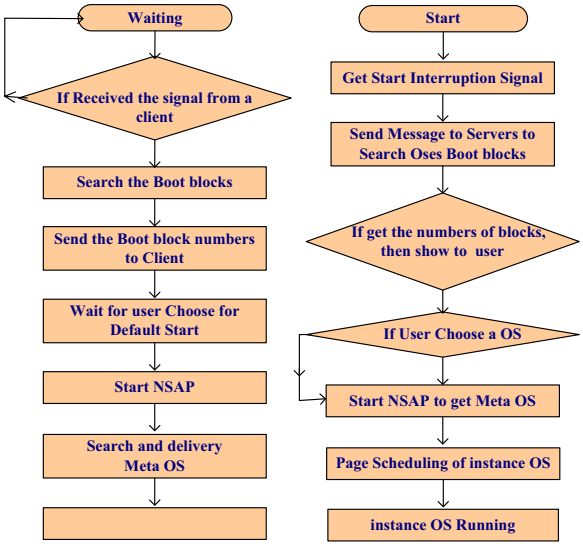

Fig. 8. Boot process of Meta OS and Instance OS

The NSAP uses UDP to deliver data packages. Thus, to maintain reliable data transmission, the NSAP server will send an explicit acknowledgment for each received remote disk request. 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 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 the most disk accesses are involved with a small amount of data. So 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 we have stated, 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 locating 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

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

Fig. 9. Format of NSAP request/response packets

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 virtualizations can be illustrated in Fig. 10. 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 below.

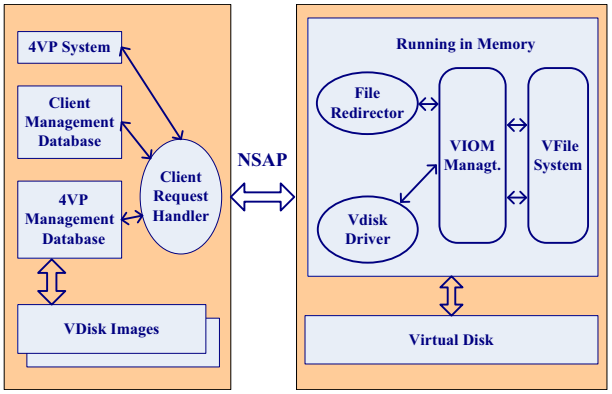

Fig. 10. Virtualizations in 4VP+

In order 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 cannot 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 copy-on-write (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. The copy-on-write semantics, however, 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 simplifes software management tasks, especially for system recovery. If a client is corrupted by accidental errors, software bugs, or malicious attacks such as viruses, worms, and 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 RedFlag 4.1 Desktop (Linux kernel 2.4.26-1) [RedFlag Linux]. Our implementation of MRBP is based on and compatible with the Intel PXE protocol [PXE 1999] for sending boot requests. Because device drivers are platform dependent, we 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 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 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 1GHz machines, each with 128 MB DDR 133 RAM and 100 Mps onboard network card. The server is an Intel Pentium IV 2.8GHz PC with 1G RAM, a 1Gbps network card, and two 80 GB 7200rpm soft RAID0 hard disks. The clients and the server are connected by an Ethernet switch with 48 100Mbps 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 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+, the 4-8 hour system maintenance took place every Thursday. No class can be arranged to use the classroom during this maintenance period. After deploying 4VP+, the classroom can be used everyday without weekly service interruption.

Improved security: After deploying 4VP+, there have been no reported viruses or worms in the system. One physical theft did happened 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 significant data loss. As a contrast, all the 4VP+ transparent clients remained intact, except for one that suffered loss of 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 our real deployment but with a more powerful server of AMD Athlon64 3000+ machine. The server is configured with 2 GB Dual DDR 400 RAM, two 80 GB Seagate Barracuda 7200rpm 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 80GB Seagate Barracuda 7200rpm). 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 [Iometer] to summit random disk access requests of different size to the client. We just show the results of random, unbuffered case on Windows in Fig. 11 and 12. The unbuffered 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 decreasing when the request size is larger than 32KB, which is the maximum size delivered. So, this decreasing may be due to that the network communication time dominates the latency, for that a large request size will result in several service requests. At the same time, because the response in 4VP+ system can be satisfied with the server's memory cache, the V-disk performance is higher than the local disk when the request size is small. The write throughput, as shown in Fig. 12 is smaller in V-disk than local disk due to the server cache can not satisfy the requests instantly. Note that in local disk scenario, the write throughput is bigger than read. This may be due to that the embedded hardware cache of hard disk can return the write result instantly without carrying the real disk operations. We also evaluated the performance in Redflag Linux environment. The results are shown in Fig. 13 and 14. For the least sector size in Linux ext3 file system is 4KB, Iometer can not test the 1K and 2K 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 measurements. In addition, we have tested the throughput in the buffer case, the result is also similar.

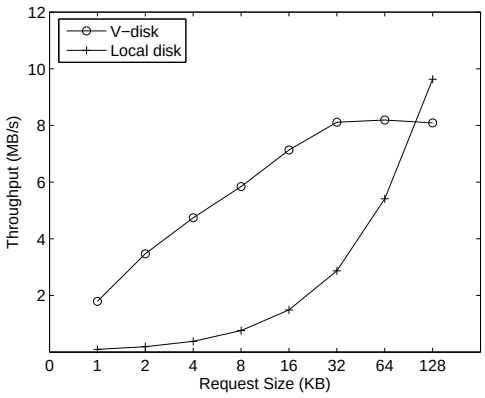

Fig. 11. Random read throughput in Windows

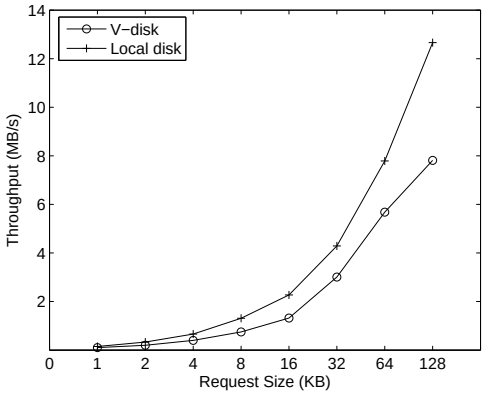

Fig. 12. Random write throughput in Windows

5.2.2 *Application Performance.* Table I and II lists 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.

For all four categories of performance shown in Table I, we observe that in Windows environment, 4VP$^+$ outperforms the regular PC in the single client scenario. The latency

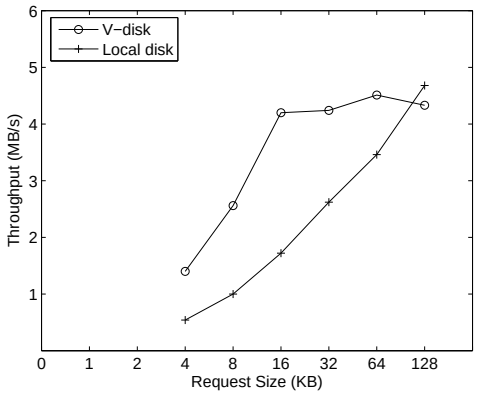

Fig. 13. Random read throughput in Linux

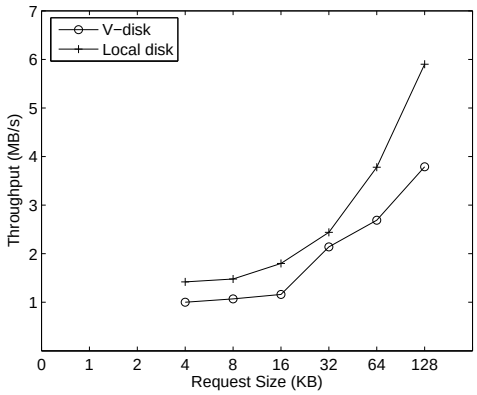

Fig. 14. Random write throughput in Linux

increases with the numbers 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 worst case scenario rarely occurs in our real deployment, where it has been found that there is strong locality of disk access patterns by studying the traces collected from the server. As we can see that in Table II 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[i-Bench 2001]. 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 [Boca 1999], Microsoft RDP [Cumberland et al. 1999]. The

Operation	PC	4VP+ clients		
		1 client	10 clients	20 client
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 1MB 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 I. 4VP+ client access latency in Windows (seconds)

Operation	PC	4VP+ clients		
		1 client	10 clients	20 client
Booting OS	85.67	58.20	97.72	127.28
EIOffice (Start up)	2.03	1.07	2.35	6.03
EIOffice (Open 1MB 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

Table II. 4VP+ client access latency in Linux (seconds)

results are shown in the Fig. 15. We observe that when the number of clients is smaller than four, ICA and RDP achieve lower client latency than 4VP+. However, as the latencies increase linearly with the number of clients in both ICA and RDP, 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 extends the von Neumann architecture spatio-temporally from the concept of "stored program" to the networking environments, to provide more active services for end users towards ubiquitous or pervasive computing. Based on this paradigm, we proposed 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 common and simpler. 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

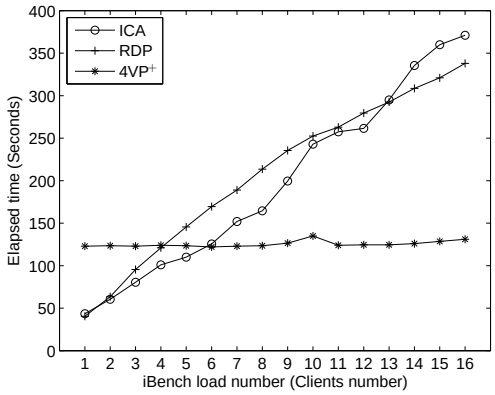

Fig. 15. Client observed i-Bench run latency

been widely deployed in China. We also gave 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 ubiquitous or pervasive computing infrastructure.

There are still many important aspects which are being investigated in our research studies. We have been extending our proposed approach to support a broader embedded devices (PDA, intelligent mobile phone, etc.) to connect with ubiquitous communication networks, and to support more instance embedded OSes and applications. Due to the space limitation, we will report further details in the near future.

ACKNOWLEDGMENTS

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. Special thanks to the people who have given their valuable comments and suggestions. We thank the people and organizations who have deployed the system. This paper is funded by the Key Technologies R & D Programs in China (No. 2005BA115A02).

REFERENCES

2004. Sun Ray Overview, White Paper, Version 2. `http://www.sun.com/sunray/whitepapers.html`.

ASPRAY, W. 1990. The Stored Program Concept. *IEEE Spectrum 27,* 9 (Sept), 51.

BARATTO, R. A., POTTER, S., SU, G., AND NIEH, J. 2004. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*.

BOCA, I. 1999. Citrix ICA Technology Brief, Technical White Paper.

CHERITON, D. R. AND ZWAENEPOEL, W. 1983. 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., 128–140.

CROFT, B. AND GILMORE, J. 1985. Bootstrap Protocol (BOOTP). RFC 951.

CUMBERLAND, B., CARIUS, G., AND MUIR, A. 1999. *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Microsoft Press.

HERRTWICH, R. G. AND KAPPNER, T. 1997. Network Computers – Ubiquitous Computing or Dumb Multimedia? In *Proceedings of the Third International Symposium on Autonomous Decentralized Systems*.

HOWARD, J. H., KAZAR, M. L., AND MENEES, S. G. 1988. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6(1)*, 51–81.

i-Bench 2001. i-Bench. http://www.veritest.com/benchmarks/i-bench/.

Iometer. Iometer. http://www.iometer.org.

KOZUCH, M. AND SATYANARAYANAN, M. 2005. Internet Suspend/Resume. In *Proceedings of the 4th IEEE Workshop Mobile Computing Systems and Applications*.

LINLAYSON, R. 1984. Bootstrap Loading Using TFTP. RFC 906.

MA, J., YANG, L. T., APDUHAN, B. O., HUANG, R., BAROLLI, L., AND TAKIZAWA, M. 2005. Towards a Smart World and Ubiquitous Intelligence: A Walkthrough from Smart Things to Smart Hyperspaces and UbicKids. *International Journal of Pervasive Computing and Communications 1,* 1, 53–68.

NEUMAN, B. C. AND TSÓ, T. 1994. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications 32(9)*, 33–38.

PXE 1999. Preboot Execution Environment (PXE) Specification. ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf.

RedFlag Linux. RedFlag Linux. http://www.redflag-linux.com/eindex.html.

RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. 1998. Virtual network computing. *IEEE Internet Computing 2,* 1, 33–38.

SAHA, D. AND MUKHERJEE, A. 2003. Pervasive Computing: A Paradigm for the 21st Century. *IEEE Computer 265*, 25–31.

SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. 1985. Design and Implementation of the Sun Network Filesystem. In *USENIX Association Conference Proceedings*.

SATYANARAYANAN, M. 1996. Fundamental Challenges in Mobile Computing. In *Proceeding of the Fifteenth ACM Symposium on Principles of Distributed Computing (PODC)*. Philadelphia, PA.

SATYANARAYANAN, M. 2002. The Evolution of Coda. *ACM Transactions on Computer Systems 20(2)*.

VMware 2001. VMware GSX server. http://www.vmware.com/products/gsx.

WEISER, M. 1991. The Computer for the Twenty-first Century. *Scientific American 265,* 3, 94–104.

ZHANG, Y. AND ZHOU, Y. 2006. Transparent Computing: A New Paradigm for Pervasive Computing. In *Proceeding of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC06)*. 1–11.

...