Implementing Portable Desktops: A New Option and Comparisons

Muthukaruppan Annamalai, Andrew Birrell, Dennis Fetterly, and Ted Wobber

Microsoft Research
One Microsoft Way
Redmond, WA 98052

Microsoft Research Technical Report
MSR-TR-2006-151
October 2006

Abstract. We consider the problem of using a wide variety of computers, dispersed geographically and with varied connectivity, while wanting to have each of them provide you with the same, personalized, desktop environment: operating system, customizations, applications and your documents. We describe the existing range of solutions available for this problem, and introduce and characterize the new "Desktop on a Keychain" system. We explain the tradeoffs that these various systems incur, and the ways in which we believe they are good and bad. Finally, we explore some of the security issues involved in using such systems.

1 Introduction

Computing is becoming ubiquitous. An adequately equipped desktop PC can be bought for $300 (item #3321522 at walmart.com); an Internet connection with 1 Mbit/second download speed can be rented for $12.99/month ("DSL Express" at sbc.com). The Internet is available at work, at home, at Starbucks; it's provided to you on a train from Edinburgh to London, or on a flight from Boston to Frankfurt; you can even access the Internet on a mountain top via your cell phone.

What's not readily available to you is your own desktop computing environment: your documents, your applications, your customizations, your choice of operating system and patch level. Several attempts have been made recently to remedy this. In the current paper, we try to characterize this problem area and the other attempts, and to describe our own experience with one such attempt.

Our goal, in an ideal world, is that whenever you have the use of a computer, and whenever that computer is connected to the Internet, you have the full power and customization of your own personal computing environment.

If the only computer that you ever use is your personal laptop, and you carry that laptop with you everywhere, then you will have achieved our goal. Unfortunately, you will have done so at a cost: laptops are big, heavy, and fragile. It's often inconvenient to provide them with enough power. They are always ergonomically bad, and a major contributor to problems such as RSI injuries. They are significantly more expensive than their desktop equivalents, and will fundamentally always be so. They provide our desired ubiquity of computing, with full personalization, but at a high cost.

At the other extreme, you can carry nothing with you, but instead rely on the computers that you find scattered around the world. Of course, to do so you need, at a minimum, some way to access your personal documents. You can carry the documents with you, for example in a USB flash disk. Or you can keep your documents on some form of network storage and access them remotely.

Unfortunately, if the only personalization of the computing environment that you are using is to have your documents available, you will be unhappy. You'll be most unhappy if the application you want is unavailable. But you will also be unhappy if the wrong version of your application is available locally. Finally, to the extent that you believe in the utility of customizing application behavior (apparently a very popular feature), you will be unhappy without your own customizations.

We see four major categories of solution to this problem: remote access to your desktop environment; web-based applications; automatic migration of your desktop state; and carrying a device that reflects your desktop state. The purpose of this paper is to compare these solutions, exemplified by some recent work in this area, and to introduce our prototype device-based solution called "Desktop on a Keychain" which we refer to as DoK for the remainder of this paper.

2 Remote Access Protocols

This class of solutions is often described as "thin clients". Your desktop environment, with your documents, applications and customizations resides permanently on a single computer attached to the Internet. When you use some particular local computer, you run there a client application that implements a networking protocol such as Microsoft's "Remote Desktop Access" [15], or Sun's "Sun Ray" [23], or the open source "VNC" service [18]. Your only use of the local computer is for display, keyboard, and mouse: all the real work happens on the remote server.

These solutions provide precise ubiquity of your desktop. They also make minimal demands on the local computer. You must have the appropriate remote access client installed there, but you care little about any other local software, and the computational demands on the local computer are slight.

The most obvious issue with these solutions is reliability: you are totally dependent on the continued operation of your remote desktop computer. If the remote computer is professionally managed, as in the typical Sun Ray installation or as when using Microsoft's remote desktop to connect to a shared server, this will most likely be fine. But if you attempt to rely on this solution to connect to a personal computer, the reliability is likely to be poor: nobody will reboot your desktop (or home) computer for you when you are out of town.

The more serious issue though is performance. With thin client solutions, all of your interactions involve round-trip communication with the remote server. The protocols used by these systems are quite smart about screen repaints, and use sophisticated compression technology to minimize bandwidth requirements, but they still become unpleasant at less than about 300 Kbits/second [16]. More critically, they rely on very low latency communication, without which the system will become unpleasant to use. For example, Sun's documentation for Sun Ray says:

> "The lower the latency, the better; latencies under 50 milliseconds for round trip delay are preferred. Latencies up to 150 milliseconds provide usable, if somewhat sluggish, performance." [17]

Low latencies are easy to achieve within a local area environment, but they degrade rapidly as you move to wide area. For example, a network round-trip from a residence in Los Altos connected to the Internet with DSL, to the adjacent www.stanford.edu server produces a typical latency of 15 milliseconds. From Los Altos to www.washington.edu, about 800 miles away, is 31 milliseconds round-trip. Both would probably give acceptable Sun Ray performance. But if we try www.mit.edu, requiring a trans-continental link, the round-trip latency increases to 100 milliseconds; transatlantic communication to www.cambridge.ac.uk results in round-trip latency of 150 milliseconds.

These latency numbers are not going to get much better, ever. The 6300 mile round-trip to MIT would take 48 milliseconds at 70% of the speed of light, without any routing delays. And while the routing delays might get better, they are never going to disappear. (Today there are 13 routers between the California residence and www.mit.edu.) Consequently, we believe that thin client solutions will never be fully satisfactory.

3 Web-based Applications

The concept of web-based applications is an appealing alternative to remote access protocols. As with remote access protocols, the application logic and your documents and customizations live on a network server. But the fine-grain, low latency user

interaction takes place on the local computer. The local computer interacts with server-based applications through HTTP transactions.

Doing this naïvely, with purely static client-side HTML pages rendered by a web browser, is unsatisfactory except as a fall-back. Static HTML pages provide quite poor user interaction capabilities, and the latency of an HTTP request and subsequent page rendering for each state change of the application provides an unacceptable user experience.

For many years it has been possible to do much better than the naïve approach. For example, in 1997 the Pachyderm prototype [2, 5] provided a web-based email system that was as user-friendly and interactive as using a local email user agent. This system offered the added benefit of ubiquitous access, since your email and application state remained at all times on a network server. Pachyderm did this by using a Java applet for the fine-grain user interactions and asynchronous HTTP transactions for the transactions with the underlying application on the server. In most situations the network latency of the application transactions was hidden from the user by appropriate use of read-ahead and write-behind.

More recently, similar results have been achieved with the technology known as "AJAX" (for "Asynchronous JavaScript and XML"). AJAX [24] comes from the confluence of three techniques. First, current web browsers allow scripts to dynamically modify every part of the current web page (by writing to the "innerHTML" property of any element of the page), and the browsers then dynamically re-render the page as required. This lets client-side scripts interact with the user without needing round trips to the network server. Second, current web browsers provide an object (known for historical reasons as "XMLHTTP", and originally provided by Microsoft as an ActiveX control) that lets client-side scripts make arbitrary HTTP requests without disturbing the contents of the browser window. Third, the client-side scripting system and the XMLHTTP object allow scripts to perform operations, including HTTP requests, asynchronously. This lets the script use read-ahead and write-behind to hide network latencies.

AJAX technology became popular with the appearance of Google's Gmail application [6] and Google Maps [8]. While Gmail is functionally extremely similar to the earlier Pachyderm system, the use of AJAX technology makes the client side significantly more user friendly.

Overall, it is possible to use web browsers and HTTP in ways that provide high quality user interaction, hiding most or all of the necessary latency incurred in communicating with a server-based application.

So there is some prospect that web-based applications will indeed provide the ubiquitous access that we desire. They don't do everything, though. First, web-based applications don't provide a general-purpose computing environment. The only applications you can use are ones that have been specially written for the web-based situation, with the appropriate separation of user interactions from application state changes, and appropriate asynchrony to hide network latencies, and careful avoidance

of persistent state on the client side. Second, this approach relies absolutely on the accessibility of the network server. The server must be always running, and your network connection must be perfectly reliable; in the absence of either, you will make no progress whatsoever. As we discuss in Section 8, there are also security concerns about using a local web browser.

Overall, while web-based applications in general (and AJAX in particular) provide a promising technique for building new applications, they don't really meet our current goals for ubiquitous access. Also, it remains to be seen how widely applicable this technology is.

4 Carrying It with You

Considering all of this, we believe that now, and for several years, the most attractive solution will be to carry most of your data and customizations with you, and execute your applications on whatever computer is available locally, while relying on network access and storage as a secondary medium.

Fig. 1: Desktop on a Keychain System Outline

The underlying enabling technology for this approach is the availability of cheap, portable, large capacity stable storage, which can be easily, efficiently, and transiently connected to the local computer (typically with USB 2.0). For example, you can buy a 5 GByte USB disk (using a 1" magnetic disk) for about $100. You can buy a 1 GByte USB disk built out of flash ROM for $60, or 4 GBytes of flash for about $200. All of these devices provide end-to-end throughput in the region of 10 MBytes/sec,

plenty for most applications (though as we'll see later, current mass-market flash-based devices do have some performance issues). Flash-based devices will be particularly attractive over the next few years, since the capacity physical devices is widely predicted to increase at 30% a year or better, at fixed cost, for several years. Considering these trends, your cell phone might easily become an attractive platform for storing your portable desktop as you move between computing kiosks.

In Section 4 of this paper we'll describe our DoK design for using such USB devices, while comparing it with the previously published "Soulpad" design [4]. In Section 5 we consider the alternative "Internet Suspend and Resume" design. Section 6 summarizes the performance of our design, and Section 7 considers some performance issues common to current USB flash disks. Section 8 presents a discussion of security issues related to the use of kiosk computers in general, and Section 9 concludes.

As with Soulpad, our DoK system aims to give you ubiquitous access to your own customized operating system, your own customized applications, and your own documents through a portable USB device that you carry with you everywhere. To achieve this, the basic usage scenario is that you plug your USB device into the local computer, and then arrange to execute your own operating system, from the USB device, on the local machine, as depicted in Figure 1.

This produces an immediate difficulty: even within the moderately well standardized hardware of an x86-based personal computer, there are numerous detailed differences among the physical devices, requiring differing device drivers within an executing operating system. The solution to this, with our system and with Soulpad, is to execute your operating system within a virtual machine monitor. For DoK we chose to use Microsoft's VirtualPC product; Soulpad uses VMware. For the present purposes, the two virtual machine products have very similar characteristics, and very similar performance.

The use of virtual machines and disks, as opposed to simple, mountable storage, carries an additional benefit when dealing with flash or portable-disk environments. Flash, and to some extent portable rotating media, will have lesser capacity than traditional desktop or network storage units for the foreseeable future. Instantiating a virtual machine, complete with applications, within a richer, home-base computing environment should facilitate data transfer to and from the portable system.

4.1 Suspend, Resume, and Boot

The Soulpad and DoK designs differ in how they arrange to execute the virtual machine. Soulpad boots directly from the USB device, into the Knoppix [13] Linux distribution. Then VMware is run under Knoppix, and within VMware you resume your virtual machine execution. Our system uses the local machine's installed operating system (Windows, in our case), and runs VirtualPC there, then resumes your

virtual machine session. The main effect of this difference is to trade off self-containedness for startup performance. Booting Knoppix from scratch takes about a minute, whereas starting VirtualPC under an existing copy of Windows takes only a few seconds. There are also secondary effects. The Knoppix approach is somewhat fragile, depending on having appropriate drivers for all possible hardware that you might encounter. Alternatively, our approach requires Windows and VirtualPC on the local machine (or Windows plus the authority to run VirtualPC from the USB device). See also our later discussion about security issues.

In either system, once the virtual machine monitor is executing, the user experience is similar to using suspend and resume on a laptop. When preparing to depart from a physical machine, your virtual machine writes out its state, mostly its DRAM image, to the USB disk. After arriving at a new physical machine, you tell the virtual machine to resume by reading its state from the USB disk. As seen by your applications, nothing much happened: they still access your virtual disk, and the other virtualized devices (such as the network and the display) provided by the virtual machine software.

With our DoK system, to total time taken to resume your computing environment, measured from when you plug the USB device into the local computer, is about 30 seconds (for a virtual machine that is currently using 256 Kbytes of DRAM; the speed varies significantly with memory usage).

This decision, whether to boot from the device or utilize the existing host operating system, is independent of the other major difference in the designs (discussed next), namely whether to depend on network resources. With sufficient engineering effort, the DoK approach can likely be made to boot off flash and Soulpad could be modified to take advantage of a pre-existing VMM to improve startup performance.

4.2 Reading from Your Virtual Disk

Soulpad uses a straightforward organization for your virtual disk: it's all on your USB device, which in the existing prototype is a 2.5″ magnetic disk. By contrast, we were not willing to assume that your USB device had sufficiently large capacity to carry all of your data with you. We wanted to work with physically tiny USB flash disks, small enough to literally carry on your keychain. Commonly available USB disks can hold up to 2 GBytes today, probably about 4 GBytes next year. For many users, this is too small for the entire operating system plus applications plus documents.

The basic step in avoiding this issue is to view the USB device as a cache of your virtual disk contents. The base copy of your virtual disk lives on the Internet, accessed by a standard remote file protocol (SMB in our case). When the virtual machine reads from its virtual disk, it first looks for content on the USB device, and only if the content is not found there does it read from the network. We implement

this primarily by using the "differencing disk" mechanism of VirtualPC. The virtual disk stored on the USB device contains only those blocks that differ from a disk image stored remotely on the network file server. We made two enhancements to the basic differencing disk machinery in order to achieve satisfactory performance.

First, we modified VirtualPC to keep a read cache on the USB device. Whenever the virtual machine needs to read data from the network disk image, it now records that data on the USB device, in an auxiliary file there. We manage this cache in a straightforward LRU manner.

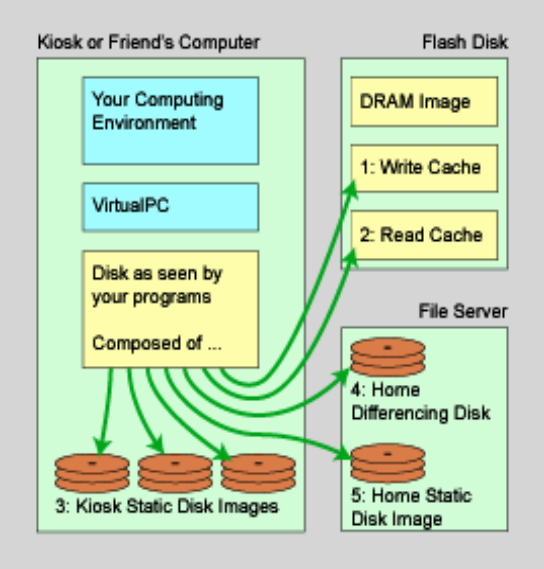

Fig. 2: Storage Abstractions

Second, we observed that much of the data read from your virtual disk is static and not at all personal: the binaries of executable programs and libraries. Further, the local computer has vast disk space available to it. So we posit that any local computer being used in this manner should be equipped with content similar to the static parts of your virtual disk content. For example, a corporate IT department could equip all such public computers with copies of each version of the IT department's standard disk images. Or a public kiosk computer could be equipped with copies of the various popular operating systems are various patch levels. We then modify VirtualPC further, so that when it wants to read content from your virtual disk, and the content is not available from your USB device, then before going to the network the VirtualPC system checks for appropriate content on the local computer. If the appropriate content is on the local computer, the read is satisfied from there, without any network operations. (We could, at that point, cache the data on the USB device, but we chose not to because of worries about the device's capacity.)

Finding the "appropriate" content on the local computer is not especially difficult. We use a content-based addressing scheme similar to several previously discussed in the literature [9, 17]. We maintain on the USB device, and dynamically as an array inside VirtualPC, fingerprints of blocks of your virtual disk. More specifically, we maintain an array of cryptographic hash fingerprints of each 16 KByte chunk of your logical disk. Similarly, we maintain an auxiliary data structure on the local computer's disk, containing the fingerprints of the chunks of the disk images stored there. When a disk read cannot be satisfied from the USB device, we determine the corresponding fingerprint of that chunk of your logical disk (from the DRAM array), and determine whether a chunk with the appropriate fingerprint exists on the local computer. If so, we read locally and only if there is no such local chunk do we read from the network file server.

The overall effect of this read design is that the data stored on the USB device is dominantly a cache, containing the working set of your personal data, including any application configuration information. Static data, such as operating system, libraries, programs and templates, mostly gets read from the local computer. The only network read operations are for personal data not in your existing working set, and not available in the local static disk images.

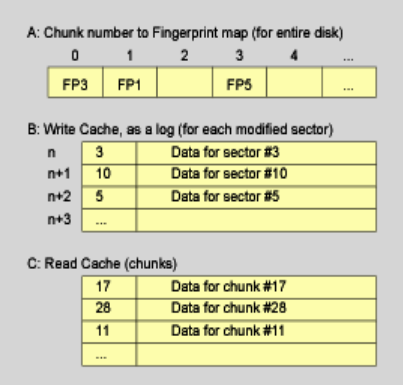

Fig. 3: Data Structures on the USB

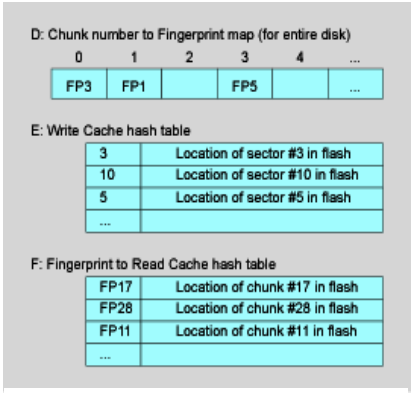

Fig. 4: Volatile Data Structures

Notice that we make no assumptions about any sort of "correctness" of the local images. If a chunk exists locally with the correct fingerprint, it necessarily contains the correct data, the same as we would have obtained by reading from your virtual disk over the network. At worst, if the data structure describing the fingerprints of the local computer lies, our performance degrades by needlessly reading data from the local computer, then discarding it because its fingerprint is wrong.

4.3 Writing to the Virtual Disk

Writes are comparatively simple. When VirtualPC writes to your virtual disk, we write the data to the USB device. Sometime later, at your convenience (or when the USB device is becoming full) the new content is written back to the network disk image. Of course, this write-behind cache is the first place that we look for content when reading from your virtual disk.

Figure 2 shows the storage abstractions involved in accessing your logical disk. Figure 3 summarizes the data structures kept on your USB device, and Figure 4 summarizes the volatile data structures that we use while running VirtualPC.

One auxiliary benefit of the write behind cache is that your disk image on the file server always contains some previous consistent version of your disk. If you lose the USB device, or it breaks, you can recover a previous state by starting over with the network disk image. Finally, users have a backup system that requires no manual operations.

As implemented in our prototype, we flush the entire write-behind cache to the network disk image as a single operation. However, it would also be appealing to retain the order of writes, so that the network disk image would go through the same intermediate states as your overall virtual disk did, allowing you to back up some arbitrary point in your history.

4.4 Contrast with Soulpad

As we discussed in Section 4.1, one difference between the Soulpad and DoK systems is the way in which they start up on arrival at a new local computer. Soulpad boots Knoppix from the USB device, and our system runs VirtualPC under the local computer's own operating system. While this changes details of the system requirements and initial start-up behavior, it is not fundamental.

The fundamental difference between the systems is that DoK takes advantage of, and relies upon, the availability of network file storage. This has several ramifications, some good and some bad.

On the bad side, if you don't have network connectivity, or if the network file server is for some other reason unavailable, our system does not function. (We could function somewhat purely from the contents cached on the USB device plus what we find on the local computer, but succeeding in this approach is the same as the long term research problem of detached operation explored by systems such as Coda [12]. We have chosen not to explore that direction.) Our belief, based on our observation of current industry trends, is that network connectivity is, or soon will be, sufficient to make the smaller form factor of USB devices attractive in many scenarios. On the other hand, we have no illusions that our solution would work as well as Soulpad where connectivity is limited or non-existent as in much of the developing world.

On the good side, with DoK your portable computing environment is not limited to the capacity of your portable USB device. Opinions may vary about how large a portable device can conveniently be. If you are willing to transport a 2.5″ magnetic disk, then today you can transport 100 GBytes and thus pretty much all of your active personal environment. We do not believe that most people would be happy with a 5 ounce device, 5 inches by 3 inches, in their pocket. Wireless-enabled personal server devices containing rotating media such as those described by Want et al. [27] eliminate the need for a physical connection, but create questions concerning battery life and weight.

If we restrict the choice of devices to flash-based media, or miniature magnetic disks, we find that today's maximum capacities are in the region of 4 GBytes. There is no prospect in the next few years that the capacities of those devices will grow faster than users' appetites for data storage. Thus we remain convinced that a cache-based approach is attractive.

When flash disks reach 16 GBytes in capacity, there will be ample space to support a substantial environment without connecting to a network. We can expect 16 GByte flash devices to reach the price-to-performance "sweet spot" currently occupied by 1 GByte devices in perhaps 4-5 years. However, even with a system volume running entirely on local flash, it might remain attractive to maintain persistently-cached access to alternate network volumes in the fashion described above.

5 Migrating Over the Internet

Alternative designs are possible if you are willing to posit a global infrastructure of cooperating network storage servers. In their Internet Suspend/Resume (ISR) system, M. Satyananyanan et al. [22] propose an environment where desktop state migrates so as to give good local performance wherever a user might login. (Sapuntzakis proposes a related design [21] for which the following discussion is also relevant.) As with our DoK and system and Soulpad, ISR is a virtual-machine-based design. Users run their computations in a virtual machine. The virtual machine can be suspended. This implies that all state, both virtual memory and disk, is saved into a distributed file system. Such computations can be later resumed, possibly at distant locations.

In the baseline ISR design, the user's memory and disk image must be made available locally in order for a user session to proceed. In-memory caching is not sufficient, so the ISR storage layer uses the local disk for caching, for example, by using the Coda file system [12]. To compensate for Coda's full-file caching policy, virtual disks and virtual memory images are subdivided into smaller (128K) chunks and these chunks are stored as individual Coda files. Coda supports trickle reintegration of cached state to the home server, but even so, the dominant cost of suspend and resume in the un-optimized design comes from the network file system latency.

ISR makes use of two techniques to improve resume performance. First, the system can proactively cache a user's state at a likely destination. This, of course, requires some way of predicting that user's future itinerary. Second, ISR can also make use of demand-fetch caching. This can produce very fast resume latencies in some cases, but performance is strongly impacted by workload and network bandwidth. There is no similar trick to improve suspend performance. The user can only opt to wait for changes to be flushed to the network, or to risk allowing these to take place in background.

Portable storage devices (such as the USB flash disks we have been considering) can be used to further optimize the ISR scheme. Such devices are treated as look-aside caches. The distributed storage layer is modified to first provide a hash for each needed chunk in lieu of the data. Now any attached device can offer up the chunk if it holds one with the specified hash value. If the chunk is not available, it can be fetched from the distributed storage system as usual. Portable devices that support writing can be used to cache a copy of the virtual memory image at suspend time, thereby improving resume performance in the presence of the target device. The ISR authors don't propose any more aggressive role for use of portable disks in caching writes.

DoK and ISR occupy a similar portion of the design space, however there are important differences. First, ISR writes user data to the local disk, whereas DoK only writes to the portable device and the network. Although the user data ISR writes locally is in fact encrypted, the DoK system is designed to avoid writing *any* persistent state to the local machine. Second, ISR treats portable devices purely as caches, while DoK stores persistent state there that is not necessarily stored elsewhere. There is a fundamental design tradeoff here. If the portable device can store persistent state, then recent writes can be carried with the user rather than waiting for them to be flushed back to the distributed storage system. This yields significantly improved behavior when network bandwidth is limited. As an extreme example, imagine an airliner with wireless Internet connectivity and built-in computer kiosks at the seats. Flushing all writes with poor network connectivity (for example, on final approach) might be a severe constraint. Writes stored to removable and portable storage would most likely be preferable. Note that writes stored temporarily on a USB key can be efficiently migrated back to distributed storage when good network connectivity is available. The obvious downside is that the portable device can be lost or damaged in the interim. Finally, widespread kiosk deployment based on a distributed storage system like Coda requires a substantial management and trust infrastructure to handle pre-fetch and write-back, for example a standard platform and a federation of contributing administrative domains. In contrast, DoK needs little mechanism from the local operating system and management infrastructure other than a VMM and connectivity to your file server.

6 Desktop on a Keychain Benchmarks

There are several potential performance issues with the DoK design: the cost of using a virtual machine, the cost of transferring data via USB, the cost of our look-aside arrangements for detecting usable chunks on the local computer, and the cost of network data transfers. To evaluate the impact of these, we measured our performance using the SYSmark 2004 benchmark [1].

We ran the benchmark on a 3.2 GHz Pentium 4 PC equipped with 1 GByte of real memory. The local computer's operating system was Windows XP SP2. We used the currently shipping VirtualPC SP1 product, except that we replaced the file "vmm.sys" with the corresponding one from the Virtual Server SP1 product, because doing so fixes a performance bug in VirtualPC SP1's virtual memory implementation. The DoK system itself is a modification to the source code of the VirtualPC SP1 product.

We tested several configurations:

- *Raw* measures the benchmark running natively on the local computer, with its local hard disk. For this test we restricted the computer to using 512 MBytes of real memory.
- *VPC-Local* measures the benchmark running directly under VirtualPC, using a virtual disk stored directly on the local computer's hard disk, not using any of the DoK system. The virtual machine was configured to use 512 MBytes of memory; the real machine had its full 1 GByte.
- *VPC-USB* is the same as *VPC-Local*, except that the virtual disk resides on a USB 3.5″ magnetic disk.
- *DoK-USB-A* measures the benchmark running inside the actual DoK system, and is otherwise the same as *VPC-USB*.
- Finally, *DoK-USB-B* replaces the 3.5″ magnetic disk of *DoK-USB-A* with a 4 GByte USB flash disk.

Test Case	*Score*	*Slowdown*
Raw	141	1.00
VPC-Local	125	1.13
VPC-USB	114	1.24
DoK-USB-A	116	1.22
DoK-USB-B	105	1.34

From these raw numbers, we can derive more interesting summaries:

- The penalty for using VirtualPC (with the vmm.sys fix) is 13% (1.13/1.00), in this benchmark.
- The penalty for using USB instead of IDE is 10% (1.24/1.13).

- The DoK modifications to VirtualPC, including the read cache, the write-behind cache, and the look-aside to the local computer, produce no performance degradation (comparing *DoK-USB-A* with *VPC-USB*). Also, in this benchmark, delays caused by accessing the network file server had no performance impact. The slight speed-up seen here is probably caused by the look-aside reads from the local IDE disk being faster than reads from the USB magnetic disk.
- Using this particular flash-based USB disk is 10% slower than using a 3.5″ magnetic USB disk (1.34/1.22).

The bottom line is to compare *DoK-USB-B* with *Raw*, which results in an overall 34% penalty for using the full DoK system with a USB flash disk.

In conclusion, we believe that these performance numbers show the viability of this approach. Using a virtual machine, using a USB disk, and relying on a network file server do not have excessive performance penalties.

The astute reader will have noticed that we haven't specified which USB flash disk we used in these tests. This is because of some generic performance issues with flash-based disks, which we explore in the next section.

7 USB Flash Disk Performance

When we first starting using the DoK system, we noticed that in some situations the system seemed ran surprisingly slowly. These situations all seemed to be related to writing to the USB flash disk, and the performance of the disk in these situations seemed to be inexplicably slow. Since the effect applies to all of the mass-market USB flash disks that we have tested, and since it will degrade any use of such devices as a system (boot) disk, we are reporting the result here.

Mass market USB flash disks, typically referred to as "keychain" disks, typically export disk-block-level I/O interfaces. These devices contain on-board flash memory that is accessed through a specialized controller chip. The controller implements a *flash translation layer* that presents the USB mass storage class protocol [24] to the host computer. The translation layer is also responsible for managing writes and erasures so as to balance wear over the flash chip. Devices of this class typically advertise comparable performance characteristics: read throughput of 8 to 16 MBytes per second, and write throughput slightly slower at about 6 to 12 MBytes per second (depending on price). Since these throughput numbers are not a lot slower than low-end magnetic disks, we initially assumed this performance would be acceptable.

Write latency associated with erasure of flash memory has been understood for some time [6]. Log-structured file systems [19, 28] can be used to optimize flash erasure behavior and to promote wear-leveling. However LFS deployment is not commonplace in most computing environments and this constitutes a hindrance to the

portability of USB flash disks. We, therefore, opted to use conventional file systems and wound up investigating flash-translation-layer performance.

After we observed our system running unexpectedly slowly, we organized some micro-benchmarks for our USB flash disks. These benchmarks, running under Windows, called the Win32 file system API to perform reads or various lengths, writes of various lengths, and to measure these either sequentially or as random access. The results immediately showed that sequential reads, sequential writes, and random access reads all achieved the advertised throughput speeds, regardless of transfer size. But random access writes with moderate transfer sizes (e.g., 4 KBytes) consistently incurred an average latency of about 22 milliseconds, producing a net throughput of about 190 KBytes per second.

We next pursued the question of where this latency arose, since there is a lot of software between our test program and the underlying flash chip. The guilt became more localized after we used a USB analyzer, and confirmed that at the level of the USB mass storage protocol, the 4 KByte write requests were indeed taking an average of 22 milliseconds to complete.

We believe that the explanation for this comes from the difference between the "disk" abstraction, and the reality of the functionality of NAND flash chips. A disk is addressed linearly by logical block address ("LBA"). The write operation provides new data for a given LBA, and the disk ensures this data will be returned when reading that LBA. NAND flash provides memory linearly addresses by "page number", but with the constraint that each page can be written only once. To permit rewriting of a page, a separate erase operation is needed, which erases a "block". The most commonly used 1 GByte flash chip is the K9W8G08U1M from Samsung [19]. With the Samsung chips, the page size is 2 KBytes and the block size is 128 KBytes. Further, each block is only guaranteed to be erasable 100,000 times in the life of the chip.

To compensate for this, when the controller on the flash disk is asked to write new contents into some LBA, it must write the data to a newly erased page in the flash chip, at a different page number from the one use previously for this LBA. Consequently, the controller must maintain a table mapping LBA to the appropriate current flash page number.

If this table were at the granularity of one entry per LBA, there would be no performance impact. However, in that case the table for a 1 GByte flash disk would have 2 million entries, each entry requiring 21 bits. This exceeds the on-chip memory of controller chips, so the device would require a DRAM chip (and memory controller), significantly increasing its price. Alternatively, if the granularity of this table is, for example, quanta of 128 KBytes, then the table would require only 8192 entries and would fit on-chip. We believe this is approximately what is happening in the current mass-market USB flash disks (although there is a lot of variability in these performance numbers, so the situation is undoubtedly more complex than this analysis).

Now consider the performance impact of this granularity. If the disk is asked to rewrite the contents of a random LBA, an entry in this table must be modified, changing the location of an entire 128 KBytes of data. Consequently, the controller must now read that data from its old location on the flash chip, modify the appropriate contents, and write it into its new location. It turns out that this read-modify-write operation takes roughly 22 milliseconds on current flash chips, thus explaining our performance observation.

This behavior does not affect sequential write performance, since the sequential LBA contents can be written sequentially to a single erased flash block, at full speed. And in the primary application of these devices, storing photographs and other large files, random access performance is not important. But when used as the system drive for an operating system, the performance degradation is important. In addition, this read-modify-write behavior increases the frequency with which flash blocks must be erased, causing the total lifetime of the flash chip to be reduced.

This problem is solvable by building slightly more complex devices with the requisite DRAM. And such devices actually exist. For our benchmarking, reported above, we purchased a flash-based disk (M-Systems model #FFD-25-UATA-4096-N-A [14]) which was originally intended as a replacement for a 2.5″ IDE magnetic disk. We placed this disk in an external USB disk enclosure. This disk, designed primarily for the military market (and priced accordingly), does indeed have the performance we initially expected. There is no penalty for random access; it behaves exactly like a magnetic disk with a seek time of 0.

The write access patterns of your primary system disk are not at all like the bulk file transfer pattern for which the current mass-market flash disks are intended. It seems clear that the random access performance problem must be fixed if USB flash disks are to be used extensively as primary system disks, despite the inevitable price increase that will result. For this reason we view the performance on such a device as the most reasonable measure of our system.

Unfortunately, significant published material on the design and performance of commercial flash-translation-layer firmware is hard to find.. As a gedanken experiment, we created a reasonably complete design for such a system, with the appropriate structures to permit full random access write performance, while retaining correct recovery from power failures and from bad blocks, and with acceptable start-up time at power on even at large capacities. The details are beyond the scope of this paper, but are available in a separate technical report [3]. A multi-platform LFS implementation specifically for USB flash disks might offer similar benefits, although such a solution would be difficult to implement entirely in device firmware.

8 Security

Walking up to a computer that you don't know, and there resuming your personal desktop environment, is fundamentally a risky thing to do, regardless of which of the systems we've been discussing you use. The risks include, for example, the presence of a hardware keystroke logger attached to the keyboard [10], which might record your bank account's password as you type it. Even if you try to boot the computer from your own software (as with Soulpad), you have no guarantee whether that's what is really happening. The screen that looks so much like the BIOS booting from your USB device might be produced by the malware that just booted from the infiltrated hard disk.

There are known technical solutions to this problem [26], but none of them have been implemented in widely available hardware. The solutions start with a trusted secret key embedded in the computer's processor. Then by means of certificates issued by authorities that you choose to trust, the local computer hardware and its operating system and its applications can prove that they haven't been corrupted or subverted, and nor have their peripherals. Equipped with such certificates, the local computer could, for example, prove its trustworthiness to your network file server, and the file server could then tell the local computer to display an image known only to you and the server. Assured by this, you would reasonably insert your USB device, or resume your ISR session, or contact your web-based applications. But we repeat: these solutions are not available today.

Without such technical solutions, your decision to use an unknown computer is a judgment call. You must make this judgment in much the same that you decide whether to insert your ATM card into a slot and type your PIN. It's based on your opinion of the environment where you found the computer and your relationship with the apparent provider of the computer. It is also based on probabilities, and on the actual value at stake. If the computer is in your work environment or at home, you are quite safe. If it is secured in an airport departure lounge, you are probably safe. At the other extreme, any non-trivial use of an unknown computer in an arbitrary location in a strange country will remain risky for the foreseeable future.

To be honest the analogy with an ATM is not quite accurate. A large part of your confidence in the ATM comes from the limitation of your liability (at least, under U.S. law – it is much less limited in most other countries). At present there is no such limitation on what could happen with a subverted computer kiosk. Further, the theft of data can be much more difficult to detect than the theft of money. We see no prospect of this situation improving.

If you are using Soulpad, or DoK, another threat is theft of your USB device. Fortunately, this at least is amenable to straightforward cryptographic solutions. For example, you can buy today USB flash disks with on-board encryption hardware [11]. The data in these devices is encrypted before being written to the flash chip. The

encryption key is provided by you dynamically (by typing it into the local computer and having the computer pass it to the USB device) after you have assured yourself of the (sufficient) safety of the local computer. If you lose such a device your only worry is the missing data, not the risk of it being stolen.

Despite the above similarities of the security risks incurred with these systems, the trusted computing base (TCB) for these systems differs. All of them trust the local computer's hardware. Soulpad is close to a minimal TCB, since it boots the entire system from the USB device. Nevertheless, Soulpad still relies that this boot process is being undertaken by a trustworthy BIOS. DoK relies on the local computer's operating system and copy of VirtualPC. This could be ameliorated by using a copy of VirtualPC from the USB device. We could of course use the same Knoppix-based approach as Soulpad, with the same penalty in start-up time and the same dependency on having appropriate drivers. ISR depends heavily on trusting the local computer's operating system, as do web-based applications and thin clients. Thin clients have the advantage that the local operating system can be reduced to only what is required for the client protocol (as with Sun Ray), and similarly AJAX style web-based applications need only enough operating system to run a full function web browser (plus the browser itself)

9 Conclusions

The increasing ubiquity of personal computers and Internet access leads to an increasing problem in accessing your personal desktop environment: system, customizations, preferences and documents.

There is a wide range of potential solutions to this problem, each with its own set of trade-offs. Thin clients work very well in tightly coupled low latency environments, but are generally unacceptable if wider area use. Web-based applications using technology such as AJAX work well in most situations, but depend critically on building from scratch the appropriate applications. It is also not clear how widely applicable (in terms of application coverage) the web-based solution is. Solutions that rely entirely on migrating your state across the Internet perform well only when connectivity bandwidth is very good, or when it is easy to predict where the state will be needed. Solutions that take advantage of small portable USB mass storage devices seem to form a good compromise, leveraging the rapidly increasing capacity of such devices.

Across all of these solutions, there are security issues revolving around trust of the local computer. These are in general intractable without complete bottom-up security based on secrets embedded in hardware and a trusted certificate infrastructure, but in reality there are many situations where an acceptable level of trust can be achieved.

Overall, we believe that ubiquitous access of this form is arriving. Solutions based on carrying a mass storage device seem most attractive for pre-existing applications, whereas web-based solutions seem most attractive when suitable new applications can be constructed. Moreover, among the options for portable mass storage, it is important to consider not only the form factor of the portable device but also the relevance of using network resources to augment storage capacity.

References

1. BAPCO. An Overview of SYSmark 2004. http://www.bapco.com/techdocs/SYSmark2004WhitePaper.pdf.
2. A. Birrell, S. Perl, M. Schroeder, T. Wobber. Pachyderm: A Web-based Application for Email and News. Powerpoint slides, http://birrell.org/andrew/talks/pachyderm.pdf, 1997.
3. A. Birrell, M. Isard, C. Thacker, T. Wobber. A Design for High Performance Flash Disks, Technical Report, MSR-TR-2005-176, Microsoft Research, December 2005.
4. R. Cáceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with Portable SoulPads. In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, pp. 65-78.
5. Digital Equipment Corporation, Pachyderm, http://web.archive.org/web/19980123212517/http://www.research.digital.com/SRC/pachyderm/, 1997.
6. F. Douglis, R. Cáceres, F. Kaashoek, K. Li, B. Marsh and J. Tauber, Storage Alternatives for Mobile Computers, *In Proceedings of 1st Symposium on Operating Systems Design and Implementation (OSDI),* November 1994.
7. Google Inc. Gmail Home Page. http://www.gmail.com.
8. Google Inc. Google Maps Home Page. http://maps.google.com/maps.
9. J. Hollingsworth and E. Miller. Using content-derived names for configuration management. In Proceedings of the 1997 ACM Symposium on Software Reusability, Boston, May 1997.
10. KeyGhost Ltd. KeyGhost: Record Keystrokes in a Flash. http://www.keyghost.com/.
11. Kingston Technology. Data Traveller Elite. http://www.kingston.com/digitalmedia/dt_elite.asp.
12. J. Kistler, M. Satyanarayanan. Disconnected operation in the Coda file system. ACM Transactions on Computer Systems 10 (1) (1992).
13. Knoppix. Home page. http://www.knoppix.net.
14. M-Systems Inc. http://www.m-sys.com/site/en-US/Products/IDESCSIFFD/IDESCSIFFD/Products_/IDE_Products/FFD_25_Ultra_ATA.htm
15. Microsoft Corporation. Working Remotely with Windows XP. http://www.microsoft.com/windowsxp/using/mobility/default.mspx.
16. M. Oliver, R. Doraisamy, B. Doolittle, K. Peacock, G. Wall, G. Sloane. Sun Ray™ Deployment on Shared Networks. Sun Blueprints OnLine, http://www.sun.com/blueprints/0204/817-5490.pdf, February 2004.
17. S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In Proceedings of First USENIX conference on File and Storage Technologies, Monterey, CA, 2002.

18. RealVNC. Home page. http://www.realvnc.com.
19. M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10 (1) pp. 26-52.
20. Samsung Electronics. 512M x 8Bit / QG x 8 Bit NAND Flash Memory. K9W8G081M/K9K4G08U0M Flash Memory Datasheet.
21. C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI),* December 2002, pp. 377-390.
22. M. Satyanarayanan, M. Kozuch, C.Helfrich, D. O'Hallaron. Toward seamless mobility on pervasive hardware. Pervasive and Mobile Computing 1 (2005) 157-189, Elsevier.
23. Sun Microsystems. Sun Ray Overview (White Paper). http://www.sun.com/sunray/techinfo/New_SR_WP_12_04.pdf.
24. USB Implementers Forum. Universal Serial Bus Mass Storage Class Specification Overview, Revision 1.2. http://www.usb.org/developers/devclass_docs/usb_msc_overview_1.2.pdf, June 2003.
25. Wikipedia. Ajax (programming). http://en.wikipedia.org/wiki/AJAX.
26. Trusted Computing Group. TCG Specification Architecture Overview. Specification Revision 1.2, April 2004.
27. R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The Personal Server: Changing the Way We Think about Ubiquitous Computing. In *Proceedings of UbiComp 2002***,** pp. 194-209.
28. D. Woodhouse. JFFS: The Journalling Flash File System. Red Hat, Inc. http://sourceware.org/jffs2/jffs2-html/.