TransCom: A Virtual Disk based Pervasive Computing Platform for Heterogeneous Services

Yuezhi Zhou, Yaoxue Zhang, Guanfei Guo, Yinglian Xie, and Hui Zhang

Abstract— **This paper presents the design, implementation, and evaluation of *TransCom*, a virtual disk based pervasive computing platform that supports heterogeneous services of operating systems and their above applications in enterprise environments. In TransCom, clients store all data and software, including OS and application software, on virtual disks that correspond to disk images located on centralized servers, while computing tasks are carried out by the clients. Users can choose to boot any client for using the desired OS including Windows, and access software and data services from virtual disks as usual without considerations of any other tasks, such as installation, maintenance and management.**

By centralizing storage yet distributing computing tasks, TransCom can greatly reduce the system maintenance and management costs in potential. We have implemented a multi-platform TransCom prototype that supports both Windows and Linux services. Our extensive evaluation based on both testbed and real usage experiments has demonstrated that TransCom is a feasible, scalable, and efficient solution for successful real world use.

I. Introduction

In the last two decades of 20th century, with the rapid advances in hardware, software, and networks, 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, handhelds, wearables, etc., marks a departure from the established traditional general-purpose desktop computing to greatly heterogeneous and scalable pervasive or ubiquitous computing [27, 38], which aims at that users can only focus on their tasks without distractions and the underlying technologies are calm and invisible. Thus, we are moving away from a machine-centric view of computing to a more service-oriented perspective, i.e., users just simply accomplish their tasks by using computing services on their data, and do not care about the machine specifics 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 pervasive computing to real world, for examples, effective use of smart spaces, invisibility, localized scalability and masking uneven conditioning [30] and finally achieving personalized services. In spite of these challenges and difficulties, the pervasive computing vision of services has stimulated many researches on embedded or wearable computers, universal electronic identification, sensors/actuators, etc., as well as associative pervasive network infrastructure and open services deployment and management frameworks [21]. Lots of studies have been devoted to the pervasive computing related technologies in system architecture, OS, middleware, interface model, design, deployment and management. However, it is highly expected, from the scalable service perspective, that a smart pervasive computing platform should enable users to get different services via a single light-weight device and a same service via different types of devices quickly, through any types of networks including LAN, WAN and wireless. Unfortunately, all of the current technologies can not achieve these uneven conditioning services. In another word, users are often unable to select their desired services freely via the devices or platforms available to them.

In order to address these two above service challenges, in this paper, we present *TransCom*, a virtual disks-based pervasive computing architecture for supporting heterogeneous services in enterprise environments. Our key technique for addressing the above challenges is the use of *virtual disks*. Each virtual disk simulates a physical block-based storage device using a disk image located on the centralized server and is accessed via network communication. Since disk operations locate beneath file systems and applications, TransCom can support heterogeneous OSes, including Windows (it does not have a single kernel file and its source code is not available for modification), and their applications.

With all computing tasks still carried out by the clients and not virtualizing local CPU and memory resources, TransCom retains the high performance, yet significantly reduces the maintenance and management overhead by completely eliminating the end user management tasks and reducing the inconsistent or faulty software states on local disks. Users can access their desired services, including OSes and their above applications with any client available to them. Specially, TransCom has the following desirable features:

- *Heterogeneous service support:* TransCom supports heterogeneous services with no or minimum OS and no application modifications. TransCom clients can flexibly choose to boot the desired operating systems via the same remote OS boot process.
- *Flexible software and data sharing:* Our design enables both data and application software sharing. The system and application file sharing is transparent to users with a novel file redirector mechanism.

Manuscript received April 22, 2008; revised July 13, 2008.

Y. Zhou, Y. Zhang and G. Guo are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China.

Y. Xie is with Microsoft Research Silicon Valley.

H. Zhang is with Carnegie Mellon University.

- *User transparency:* The use of virtual disks is transparent to users and applications, and requires no application modification. From the perspective of applications and users, there is no difference between accessing data from virtual disks and accessing data from local hard disks.

We have implemented a multi-platform prototype system for running both Windows and Linux at TransCom clients. The Windows-based system has already been deployed at universities e-learning classrooms for daily use. Our extensive evaluation, consisting of both testbed experiments and real usage experiments, has shown that by using a powerful server, TransCom clients can achieve comparable or even better disk and application performance than regular PCs with local hard disks. By centralizing storage yet distributing computing, TransCom is more responsive and scalable than the existing thin-client systems [3, 10, 26]. By only virtualizing disks, TransCom can achieve higher performance than virtual machine based pervasive computing approaches (e.g., stateless thick clients [32, 35]) which virtualizing all hardware resources.

The remainder of the paper is organized as follows. Section II overviews the TransCom system architecture. Section III, IV, V describe the system in detail, including the remote OS boot, virtual disk access and sharing, and implementations. In Section VI, we present extensive testbed and real usage experiment evaluation of TransCom. Section VII discusses possible extensions and optimizations. Section VIII discusses the related work before we conclude.

II. System Overview

TransCom adopts the conventional client and server architecture, where a single server supports up to tens of client hosts connected in a network system. A client machine can be any regular PC without local storage devices. The server can be either a regular PC or a higher-end dedicated machine. In our current design, TransCom server uses MAC address to identify a unique client. The entire system should reside in a local area network, protected from other networks by NATs (Network Address Translator) or firewalls for security.

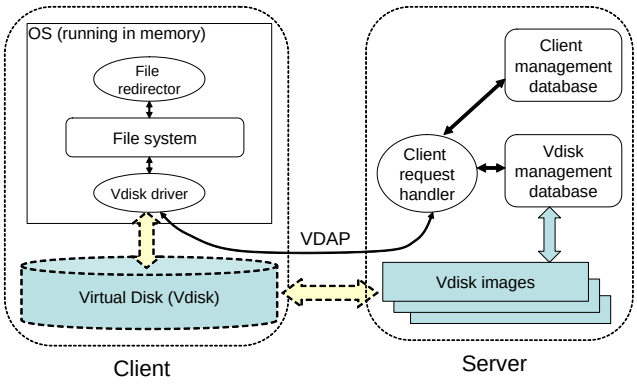


Fig. 1. Overall architecture of a TransCom system with a server and a single client

Figure 1 shows the overall architecture of a TransCom system with a server and a single client. Without local hard disks, each client accesses software (including OSes) and data from one or more *virtual disks* (Vdisks) that simulate physical block-based storage devices. A Vdisk, in essence, is one or more disk image files located on the server and accessed by the client remotely via a Vdisk access protocol (VDAP). Access to Vdisks can be generally supported across different operating systems with a Vdisk driver — a specialized device driver running on the TransCom client. Disk buffer requests and page faults are handled in a regularly way through Vdisk driver as if there existed a local hard disk.

TransCom client boots from the server remotely to load the desired OS, including the Vdisk driver. TransCom clients issue separate requests for OS boot and disk access. When a client is powered up, it makes a boot request to the TransCom server, and uses a remote boot protocol to first download and enable Vdisk access functions. The client then issues Vdisk access requests to launch the desired OS from its own Vdisks. During this OS loading process, a Vdisk driver will be loaded in replace of the traditional hard disk driver. Further disk requests will go through the Vdisk driver, and the control of the hardware will be handed to the OS for the boot process to finish regularly, as if with a local hard disk (Section III).

By using virtual disks, TransCom enables not only data sharing, but also software sharing for operating systems and popular applications. Specially, multiple user-perceived Vdisks on different clients can map to the same Vdisk image on the TransCom server. In order to avoid the conflict of multiple users' written and then read operations on the files of the same shared Vdisk, each client uses a file system agent called *file redirector* to redirect the access of written files on user-perceived Vdisks to the access of different shadow files on user-specific server-perceived Vdisks (Section IV-B). Eventually, every file access request will be converted into one or more disk block access requests, handled by the corresponding Vdisk driver by communicating with the server (Section IV-A).

The TransCom server is running as an application daemon. It maintains a client management database, a disk management database, and a list of all Vdisk image files belonging to all clients in the system. The client management database is configured with a list of mappings between clients' IP addresses and their MAC addresses. The Vdisk management database maintains a mapping between clients' IP addresses and the corresponding Vdisk image files. Given a disk access request, TransCom server first looks up the Vdisk management database to find the corresponding Vdisk images. It then performs the requested operations before sending replies back to the client (Section V-B).

III. OS Independent Remote Boot

Figure 2 lists the steps involved in the TransCom remote boot process. As a first step in OS boot, each TransCom client submits an OS boot request to the server and obtains an IP address for subsequent communication. This step can be supported by hard coding a remote boot protocol into the client's BIOS beforehand. In our current implementation, we adopt the existing Intel PXE protocol [24] and tools.

TransCom server maintains a client MAC address to IP address mapping table in the client management database. Given the boot request, the server assigns the corresponding IP address to the client via DHCP [11] based on the client MAC address.

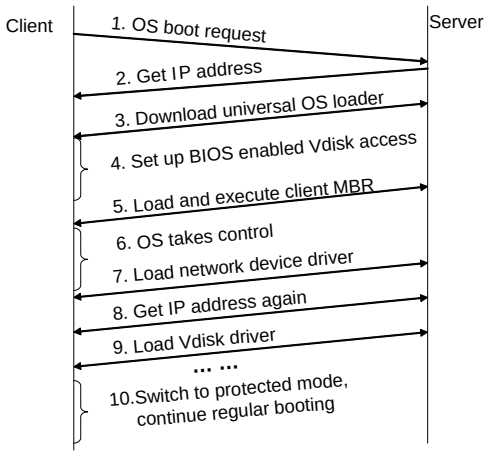


Fig. 2. The remote OS boot process of a TransCom client

The client then downloads from the server a universal OS loader using TFTP [33], also supported by PXE (step 3). The purpose of the universal OS loader is to replace the BIOS hard disk access function (e.g., INT 13 in X86 architecture) with a customized one that redirects all disk access requests to the server using a Vdisk access protocol. Once the universal OS loader is up and running, the client will have the ability to access Vdisks remotely from the server (step 4). Thus the immediate next step is to read and execute the Master Block Record in the client's Vdisk, which is usually the first step in the regular OS boot process (step 5). After this point, OS takes control to resume initializing various modules including device drivers and file systems (step 6).

As this regular OS boot process progresses, however, the BIOS enabled Vdisk access will no longer work due to its real mode memory management. Most modern OSes today access memory in protected mode. Once in control, they will bypass the BIOS real mode disk access functions, and replace them with disk device drivers for better performance. So they will not be able to access those memory segments that store the BIOS functions in real mode, including the specialized Vdisk access instructions installed by the universal OS loader.

To solve this problem, we replace the normal disk drivers with a Vdisk driver in the boot process, so that the client machine can continue accessing Vdisks after the OS switches to the protected mode of memory access. Note the disk drivers are typically loaded before network device drives in the normal OS booting process. Since the Vdisk driver relies on the network functions, we switch the device driver loading order so that the network devices are launched first for establishing connections to the server (step 7,9).

Another important detail is to enable DHCP client before Vdisk driver is activated. This is because after the OS switches to the protected mode, the IP address obtained under the real mode will no longer be accessible. So after loading network drivers, TransCom client needs to contact the server again using DHCP to re-obtain its IP address (step 8). By the time when the real mode access fails, all the required modules for accessing Vdisks, including the network interfaces and the Vdisk driver, will have been loaded into the memory. Once the OS switches to the protected mode, all disk access goes through the Vdisk driver, as if there existed a physical local disk (step 10). The OS can resume the rest of boot procedures until all the other required modules are up and running normally.

IV. Accessing Virtual Disks

The core concept of TransCom is the notion of virtual disks. From a user or application's perspective, there is no difference between accessing data from a Vdisk and a local hard disk. The actual contents in Vdisks reside at the remote server, and will be fetched to the client on demand. In this section, we describe the detailed process of accessing data from Vdisks as well as how data are organized and shared across different clients.

A. Vdisk Access Protocol

Vdisks are flat addressable block-based virtual storage devices locating beneath the file systems. Each block has 512 bytes. A TransCom client can be configured to access data from one or more Vdisks, with each corresponding to a Vdisk image located on the server. A Vdisk image consists of one or more files, depending on the required disk size. To manage different clients' Vdisks, the TransCom server sets up a *Vdisk management database* to maintain the mappings between a client's IP address and the corresponding Vdisk images. Administrators can configure the quotas of client Vdisks using a TransCom application tool that updates the server's Vdisk management database.

Every Vdisk data access request goes though a specialized Vdisk driver on the client. There are two types of disk requests involved in accessing Vdisk data:

- *A virtual disk request* is a disk access request issued by the file system to the Vdisk driver. It is of the same format as a disk access request to regular SCSI or IDE disk devices. Each request consists of the disk block number, the start offset, and the length of data to be read or written.
- *A remote disk request* is a disk access request issued by the Vdisk driver to the remote TransCom server through network communication. Given each virtual disk request received from the file system, the Vdisk driver will compose one or more remote disk requests to send to the server.

Each Vdisk driver maintains two request queues, one for the virtual disk requests received from the file system, the other for the remote disk requests to be sent to the server. For simple client fail over, the network communication between the Vdisk driver and the server uses an UDP-based Vdisk access protocol. Thus TransCom server maintains no client states or active connections, and processes requests individually as they arrive. For reliable data transmission, TransCom 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.

Vdisk drivers use timeouts to detect lost remote disk requests or replies for request retransmission. Each remote disk request has an unique ID, so that both the client and the server can detect and drop duplicate packets. For efficient performance, we would like a small timeout value in case of network loss. Our empirical experience has shown that most disk access involves a small amount of data (see Section VI-G.2). So we set the maximum size of data to be read or written by a remote disk request to 32 KB.

To ensure no out of order requests, each Vdisk driver 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 Vdisk driver 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.

Figure 3 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 Vdisk. If the packet is a write request or a read response, then the actual disk data to be written or read will follow.

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

Fig. 3. Format of VDAP request/response packets

Given a client remote disk request, the server first looks up the corresponding Vdisk image based on the client IP address. If the client is allowed to perform the requested operation (See Section IV-B), the server retrieves or updates the requested disk image, and sends an response to the client, acknowledging the completion of the process. If the client request is a read request, the server will also attach requested disk data in the reply.

B. Disk Organization and Sharing

The disk-level remote data access provides a wide spectrum of design space to share data across different clients at the server. At one extreme end, we can perform coarse-grained sharing based on an entire disk. This solution is simple but not efficient in disk utilization. At the other extreme end, we can perform fine-grained sharing based on each disk block, which could be efficient in disk utilization, but may result in high management overhead due to information booking and access control.

TransCom shares data at the granularity of files, since they are the most natural units for sharing and access control in convention. However, since TransCom server handles disk-level access requests, it is difficult to directly support file-level sharing. For maximizing the disk utilization, we would like TransCom to support the sharing of *system files* including OS and application software that would be used by multiple clients, while isolating *user files* that involve user private data. TransCom uses different mechanisms to share system files and user files:

1) Sharing User Files: TransCom uses existing remote file system solutions such as AFS [16], NFS [28], or CIFS [20] to share user files. This solution gives users flexibility in choosing different remote file systems based on the desired degrees of sharing. It is also simple, as access control and user authentication will be performed at the client file systems, reducing the server management overhead.

2) Sharing System Files: However, it would be difficult to use the existing remote file system solutions to share the above defined system files, mostly because certain system files are also user-specific, containing customizable configuration entries, yet having fixed path names for OSes or applications to work correctly. However, the client users need to write or modify these system files resulting conflicts. For example, the World Soccer Winning 11 game must save the user-specific game progress information in a fixed subdirectory. In order to share such software with NFS like solutions, users will need to change the directory locations. This potentially requires OS or application software modification and is often impossible in practice.

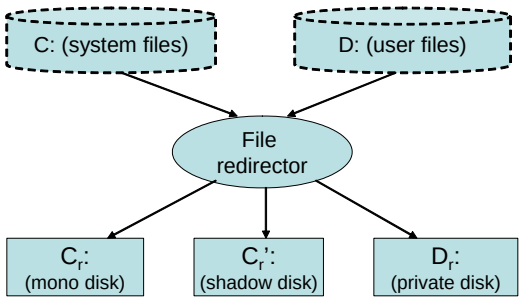


Fig. 4. Vdisk mappings. C and D are user-perceived Vdisks for storing system files and user files respectively. C_r, C'_r and D_r are server-perceived Vdisks for storing shared system files, customized system files, and user files, respectively. The file redirector is a file system agent for redirecting user-perceived Vdisk requests to server-perceived Vdisk requests.

TransCom uses a file system level agent called *file redirector* to map *user-perceived* Vdisks into *server-perceived* Vdisks for sharing system files. The set of system files are specified by the system administrator, with prior knowledge of potential TransCom usage scenarios. Figure 4 illustrates this idea. From a user's perspective, each client is configured with two types of Vdisks, one for storing shared system files, the other for user files. From the server's perspective, each client has three types of Vdisks. The existence of these three types of Vdisks is transparent to the users.

- *Mono disk:* It is used to store operating system and application files that are shared across all clients in

a TransCom system. Different clients' mono disks are mapped to a single Vdisk image at the server. Thus mono disk is for read only.

- *Shadow disk:* Each client has a shadow disk that is used to store the customized or written system files. The shadow disk maps to a client or user specific Vdisk image at the server. The corresponding data are private and will not be shared by other client hosts. Therefore, the shadow disk is in essence a copy-on-write (COW) disk for isolating written user-specific configuration files and system files.
- *Private disk:* Each client has one or more private disks that are used to store private user data. Similar to shadow disk, each client's private disks map to client-specific Vdisk images at the server, and will not be shared.

The file redirector translates the file access requests on user-perceived Vdisks into those on server-perceived Vdisks by intercepting all file system calls. If the file to be accessed locates on the private disk (user-perceived), the redirector simply maps the request to the same file on the server-perceived private disk. If the file to be accessed is on the system disk, the file redirector will redirect the request to the shadow disk in the following two cases: (1) a read request to a system file that already has a customized copy on the shadow disk, and (2) a write request to a system file (in this case, a copy of the file will be created first on the Shadow disk before written). Otherwise, the file redirector will redirect the request to the mono disk. The file redirector therefore supports *dynamic redirection* of system files for enabling file system level copy-on-write semantics. This software agent will be loaded from the Mono Vdisk as part of the underlying file system. We defer the discussion of its implementation in Section V. Figure 13 shows the pseudo code for the file redirector to handle an `open` file system call.

V. TransCom Implementation

We have developed a multi-platform (Windows and Linux) prototype of TransCom based on Intel X86 architecture. In this section, we present the implementation details including the server architecture.

A. Client Implementation

We choose Windows 2000 Professional as the Windows client OS. For Linux client OS, we adopt RedFlag 4.1 Desktop [25] (Linux kernel 2.4.26-1). There are three major implementation modules for supporting TransCom clients under different operating systems: (1) universal OS loader, (2) Vdisk driver, and (3) file redirector.

1) Universal OS Loader: Our current implementation uses the Intel PXE as the remote boot protocol for sending boot requests. The PXE tools are burned into the client BIOS beforehand. With X86 architecture, the universal OS loader will replace the INT 13 interrupt code to enable real-mode Vdisk access. It is implemented as 15,000 lines of ASM code. The implementation of the universal OS loader is independent of the client OS to be loaded.

2) Vdisk Driver: Because device drivers are platform dependent, we implemented two different Vdisk drivers, customized for Windows and Linux, respectively. The implementations are in C++. The loading procedure of Vdisk driver is also non-trivial and platform dependent. This is because TransCom client must load the network device drivers before loading the Vdisk driver in the remote OS boot procedure as discussed in Section III.

Since Windows 2000 is a modified microkernel, we simply modify the corresponding Windows Registry files to ensure the network device driver gets loaded before the Vdisk driver. There is no need to change or recompile the kernel. However, Linux is a monolithic kernel, so the Vdisk driver has to be compiled into the kernel. We also change the initialization sequence of device drives by modifying the related kernel source code before recompilation.

Note we also need to enable the client DHCP service for re-obtaining the client IP address before loading the Vdisk driver. While Linux kernel already provides DHCP facility, the DHCP client provided by Windows 2000 operates in user mode, and will not be launched until the kernel boots successfully. Therefore, for Windows, we also implemented a DHCP client driver in kernel mode to enable IP configuration before the kernel initializes TCP/IP.

The current implemented Vdisk driver uses a timeout value of 100 ms for sending remote disk requests to the server. It does not support device driver level cache or read ahead functions. Such optimization can potentially increase performance, and is a topic of ongoing work.

3) File Redirector: The file redirector is a file system agent intercepting all file system calls. For Windows 2000, it is implemented as a file system driver sitting on top of the device driver chain for handling file system operations. It intercepts the I/O Request Packet (IRP) from the built-in Windows I/O manager to perform the redirection as described in Section IV-B. For Linux, we modify the `open()` file system call provided by the Virtual File System of Linux kernel, by maintaining a dirty table that records the mappings between the modified system files and their corresponding shadow disk versions.

B. Server Implementation

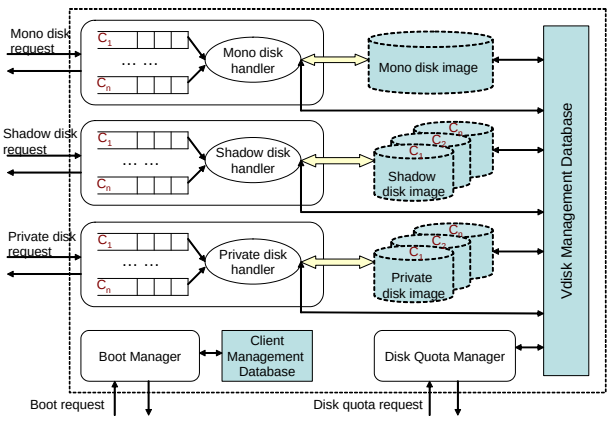


Fig. 5. TransCom server architecture

TransCom server is implemented as a multi-process program, with each process listening on a dedicated port, handling different types of client requests (Figure 5).

Specifically, there are three *disk request handler* processes handling the requests for three different types of disks. Each process maintains a request queue per TransCom client, and serves queues in a round robin fashion. Conceptually, there is only one Vdisk management database at the TransCom server. In implementation, however, each disk request handler deals with a dedicated Vdisk management database implemented as a file. As discussed in Section IV-B, there exists only one mono Vdisk image for all TransCom clients. For shadow disks and private disks, TransCom server maintains one Vdisk image per client disk. The default mono disk size is 8 GB, and the default shadow disk and private disk sizes are set to 1 GB per disk.

The *boot manager* handles boot request with a DHCP service and a TFTP service for clients to obtain IP addresses and to download the universal OS loader, respectively. The *disk quota manager* is responsible for processing disk quota related request. System administrators can login from a pre-configured special client to submit disk quota requests to the server and change the number of user disks. After that, a client must reboot to reflect the changes.

The server is implemented in C++ for both Windows and Linux. The Windows server OS is Windows 2003 Server (SP1) edition, and the Linux server OS is RedFlag DC Server 5.0 (Linux kernel 2.4.21-32).[1] Server side caching has not been implemented, and is ongoing work.

VI. Performance Evaluation

In this section, we evaluate TransCom performance using both testbed experiments and real usage experiments. We focus on the TransCom Windows prototype, which has already been deployed for us to collect the real-usage data.

We are primarily interested in the following questions: (1) What is the Vdisk access performance? And how does it compare against the performance of accessing local disks and virtual machine disks? (2) What is the TransCom client OS boot latency? (3) How does Vdisk access impact file system and application performance? (4) What is the scalability of TransCom, and how does it compare against other pervasive solutions? (5) What is the TransCom performance in real world usage?

A. Experiment Setup

We use five sets of experiments to answer the above questions. In all our testbed experiments, TransCom clients are configured as Intel Celeron 1GHz machines, each with 128 MB DDR 133 RAM and an 100 Mbps onboard network card. The server is configured as an AMD Athlon64 3000+ machine, with 2 GB Dual DDR 400 RAM, two 80 GB Seagate Barracuda 7200rpm soft RAID0 hard disks, and a 1 Gbps onboard network card. The clients and the server are connected by an Ethernet switch with 48 100 Mbps interfaces (used for clients) and two 1 Gbps interfaces (one used to connect the server). We also compare the TransCom client performance with a regular PC, which has the same client hardware configuration but with an additional local hard disk (80GB Seagate Barracuda 7200rpm), and a virtual machine that emulates the stateless-thick-client-like approaches, which is virtualized as with 128M memory and 8GB (dynamically expanding) SCSI hard disk using VMWare Workstation 5.0 hosted by Windows 2000 professional (SP4) with FAT32 file system on the the same regular PC hardware (but 512M physical memory). The server OS is Windows 2003 Server (SP1) edition running a NTFS 5.2 file system and Redflag DC Server 5.0 running EXT3 file system. The TransCom clients, the regular PC and stateless thick client all use Windows 2000 professional (SP4) with NTFS 5.0 file system. Among the 128 MB client (PC) RAM, about 90 MB is used for the working set of Windows, 8 MB is used as the video frame buffer, and the rest 30 MB is free after OS boot.

[1] In fact, these two server can be merged to one. Here we separate them just for consideration of user custom in real deployment. The single one server version is also available in the lab.

B. Vdisk Performance

We evaluate the Vdisk access performance in terms of latency and throughput in a single client TransCom system.

1) Vdisk Access Latency: We first examine the latency spent on various steps of accessing Vdisk data. We disabled the client file system cache, and used the Microsoft I/O performance evaluation tool SQLIO [7] in Windows (with the testing file size being 512 MB) and Iometer [19] in Linux environment respectively to submit random disk access requests of different sizes to the client. Because the request data size is small, each disk access request triggered only one remote disk request in our experiments.

Table I shows the average measured latencies of 20 disk requests. The standard deviations are small (less than 10%) and omitted. The "total" latency corresponds to the time elapsed between the Vdisk driver receives a disk access request from the file system and returning the results back to it. The first three steps are processed by the client Vdisk driver. The "Queuing" step measures the queuing delay before Vdisk driver starts processing the request. The "Request pack" step corresponds to the time Vdisk driver spends parsing the request and packing it into remote disk requests for sending to the server. The "Reply parse" step corresponds to the latency for the Vdisk driver to parse the server reply and return the results back to the file system.

The next three steps are processed by the server, including parsing received requests and looking up the corresponding Vdisk image ("Server parse"), performing the requested disk access operation ("File system I/O"), and packing replies for sending to the client ("Reply pack"). Finally, the "network" step is the time for sending both the request and reply over the network.

We observe that for Vdisk read request, the majority time is spent on network transmission. Server file system I/O accounts for only a small fraction of latency. This is mostly due to the server side memory cache, which we will further explore in the next section. For write request, in addition to the network communication, the server file I/O is also a bottleneck, in particular

Operation	Vdisk Read (Windows, Linux)		Vdisk Write (Windows, Linux)	
	4 KB	8 KB	4 KB	8 KB
Queuing	6.44, 1.02	6.38, 1.03	6.35, 1.02	6.51, 1.13
Request pack	2.58, 1.02	2.39, 1.06	17.04, 1.27	32.57, 1.29
Reply parse	9.92, 4.51	15.69, 4.82	5.53, 2.15	5.74, 2.85
Server parse	5.25, 1.06	5.04, 1.08	7.43, 9.82	8.57, 11.84
File system I/O	18.53, 372.67	19.99, 432.05	541.49, 531.53	2407.2, 642.66
Reply pack	1.60, 1.04	1.56, 1.06	1.62, 1.03	1.66, 1.06
Network	676.45, 657.54	1050.85, 1071.92	571.04, 669.23	1076.05, 1167.80
Total	720.77, 1038.86	1101.9, 1513.02	1150.5, 1216.05	3538.3, 1828.63
Local disk	7407.41, 8443.80	7462.69, 8557.61	5208.33, 7928.82	5347.60, 8264.10
VM-based disk	8264.46, 9097.50	8474.57, 9317.59	5464.48, 8759.97	5617.97, 8887.60

TABLE I

AVERAGE TIME SPENT AT VARIOUS STEPS IN PROCESSING A DISK READ OR WRITE REQUEST OF TWO DIFFERENT SIZES(μS).

when we increase the amount of data to be written each time. The total latency is on the order of millisecond, acceptable to most users and applications today. Because the bottleneck of read access is the network communication, by increasing the network speed (e.g., using 1Gbps network card) and using TCP/IP off-load engine (TOE) technology, we expect the performance to improve. To reduce the disk write latency, we can additionally optimize server Vdisk write operation by increasing the I/O speed or implementing application-level write optimization schemes such as lazy write.

We also compare the total latency with that of local disk and VM-based disks in stateless-thick-client-like systems. We can see that the latency of VM disk accessing is a little more than the local disk, showing a virtualized overhead of around 10 percent in read operation and 5 percent in write respectively. However, these two latency is much more than TransCom. This achievement is due to that in TransCom, the virtualization of disk is not only implemented inside the operating system, but also can leverage the high performance network and server capacity.

2) Vdisk Throughput: We evaluate the Vdisk throughput, also using SQLIO for Windows and Iometer for Linux. Previous studies have shown that most file access involves random disk access with small request sizes [37]. We thus focus on the random disk read/write, and compare the performance with the local hard disk throughput measured from the regular PC (LD) and that of the VM-based disk in stateless-thick-client-like systems (See Section VI-A).

Figure 6(a)-(f) show the Vdisk read and write throughput in various scenarios in Windows. For read access, Vdisk throughput increases with the request size, but saturates when the request size is larger than 32 KB, which is the maximum size handled by a remote disk request. This is because network communication time dominates read latency, and a large request size will result in multiple remote disk reads.

When the request sizes are small ($\leq$ 64KB), the Vdisk read throughput achieved using the 2 GB memory server is higher than local disk throughput. We varied the server memory size, and observed that decreasing server memory size reduced the performance significantly. When the server is configured with the same 128 MB memory as the regular PC, the Vdisk read throughput is lower than the hard disk throughput. As further investigation, we disabled the server file system cache in the 2 GB memory case, and observed a significant performance drop. Thus a large server memory cache is the key factor for explaining the high Vdisk read throughput.

For write requests, the throughput increases monotonically with larger request sizes, as server file system I/O performance is the dominant factor for Vdisk write. Server memory cache does not play a significant role. Interestingly, we find that for local disk access, write throughput is higher than read throughput. One reason could be lazy write optimization implemented by the OS and the hard disk, in which case data are written to the cache before return. This same reason may also explain why a larger server memory increased the Vdisk write throughput.

We also compare the disk throughput with and without client file system and device driver buffers. When these buffers were enabled, the performance increased slightly as expected. There is no significant difference between accessing RAID0 and IDE disks at the server with the former is little more.

We compare the throughput with that of local disk and VM-based disk (VMD) in Figure 6(e)-(f), we can see that the read and write performance of TransCom is better.

We also measure and compare the disk throughput with local disk in Linux environment. As shown in Figure 6(g)-(h), the trend is very similar to that in Widows only with a relative low value. This may due to the difference in operating system and evaluation tools.

C. Remote Boot Performance

In this section, we evaluate the remote OS boot latency, and compare it against a regular PC boot latency. To timestamp the boot process of a client, a dedicated monitor machine was linked to the client with a hub that connected to the server via the Ethernet switch. The monitor machine collected all observed network packets to and from the client using the Etherpeek tool [39]. For each experiment, we repeated the above process to measure the boot latency of every client in the system, and computed the average latency across all clients. Figure 7 shows the time spent on various steps by varying the number of clients booting simultaneously in the system.

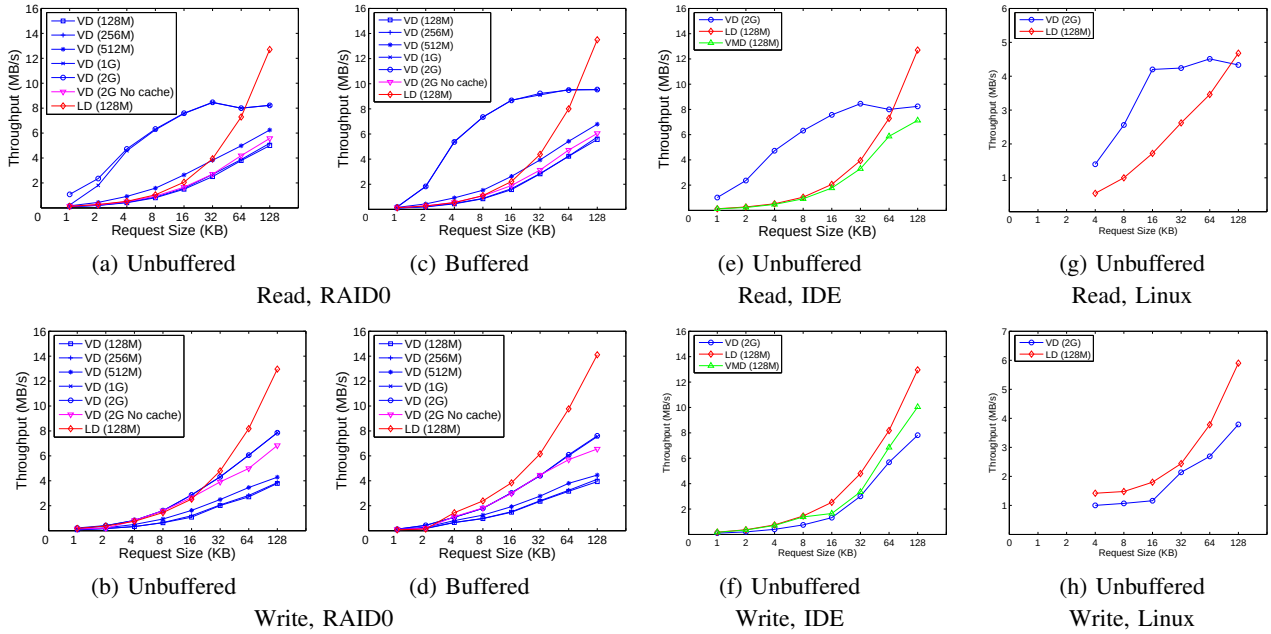


Fig. 6. Random Vdisk throughput. "Buffered" means the client file system and device driver buffers were enabled. "Cache" means the server file system cache was enabled.

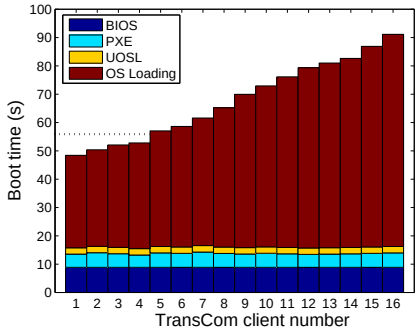


Fig. 7. Average time spent at various steps for a TransCom client to boot remotely from the server. We vary the number of simultaneously booting clients and report the average latency observed among all clients.

Phase	TransCom	Local disk	CIFS	VM-based disk
mkdir	0.93	1.03	2.93	1.69
cp	27.58	58.36	81.96	49.01
scan dir	92.41	89.88	156.43	283.61
cat	175.56	214.64	296.78	550.13
make	330.82	319.89	535.37	628.10
Total	627.30	683.79	1073.47	1512.54

TABLE II
AVERAGE TIME SPENT AT VARIOUS PHASES OF THE APACHE WINDOWS SOURCE TREE BENCHMARK (SECOND)

The "BIOS" step is the time spent by the client launching the hardcoded BIOS program. "PXE" is the time spent obtaining an IP address and downloading the universal OS loader. "USOL" is the time for initializing the universal OS loader to read the Vdisk MBR. These three steps take constant amount of time regardless of the number of clients, and account for a small fraction of the total boot latency. We observe that the majority boot time is spent in loading OS remotely from the server ("OS loading"). This portion of time increases when the number of clients is increasing.

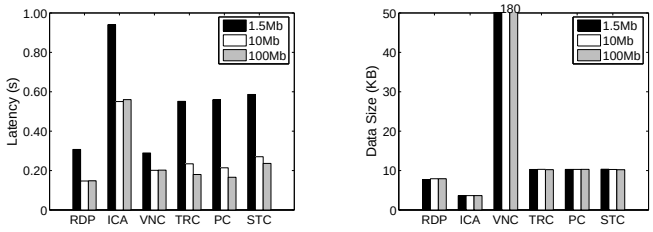
Overall, with a small number of clients (fewer than five), the total boot latency of a TransCom client is even smaller than a regular PC (The dot-line marked value is the total latency). This is due to the higher Vdisk throughput than the local disk throughput in our experiments. The total latency increases roughly in linear to the number of clients within our range, and is on the order of tens of seconds.

D. File System Performance

In this section, we evaluate the overall file system performance of a TransCom client, using a modified Andrew benchmark [8, 16]. We compared the performance against the file system performance of the regular PC with a local disk and the Common Internet File System (CIFS) [20] and that of the VM-based disk in the stateless-thick-client-like systems. For CIFS, we used the same TransCom client and server hardware configuration. In our benchmark, we used the Windows Apache 2.0.53 source tree. This source tree has 39.3 MB data before compilation, and 42 MB data after compilation. Table II shows the average performance over five runs. For each run, we rebooted both the client and the server to clean various caches.

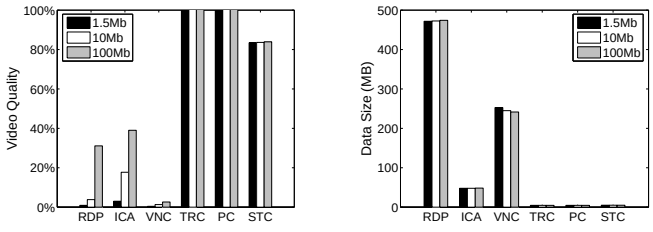
We observe that TransCom achieves better performance than both the regular PC and CIFS, except the "scan dir" phase, which requires accessing a large number of directories and files. With Vdisks, this phase will result in a large number of remote disk requests, and thus incur larger network communication overhead.

Our file system performance evaluation shows that, by using



(a) Latency (b) Data size

Fig. 8. Web access performance in terms of access latency and the total amount of data transferred



(a) Video quality (b) Data size

Fig. 9. Video playback performance in terms of video quality and the total amount of data transferred

a more powerful server and fast network access, TransCom can achieve comparable file access performance to a regular PC, and can potentially perform better than other remote file system solutions and that of VM-based approaches in stateless-thick-client-like systems.

E. Application Performance

In this section, we study how virtual disks impact real application performance. We compare TransCom (TRC) with the regular PC, the VM-based stateless-thick-client-like approaches(STC), and a number of commercial thin-client systems including Microsoft RDP 5.2 (included in Windows Server 2003) [10], Citrix MetaFrame XP server for Windows Feature Release 3 (with a client of Citrix ICA 6.2) [3], and VNC Viewer free Edition 4.1.1 [26]. All the thin clients run on the same platforms as the regular PC that we described in Section VI-A. The corresponding servers use the same configuration as the TransCom server.

We use Web browsing and Video playback as our two applications. The Web browsing performance was measured with Microsoft IE 6.0, using the Web text page load test provided by Ziff-Davis i-Bench benchmark suite 5.0 [17]. This benchmark consists of a sequence of 30 different Web pages, each containing mixed text and graphs. To compare the performance across other thin-client systems, we set the window resolution to 800x600. The video playback performance was measured using Windows Media Player 9.0 to play a 21 second (320x240 pixels, 24 frames/s) video clip displayed at 800x600 resolution. For both applications, we used a packet monitor to capture network traffic and measured the performance using slow-motion benchmarking [23], which allows us to quantify performance in a non-invasive manner.

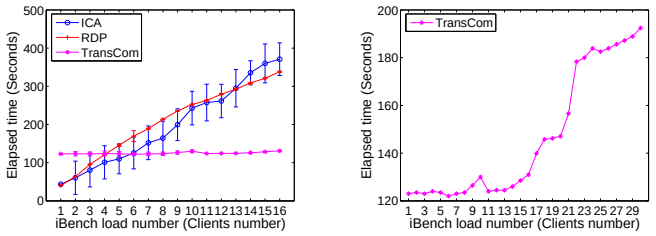
For Web browsing, we examine both the average page download latency and the average amount of data to be transferred between the client and the server with different network speeds. We observe that TransCom achieves similar performance to the regular PC. Since RDP and VNC ran the Web browsers on the more powerful server, they incurred smaller client-perceived latency than TransCom, but required a larger amount of data to be transferred between the client and the server. The ICA client experienced a significant higher browsing latency, even though the system transfered the least amount of data by using a higher compression rate algorithm for the screen display. It is observed that there is a little more latency than regular PC but similar data size in VM-based approaches. This means that while it require only a small data transfer as in PC and TransCom, the stateless-thick-client approaches decrease the computing performance due to its virtualization of both CPU and memory besides disks.

For video playback, we use the playback quality, defined in the slow-motion benchmarking, in addition to the data transfer size as our performance metrics. Since Vdisk read throughput with a powerful server is comparable or even better than the hard disk throughput on the regular PC, the TransCom client achieved as good playback quality as the regular PC, significantly outperforming other thin-client solutions. In these thin-client systems, the servers need to perform expensive video clip decoding for sending display data to the clients, resulting in longer server processing latencies and a larger amount of data to be transferred, which explains the low playback qualities at clients. However, due to its virtualization of both CPU, memory and graphics, the stateless-thick-client approach results a decrease in video quality while a similar data to be transferred to PC and TransCom.

In summary, these two experiments demonstrate the robust performance of TransCom across popular applications of different types. They show that TransCom can achieve better performance than of virtualization of all hardware resource in the stateless-thick-client approach. They also show that TransCom can achieve better performance than thin-client solutions, especially for applications that require heavy computation, although a new optimization method for thin-client display protocol [1] can reduce the amount of network transfer data, which may potentially improve the thin client performance.

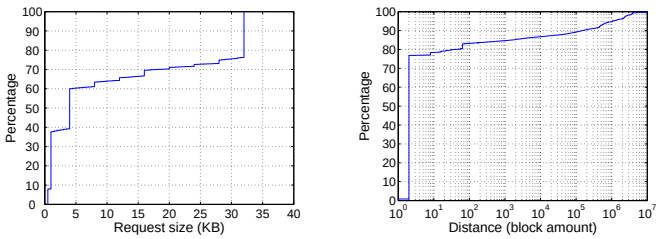
F. Scalability

We proceed to study the scalability of TransCom by varying the number of clients in the system. We again used i-Bench 5.0, but without slow-motion benchmarking, as our workload to evaluate the average client latency of the entire i-Bench run, and compared the performance with both ICA and RDP thin-client systems (Figure 10(a)). We observe that when the number of clients is smaller than four, ICA and RDP achieve lower client latencies than TransCom. However, as the latencies increase linearly with the number of clients in both ICA and RDP, the client latency in TransCom remains constant with small deviations, suggesting that TransCom is more scalable. This is because the TransCom server handles



(a) Client scalability (b) TransCom scalability

Fig. 10. (a) Client observed i-Bench run latency by varying the number of clients in the system. We plot both the mean and the standard deviations (b) TransCom client observed i-Bench run latency by varying the number of clients in the system. We plot only the mean.



(a) Block size (b) Block distance

Fig. 11. (a) The distribution of the read request block size (b) The distribution of the initial block position distance between two consecutive read requests

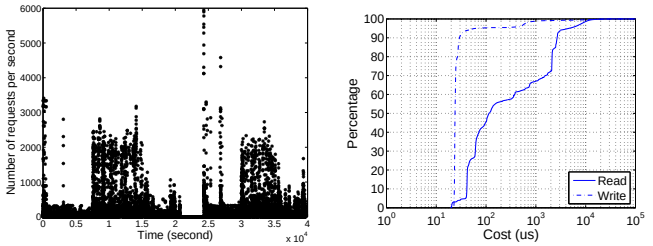
only Vdisk access requests, while thin-client servers perform both file access and computing tasks. In order to further investigate the scalability of TransCom, we also study the performance with a larger number of clients and present the in Figure 10(b) with a small scale. This shows that when the clients number is small, the slope is little, but when the number is larger than 15, the slope gets bigger, but the elapsed time is still smaller than that of ICA and RDP.

G. Real Usage Experiment

TransCom has been deployed in universities e-learning classrooms and used by students for daily usage. In this section, we study the real world usage of TransCom by instrumenting the server and letting it run for three days in one such e-learning classroom. This classroom has more than 100 client hosts, and is dedicated for students to learn English online from 8 am to 11 am and 2 pm to 5 pm each day.

For our experiment, we isolated 20 clients (the typical number of clients connected to a server) and connected them to our instrumented server. The server is an Intel Pentium IV 2.8GHz PC with 1G RAM, an 100Mbps network card, and an 80 GB 7200rpm soft RAID0 hard disk. For each disk request, we recorded the requested initial block number, the block length, and the operation type. There are about five million requests collected in total, with 80.5% being read requests, and the rest 19.5% being write requests.

1) Vdisk Access Workload: To characterize the request workload, we first examine the read request block size distribution in Figure 11 (a). The disk write request size is fixed to 2 KB in our real deployment. We observe that the majority of read request involved only a small amount of data, with



(a) Request arrival rate (b) Server process latency

Fig. 12. (a) The request arrival rate over time (b) The distribution of server disk request process latency.

60% of read access requesting less than 5 KB data each time. We have shown in Section VI-B.2 that the TransCom client read throughput (with a powerful server) can be higher than the regular PC for small size disk read access. Thus in these real world scenarios, TransCom users will experience similar or even better disk performance compared with a regular PC.

We further study how many read requests can benefit server memory cache, which is a key factor for achieving high Vdisk performance. We plot in Figure 11 (b) the distribution of the distance between the start block numbers requested by two consecutive read access. A small distance means two consecutive disk read requests have strong spatial locality in accessing data. After the first request, the data requested by the second read request are likely to be in the server memory. We observe that more than 75% of the consecutive read requests asked for data whose initial block numbers differed by only a few blocks, suggesting disk read requests have strong temporal and spatial locality in real usage, where TransCom is most helpful in performance.

2) Real Usage Performance: Given the 20 TransCom clients, we examine both the request arrival rate and the server process latency in such real world scenarios. Figure 12 (a) plots the client request arrival rate over time for one of the three data collection days. The request patterns in the other two days look similar. We observe that for most of the time, the request rate was low, on the order of hundreds of requests per second. There were some request burstiness during the morning and afternoon active usage periods, with peak rate as high as 6000 requests/second. The short quiet period with no request was the noon rest time.

Figure 12 (b) shows the server request process latency distributions for both read and write disk access. Because the write request size is maximum 2 KB (4 disk blocks), more than 90% of the write requests finished in tens of microseconds. For read requests, due to their different sizes, the server process time also varied. The majority of them (more than 60%) finished within 1 ms. While there were a small percentage ($\leq$ 10%) of requests having a relative long processing latency due to perhaps request burstiness, all disk access requests finished within 100 ms.

The strong locality of disk access patterns suggest that by using large memory and fast I/O servers, TransCom may outperform regular PCs with local disks in terms of real usage performance.

VII. Discussion

In this section, we discuss possible extensions and optimizations to TransCom for enhancing the system performance, robustness, and security.

The use of explicit caches at both client side and server side can potentially enhance performance significantly. At the client side, Vdisk driver cache can reduce the number of network communications, which latency is the current performance bottleneck. At the server side, we can exploit the locality of read requests across different clients by using a Vdisk image cache, and optimize write request using application level write optimization schemes (e.g., lazy write). The Vdisk access protocol can be further optimized to improve performance. In our current implementation, remote disk requests are sent in sequential order. As future work, we can enhance the disk access latency and throughput by sending multiple remote disk requests concurrently.

So far, we have not discussed how to generate the contents of Vdisk images at TransCom server. In our current implementation, the contents of Vdisk images exactly simulate those of a hard disk, with several special, initial blocks proceeding the data blocks. These special blocks are hardware dependent, containing disk parameters such as disk capacity, cylinders, heads, and sectors. Thus Vdisk image files, in particular the mono disk image, can work for only homogeneous hardware machines. Supporting machines with heterogeneous hardware is our ongoing work.

Our current prototype does not implement automatic server fail over. When server crashes or the mono disk image needs to be updated, the entire system needs to be manually shutdown and reboot. As future work, we plan to use server replication mechanisms (e.g., [13, 14]) to handle these scenarios, where clients can switch to an identical backup server when required.

TransCom server needs to prevent and detect unauthorized access of information. The current use of MAC addresses for authentication can not protect against malicious attackers that spoof client MAC addresses. A user-level authentication mechanism (e.g., [22]) might help mitigate the possibility of such attacks. We can also augment the current system using various encryption based approaches to protect the privacy of disk data access.

VIII. Related Work

There has been extensive research on distributed and pervasive computing platform. Our work is mostly related to systems such as network computers, thin-clients, network file systems, and virtual machine based systems.

In order to deal with the management challenge of personal computers, network computers, such as the Java Station by Sun [15], are proposed to replace the personal computers. Such solution supports WWW & Java applications only, and does not work with general commodity OSes or other applications such as Microsoft Office.

Thin client systems have been very popular, by providing a full featured desktop to users with low management costs, thus being considered as a pervasive computing platform. Example systems include Microsoft RDP [10], Citrix ICA [3], Sun Ray 1 [34], VNC [26], and MobiDesk [2]. In the thin-client system paradigm, all computing tasks are performed at the central server, while a client works only as a user interface by performing display, and keyboard/mouse input/output functions. Although such systems also achieve centralized management, they 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. Furthermore, there is no isolated user performance guarantee.

Network file systems and devices, such as NFS [28], AFS [16], and NAS [12], are popular solutions for sharing data in distributed enterprise environment. Although these systems can be used to share user files flexibly, they generally do not support sharing system files for the reasons we described in Section IV-B.

Our idea of centralizing storage while distributing computing is similar to the concept of diskless computers (e.g., [6, 9]) 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. Further, Vdisks perceived by TransCom users can be flexibly mapped to Vdisk image files on the server. Such flexibility allows TransCom to share OS and application software across clients to reduce the storage and management overhead, while still isolating personal files for user privacy.

The iSCSI protocol has been used to access disk blocks through network communication [29]. In particular, the iBoot [18] project at IBM has proposed a method that can remotely boot a commodity OS through iSCSI. An iBoot client needs a special type of BIOS ROM to carry out the OS boot process, hence is not generally applicable. For better performance, TransCom can potentially adopt iSCSI to replace the current Vdisk access protocol, but may need to modify it in order to fit the small size client BIOS memory.

The concept of resource virtualization has been introduced long ago and recently has been adopted to address security, flexibility, and user mobility. For example, commercial products such as VMware [36] have extended the concept of virtual machines to support multiple commodity platforms. The disks in these virtual machines are also virtualized, but reside in local host machine and accessed through the file system of the Host OS. In contrast, TransCom virtual disks locate in the remote server, with different types of Vdisks for sharing and isolating data among users.

VM-based stateless thick client approaches, such as ISR (Internet Suspend/Resume [**?**]) use virtual machine technology (e.g., VMware) together with a network file system (e.g., Coda [31]) to support user mobility. Each ISR client runs OS and applications on top of a preinstalled 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 TransCom, client OSes are running directly on top of the CPU, memory and graphics resource.

SoulPad [4] is another project that used virtual machine

concept for mobility, with a portable storage device for storing the entire virtual machine image. The Collective project [5] proposed a cache-based system management model based on virtual machines to reduce the management tasks of desktop computing. Similar to TransCom, it also uses different types of virtual disks, among which there is an immutable system disk to protect it against outside threats. Compared with Collective, TransCom uses a COW file semantic instead of COW disks. Moreover, TransCom adopts on demand block level disk access instead of using network file systems such as NFS to access and cache disk images.

IX. Conclusions

We have developed TransCom, a novel virtual disk based pervasive computing architecture for enterprise environments. TransCom clients store all data and software on virtual disks that correspond to disk images located on a centralized server. By using disk-level remote data access, TransCom can flexibly support running heterogeneous services, like OSes including Windows.

We have performed both testbed experiments and real usage experiments to evaluate TransCom. We show that by using a powerful server, TransCom clients can achieve comparable or even better disk and application performance than regular PCs with local hard disks. It is more scalable than traditional thin-client systems and other recent suggested pervasive computing systems.

Future work includes further optimizing TransCom performance, increasing the system robustness, enhancing the system security and supporting more types of devices and networks.

```
// The function returns a pointer to the file to be
// opened
// Input variable F is the name of the file
// mode is the operation mode (read or write) for open
//
fptr_t redirect_open(string_t F, opmode_t mode)
{
  fptr_t Fp;

  // open a user file F on the private_disk
  if F locates on the user_disk
  {
    Fp = open_file(F, mode, "private_disk");
    return Fp;
  }

  // F is a system file
  if F has a copy F' on the shadow_disk
  {
    Fp = open_file(F', mode, "shadow_disk");
  }
  else {
     if (mode == "read")
     {
       // no customized copy of F exists
       // open the default one shared by all clients
       Fp = open_file(F, mode, "mono_disk");

     }
     else if (mode == "write")
     {
       // need to customize F
       // open a new one in the shadow_disk
       Fp = open_file(F, mode, "shadow_disk");
     }
  }

  // return the file pointer
  return Fp;
}
```

Fig. 13. Pseudo-code for the file redirector to handle an `open` file system call.

References

[1] R. Baratto, L. Kim, and J. Nieh. THINC: A Virtual Display Architecture for Thin-Client Computing. In *Proc. of the Twentieth ACM Symposium on Operating Systems Principles*, 2005.

[2] R. A. Baratto, S. Potter, G. Su, and J. Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proc. of the 10th Annual International Conference on Mobile Computing and Networking*, 2004.

[3] I. Boca Research. Citrix ICA Technology Brief, Technical White Paper. Boca Raton, 1999.

[4] R. Caceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with Portable SoulPads. In *Proc. of ACM/USENIX MobiSys*, 2005.

[5] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-Based Systems Management Architecture. In *Proc. of NSDI*, 2005.

[6] D. R. Cheriton and W. Zwaenepoel. The Distributed V Kernel and its Performance for Diskless Workstations. In *Proc. of the 9th ACM Symposium on Operating Systems Principles*, 1983.

[7] L. Chung, J. Gray, B. Worthington, and R. Host. Windows 2000 Disk IO Performance. Technical Report MS-TR-2000-55,Microsoft Research, 2000.

[8] L. P. Cox, C. D. Murray, , and B. D. Noble. Pastiche: Making Backup Cheap and Easy. In *Proc. of Fifth USENIX Symposium on OSDI*, 2002.

[9] B. Croft and J. Gilmore. Bootstrap Protocol (BOOTP). RFC 951, 1985.

[10] B. Cumberland, G. Carius, and A. Muir. Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference. Microsoft Press, 1999.

[11] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, 1997.

[12] G. A. Gibson and R. Y. Meter. Network Attached Storage Architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[13] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer*, 30(4):38–74, 1997.

[14] A. Helal, A. Heddaya, and B. Bhar. Replication Techniques in Distributed Systems. Kluwer Academic Publishers, Aug. 1996.

[15] R. G. Herrtwich and T. Kappner. Network Computers – Ubiquitous Computing or Dumb Multimedia? In *Proc. of Third International Symposium on Autonomous Decentralized Systems*, 1997.

[16] J. H. Howard, M. L. Kazar, and S. G. Menees. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

[17] i-Bench. `http://www.veritest.com/benchmarks/i-bench/`.

[18] iBoot—Remote Boot Over iSCSI. `http://www.haifa.il.ibm.com/projects/storage/iboot/index.html`, 2007.

[19] Iometer. `http://www.iometer.org`.

[20] P. Leach and D. Perr. CIFS: A Common Internet File System. Microsoft Interactive Developer, 1996.

[21] J. Ma, L. T. Yang, B. O. Apduhan, L. B. Runhe Huang, and M. Takizawa. Towards a smart world and ubiquitous intelligence: A walkthrough from smart things to smart hyperspaces and ubickids. *International Journal of Pervasive Comp. and Comm.*, 1(1):53–68, 2005.

[22] B. C. Neuman and T. TSó. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, 1994.

[23] J. Nieh, S. J. Yang, and N. Novik. Measuring Thin-Client Performance Using Slow-Motion Benchmarking. *ACM Transactions on Computer Systems (TOCS)*, 21(1):87–115, 2001.

[24] Intel Corporation. Preboot Execution Environment (PXE) Specification, Version 2.1, 1999.

[25] RedFlag Linux. `http://www.redflag-linux.com/eindex.html`.

[26] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper.

Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[27] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3):25–31, 2003.

[28] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *USENIX Association Conference Proceedings*, 1985.

[29] J. Satran, C. S. K. Meth, M. Chadalapaka, and E. Zeidner. Internet Small Computer Systems Interface (iSCSI). RFC 3720, 2004.

[30] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[31] M. Satyanarayanan. The Evolution of Coda. *ACM Transactions on Computer Systems*, 20(2):85–124, 2002.

[32] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, D. R. O'Hallaron, A. S. abd A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, M. A. Kozuch, C. J. Helfrich, P. Nath, and H. A. Lagar-Cavilla. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing*, 11(2):16–25, 2007.

[33] K. Sollins. The TFTP Protocol. RFC 1350, 1992.

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

[35] N. Tolia, D. G. Andersen, and M. Satyanarayanan. Quantifying Interactive User Experience on Thin Clients. *IEEE Computer*, 39(3):46–52, 2006.

[36] VMware Workstation. `http://www.vmware.com/products/ws`.

[37] W. Vogels. File System Usage in Windows NT 4.0. In *Proc. of the Seventeenth Symposium on Operating Systems Principles*, 1999.

[38] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[39] Etherpeek 4. `http://www.wildpackets.com`.