SLYK: A Transparent Fault-Tolerant Migration Platform

Jasper Lin, Jennifer Shu, Olivier Koch, Shuchyng You
{jasperln, jshu, olivierk, yoshu117}@mit.edu

Abstract

The recent trend towards mobile computing has introduced new challenges such as migrating a user's computing environment as he moves from location to location. Although laptops offer a great deal of mobility, they still suffer from traditional drawbacks, such as having weak computing power compared to desktops and being relatively expensive and encumbering. In the past few years, the concept of a virtual environment that can be suspended at one place and resumed at another has started to emerge, opening the door to true mobility. In such a world, a user would be able to work on his desktop at home in Texas, take a plane to California, and resume his work on any computer in LA exactly as it was when he left Texas.

We propose a virtual machine-based migration platform that preserves active network connections across machine migrations. To make our system fully deployable, we require neither cooperation from the outside world nor any modification to the host operating system. The platform provides machine and network transparency as well as fault tolerance in data integrity.

1 Introduction

In today's computing environment, it is common for one user to encounter several computers in the course of a day. Computers are increasingly being viewed as public utilities that are as ubiquitous as electricity and water, and as a result, users will no longer need to value computers as an expensive resource. Instead, users will place greater worth on their personal data stored on these computers.

Such mobility would be useful to virtually anyone using computers today. For instance, a student could have two workstations, one at home and one at the lab, and switch back and forth every day without having to carry a laptop with him. A businessman could travel across the United States and recover his latest work on any computer as long as he has Internet access. On top of making life easier, this concept of migration would make mobility more secure and less expensive.

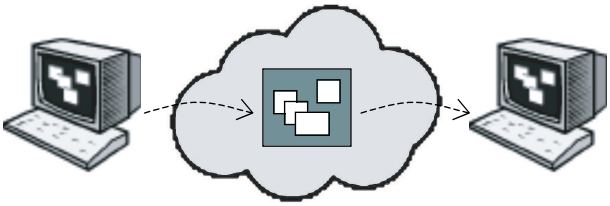

Figure 1: SLYK facilitates the migration of user state between network connected machines.

This novel view of computing, however, opens up a new batch of interesting challenges that need to be addressed before such a system can be successfully deployed. First, for maximum deployability reasons, no cooperation from the rest of the network should be required. In other words, if a user is migrating from one machine to another, the rest of the network should not be aware of it. We refer to this challenge as *network transparency*. Second, in order to offer the most flexibility, the migration platform needs to be hardware- and operating system independent. One promising approach is to use virtual machines which, by nature, do not depend on the underlying operating system and therefore allow true *machine transparency* [8]. Third, our system needs to provide a *fault-tolerant* approach for data storage.

Clearly, using the user machine hard-drive to store the virtual machine state is hazardous since the failure of the user machine would imply an unrecoverable loss of data. We propose to use a Distributed Hash Table (DHT) store acting as a virtual hard-drive (HD) to cache blocks of the memory image. We thus have to develop a communication scheme allowing to load and fetch memory blocks from the DHT. One extra advantage of such a scheme is that it offers a backup copy of the virtual machine that can be

used for recovery later on.

In this paper we focus on the problem of migrating user state as users traverse machines. This state can be packaged and sent over a network to resume execution on a different physical machine, as shown in Figure 1. Since the virtual machine simulates a complete architecture, users are permitted to run any operating system and application compatible with the emulated architecture.

We propose SLYK[1], a virtual machine-based migration platform that preserves active connections across machine migrations and offers a higher level of fault-tolerance. Based on QEMU [2], SLYK emulates an x86-based platform capable of running many major operating systems, including Linux, Windows, and FreeBSD. The state of any virtual system running on SLYK can be packaged up and sent over the network to be resumed by any other machine running SLYK.

2 Related Work

Previous work on using virtual machines to migrate state focuses mainly on optimizing the performance of emulation and the speed of migration [14]. Although performance is important for the mainstream adoption of virtual machines, there are other important factors such as fault-tolerance and transparent operation with the outside world.

Internet Suspend and Resume (ISR) [9] presents a straightforward implementation of a virtual machine migration infrastructure. Upon suspend, the state of the virtual machine is stored on a remote NFS server. When resumed, the state of the virtual machine is copied from NFS onto the target machine and the virtual machine is started.

Optimizing the Migration of Virtual Computers describes several optimizations to speed state migration time [14]. The goal is to make it practical to migrate state between home and work computers over a 384kbps link. The first optimization, called ballooning, involves requesting many memory pages and clearing them to zero to coerce the operating system into swapping out all but the most important memory pages immediately before migration. The memory is then compressed using gzip and sent to the target machine. Virtual HD blocks are left on the source machine to be requested as needed by the target machine.

[1]SLYK, pronounced "Slick", is taken from the authors' last names: Shu, Lin, You, and Koch.

This infrastructure suffers from two drawbacks. First, all active network connections are lost during migration. Applications that depend on these connectins need to be reset on the target machine. Second, HD blocks which have not been requested and cached locally may become inaccessible when their host machines go down.

Venti-DHash [15] is a peer-to-peer backup system based on the Venti archival storage system [13] with DHash running as the back-end storage. In this system, regular snapshots of filesystems are kept over the Internet by streaming active blocks into DHash, in the form of a Venti stream. It ensures that any unique block is stored only once. SLYK shares much of the same DHash infrastructure as Venti-DHash but relaxes many of the requirements of a complete archival system.

Mobile IP [12] provides mobility by always routing packets first to a static home host then to the mobile host, which works when a static host is always available and not separated by the network. However, failure of this home host results in loss of all active mobile connections.

There have been several proposals to migrate state at a finer granularity [3, 18, 20]. These systems exploit specific knowledge about the state or execution environment to ship the minimum amount of data needed for seamless transition. In contrast, virtual machine approaches involve potentially needing to send much more state than needed. However, the general approach adopted by virtual machine migration platforms allows the migration of many more operating systems and applications without any modification. Further, several optimizations can be performed to reduce the inherent overhead of migration via virtual machines [11, 14].

3 Challenges

There are many challenges to making a transparent, fault-tolerant migration platform. To provide full migration, we need to transfer a lot of state efficiently, including RAM and hard drive states. Unfortunately, the majority of state is kept on the hard drive, and it is both inefficient and unnecessary to transfer the entire hard drive during migration. Despite this, we still value personal state more than the actual hardware that it is running on. As a result, we would like to optimize hard drive migration, all the while keeping enough state for full recovery in the case of system failures.

One of the many things we aim for in SLYK is network transparency, so that no matter where a user migrates to, his active connections would still be intact. Network transparency would allow a SLYK user to, for example, continue downloading the same file when he moves from one SLYK machine to another. To achieve this goal, we need mechanisms like packet forwarding that is not dependent on one central routing client, unlike what has been done with mobileIP.

To make SLYK as deployable as possible, we want as little cooperation from the guest O/S or services as possible. In other words, we don't want to require any modifications on the part of the guest O/S or services in order to use SLYK. Transferring data from one host O/S to another requires dealing with different endians and data representations. SLYK needs to overcome these differences to be portable to all the commonly-used platforms like Linux, OS X, and Windows.

In the past, a lot of work has been done on migrating states without keeping the active connections (ISR), or optimizing performance with a trade-off in fault-tolerance. What distinguishes SLYK from previous projects is that it incorporates the properties that you would normally want in a migrated system, such as continuing connections and fault-tolerant demand paging, into one single platform.

4 System Overview

SLYK is a virtual machine-based platform that allows states to be migrated across three platforms (Linux, OS X, Windows), but yet, transparent to the network, to the host machine, and with fault-tolerance in data integrity. The platform-independent migration starts with a virtual machine emulating hardware states and keeps a record of all the CPU, timer states, and such. To allow connections to stay alive after migrating the virtual system, we implemented the additional capability within the virtual machine to communicate with one another using rpc's, without disrupting servers or other non-SLYK nodes on the network. Basically, it operates within its own virtual network environment. Holding a copy of the hardware, RAM, and connection states, the only other thing that could keep a SLYK machine from running as if it were a real physical machine is loosing parts of the hard drive image. To address this problem, we designed SLYK to run on a virtual hard drive demand-paged from a DHT store.

4.1 Machine Emulation

SLYK is built on top of QEMU [2], a highly portable open source emulator that supports both the x86 and PowerPC architectures. Based on dynamic translation, QEMU is a much faster processor emulator than Bochs [10], and is capable of either full system emulation, including a processor and various peripherals, or user mode emulation that allows Linux processes to be migrated across CPUs.

4.2 Remote Communication

XMLRPC++

4.3 Reliable Storage

To provide reliable storage, we chose to replicate the hard drive image on a DHT. The specific implementation of DHT that we use is OpenDHT, which runs on PlanetLab 24-7 with over 300 nodes available for answering put and get requests. OpenDHT supports communication through xmlrpc, and provides the reliable storage by replicating segments across its network of nodes. DHash, another implementation of the DHT service, also exhibits the fault-tolerant property that we aim for in SLYK. However, accessing services on OpenDHT does not require that you also become one of the nodes. In addition, because of a previous implementation choice on using xmlrpc for remote communication, OpenDHT undoubtedly becomes the better choice of the two.

4.4 Migration

State transfer amounts to serializing QEMU's memory representation, CPU and timer states, sending it over the network, unserializing the image on the other end, and bootstrapping SLYK with this memory image. However, suspending the virtual system while waiting for the large image to transfer may not be acceptable to the user, so SLYK follows these steps when transferring ownership to the new machine:

1. Begin transferring memory pages to the new machine.

2. Continue executing the virtual system on the old machine, marking any pages written to as dirty.

3. When the initial transfer of memory is done, suspend the virtual system and transfer only the dirty pages to the new machine.

4. Send a message to the new machine that indicates all pages have been transferred and it can gain ownership of the virtual system and resume operation.

In this scheme the virtual machine does need to suspend temporarily, but only while the dirty pages are being transferred before resuming operation.

5 Network Transparency

5.1 Virtual Network Environment

network environment. illusion behind NAT.

5.2 Active Connection Migration

Without cooperation from the remote machine, network connections cannot be fully migrated. There have been several proposals to augment internet routing with a truly mobile system such as i3 [16] and uip [7]. However, without the widespread deployment of such systems, and given the restriction that no modification can be done to the remote machines for maximal deployability, SLYK is forced to adopt a packet forwarding-like technique such as that used in MobileIP [12]. Unlike MobileIP, however, SLYK does not have a special home machine where all packets must first go through.

SLYK is optimized for the common case of the virtual system trying to establish a direct connection with a remote machine. In this situation, a direct connection is established between the host machine running SLYK and the remote machine, and packet rewriting tricks them into thinking they are communicating directly. It is only when there are lingering active connections at the time of migration that forwarding needs to take place.

When there are lingering active connections, the machine that the virtual system migrated from cannot fully shutdown SLYK. Instead, SLYK switches into a simple routing mode which keeps the connection alive with the remote machine and forwards any incoming packets to the new location of the virtual system. In this mode, SLYK also forwards packets from the virtual system to the remote machine. Figure 2b shows computer A switching into this routing mode since the connection to S1 was lingering at the time of migration.

This scheme implies that SLYK needs to maintain a table for all indirect remote machines the virtual system is actively in contact with. For example, in Figure 2b, computer A needs to take note that computer B is the current location of the virtual system and computer B needs to record that packets intended for the open connection to S1 first need to go through A. This bookkeeping is also important to update the routing machines when ownership is transferred. Figure 2c shows another migration, this time to C, which requires a message to be sent to A to update its route table and for C to remember the indirect route to both S1 and S2.

Two potential problems may arise from this forwarding scheme. First, a forwarding machine may go down. This would appear to the virtual system the same as if the remote machine dropped the connection. In many situations, the interested application or operating system may attempt to reestablish the connection and would succeed in directly connecting with the remote machine. Other times, the connection may not be so recoverable. These situations are unfortunate, but at least the loss of a forwarding machine would only result in dropping the indirect connections through that machine, versus the MobileIP scheme where losing the home machine would result in the loss of all indirect connections.

The second problem is that the routing tables may grow large. Entries in the table are cleared as connections close. The hope is that most TCP/IP connections are not very long lived. In the worst case, SLYK can always break the most idle connections to make space. If an application attempts to reestablish this connection, it would be a direct connection this time, which would not grow the routing table.

6 Virtual HD Store

Using the local hard drive of the SLYK machine clearly makes the system error-prone. If one of the SLYK machines were to go down, the data stored on its HD would be desperately lost until the machine recovers. This loss of data could prevent the user from resuming his activities on a new machine. In addition, transferring the content of a full hard-drive over the network would make the migration untractable.

For these reasons, we provide a virtual hard drive accessible from all SLYK machines at any time. This hard drive is fully transparent, in a sense that from the point of view of the SLYK machine, everything happens as if the local hard drive was used. The read and write operations to the local hard drive are simply intercepted and converted into read and write operations to the virtual hard drive. The virtual mem-

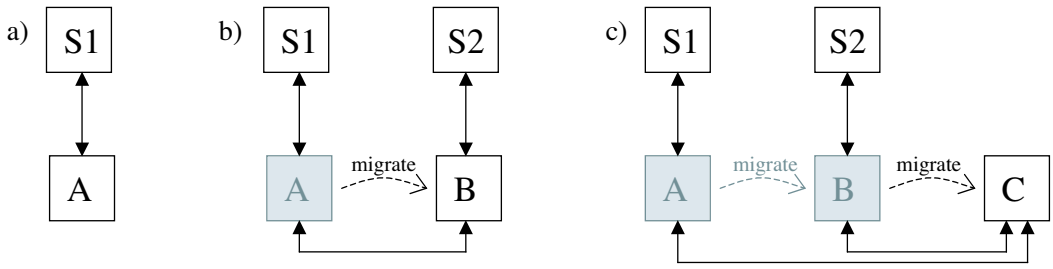

Figure 2: SLYK preserves active network connections through machine migration. (a) Before any migration occurs, machine A connects to sender S1. (b) After migration to machine B, the connection to S1 is indirectly maintained via forwarding through A. B also establishes a direct connection with sender S2. (c) Finally, when migrated to C active connections to S1 and S2 are maintained via forwarding though A and B, respectively. As active connections close these forwarding routes can be cleaned up.

ory, however, is transferred over the network during migration.

The virtual HD is stored on a DHT store such as DHash [5] running on top of Chord [17]. Since the virtual system runs only on one computer at a time, no complex coherency protocol between SLYK and the DHT is needed. The first time a block is fetched, SLYK caches it in its local store. On writes, SLYK marks the block as dirty and eventually flushes the block out to the DHT.

We use OpenDHT ([19]), a publicly accessible DHT service running on 200 widely distributed hosts provided by PlanetLab ([1]). OpenDHT offers a very transparent way to store and access ¡key,values¿ pairs. There is a 1024-byte limit on values in OpenDHT, therefore data has to be broken into blocks before being stored. Since QEMU's data storage is based on 512-byte disk sectors, we use these sectors as values in OpenDHT. It is important to note that data cannot be changed or removed once it has been stored on OpenDHT. Instead, a timeout mechanism makes the data obsolete on the DHT after a certain amount of time. The timeout value is specified by the user and should be chosen carefully. Using a too small timeout may result in loss of data. Using a too large value may conflict with PlanetLab requirements. However, the operability of SLYK is not impacted by this choice as long as a reasonable value is taken. Our implementation uses a timeout of 10 minutes.

We use Secure Hash Algorithm 1 (SHA1 [6]) for the key computation. SHA1 belongs to the family of MD4 hash functions. It takes as input a message of less than 2.64 bits and produces a 20-byte message digest. The code for SHA1 is delivered under the GNU license and follow the specifications defined in [4].

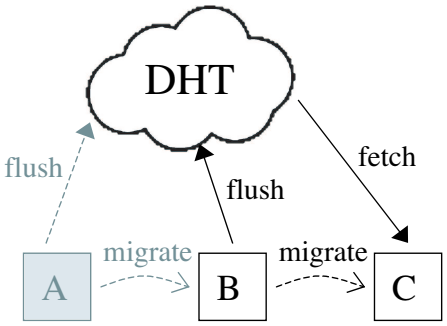

Figure 3: For fault-tolerance and fast migration, SLYK stores virtual HD blocks on an external DHT store. HD blocks are paged in on demand and cached locally. Dirty blocks are periodically written out to keep the DHT updated. Before migration, the final dirty blocks are flushed to the DHT.

The DHT should be kept as updated as possible; otherwise, flushing dirty blocks out to the DHT may become part of the critical path in migration. SLYK should not be too eager to flush blocks either, since batching writes to the same block saves bandwidth.

The approach adopted by SLYK is to prioritize dirty blocks by least recently written and flush those blocks first. The idea is that some blocks are going to be more actively written to than others. The ones actively being written to should be the last to be flushed since there is a good chance that deferring them will save bandwidth. With this priority in mind, SLYK continuously in the background flushes dirty blocks to the DHT, but it throttles itself to keep from using too much bandwidth. Only when migration is imminent does SLYK use its full bandwidth to fully flush out dirty pages.

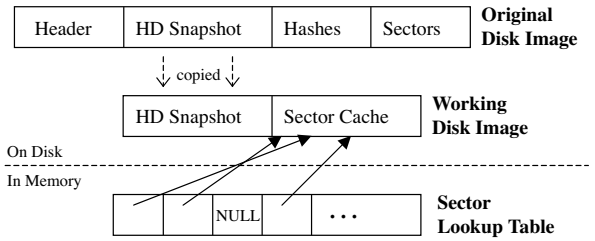

Figure 4: Disk structures...

Figure 3 shows the migration process from B to C. Before control is transferred to C, B flushes the remainder of its dirty blocks to the DHT. Then as C runs, it demands pages in blocks from the DHT to cache locally. Computer A shown in gray has already flushed its dirty blocks.

6.1 File Format

(figure)

6.2 DHT

6.3 Cache

(figure)

7 Backup

To prevent failure of the current machine from causing permanent loss of the virtual system, the memory image is periodically backed up onto the DHT. This copying can be very costly in terms of bandwidth if done too frequently.

There may be a few ways to speed this up besides scaling back the frequency of backups. For very low or no bandwidth situations, the memory can be backed up to the physical HD instead. In addition, memory backup might be able to piggy-back on the migration process. After control is transferred to the new machine, the old machine will still have a very recent copy of the virtual system's memory. Instead of immediately discarding this data, the old machine can contact the DHT and back up this data without directly consuming bandwidth from the user who is now on the new machine.

Table 1: Migration times...just a sample table

O/S	Linux	Mac OS X	Windows
Debian	245	61	75
Win2k	123	124	89
Knoppix	110	183	193
Blah	164	223	213

8 Results

9 Future Work

10 Conclusion

SLYK attempts to offer strong notions of mobility including machine transparency, network transparency, and active connection migration. It does so at a performance cost in order to ensure compatibility with as many operating systems, applications, and computing environments as possible. Our prototype demonstrates that despite this overhead a working, usable system can be built that offers the illusion of full mobility.

Many optimizations can be performed to bring down the cost of mobility. Some such as dynamic binary translation, gzip compression, and ballooning have already been explored. Others may benefit greatly from hardware, operating system, and network support. The costs associated with mobility will continue to decrease until one day full user mobility may become a normal part of our computing environment.

References

[1] M. Beck, T. Moore, and J. S. Plank. An End-to-End Approach to Globally Scalable Network Storage. Technical Report PDN–02–007, PlanetLab Consortium, November 2002.

[2] F. Bellard. Qemu CPU emulator User Documentation, 2003.

[3] F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: generative migration of mobile agents in heterogeneous environments. In *SAC*, pages 101–106, 2002.

[4] R. Brown. Secure hask standard. Federal Information Processing Standards Publication 180-1, April 1995.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[6] D. Eastlake. Us secure hash algorithm 1 (sha1). The Internet Society: RFC 3174, September 2001.

[7] B. Ford. Unmanaged Internet Protocol.

[8] IBM Corporation. *IBM virtual machine facility/370: planning guide Publication Number GC20-1801-0*, 1972.

[9] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 40, Washington, DC, USA, 2002. IEEE Computer Society.

[10] K. Lawton, B. Denney, N. D. Guarneri, V. Ruppert, C. Bothamy, and M. Calabrese. Bochs x86 PC emulator Users Manual, 2003.

[11] J. Nieh and O. C. Leonard. Examining VMware. *j-DDJ*, 25(8):70, 72–74, 76, Aug. 2000.

[12] C. E. Perkins. IP Mobility Support for IPv4. Internet Engineering Task Force: RFC 3344, August 2002.

[13] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *First USENIX conference on File and Storage Technologies*, Monterey,CA, 2002.

[14] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[15] E. Sit, J. Cates, and R. Cox. A DHT-based Backup System, 2003.

[16] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure, 2002.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[18] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, R. Jeffers, T. S. Mitrovich, B. R. Pouliot, and D. S. Smith. NOMADS: Toward a Strong and Safe Mobile Agent System. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 163–164, Barcelona, Catalonia, Spain, 2000. ACM Press.

[19] B. K. Sylvia. Spurring adoption of dhts with openhash, a public dht service.

[20] T. Walsh, P. Nixon, and S. Dobson. As strong as possible mobility: An Architecture for stateful object migration on the Internet.