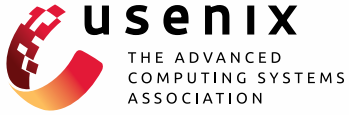

XvMotion: Unified Virtual Machine Migration over Long Distance

Ali José Mashtizadeh, *Stanford University;* Min Cai, Gabriel Tarasuk-Levin, and Ricardo Koller, *VMware, Inc.;* Tal Garfinkel; Sreekanth Setty, *VMware, Inc.*

https://www.usenix.org/conference/atc14/technical-sessions/presentation/mashtizadeh

This paper is included in the Proceedings of USENIX ATC '14: 2014 USENIX Annual Technical Conference.

June 19–20, 2014 • Philadelphia, PA

978-1-931971-10-2

XvMotion: Unified Virtual Machine Migration over Long Distance

Ali José Mashtizadeh[⋆]*, Min Cai, Gabriel Tarasuk-Levin*
Ricardo Koller, Tal Garfinkel[†]*, Sreekanth Setty*
Stanford University[⋆] *– VMware, Inc. – Unaffiliated*[†]

Abstract

Live virtual machine migration allows the movement of a running VM from one physical host to another with negligible disruption in service. This enables many compelling features including zero downtime hardware upgrades, dynamic resource management, and test to production service migration.

Historically, live migration worked only between machines that shared a common local subnet and storage system. As network speed and flexibility has increased and virtualization has become more pervasive, wide area migration is increasingly viable and compelling. Ad-hoc solutions for wide area migration have been built, combining existing mechanisms for memory migration with techniques for sharing storage including network file systems, proprietary storage array replication or software replicated block devices. Unfortunately, these solutions are complex, inflexible, unreliable and perform poorly compared to local migration, thus are rarely deployed.

We have built and deployed a live migration system called XvMotion that overcomes these limitations. XvMotion integrates support for memory and storage migration over the local and wide area. It is robust to the variable storage and network performance encountered when migrating long distances across heterogeneous systems, while yielding reliability, migration times and downtimes similar to local migration. Our system has been in active use by customers for over a year within metro area networks.

1 Introduction

Live virtual machine (VM) migration moves a running VM between two physical hosts with as much transparency as possible: negligible downtime, minimal impact on workload, and no disruption of network connectivity.

Originally, VM migration only moved a VM's memory and device state between two closely related physical hosts within a cluster i.e., hosts that shared a common storage device, usually a storage array accessed via a dedicated SAN, and a common Ethernet subnet to enable network mobility. Assumptions about host locality made sense given the limited scale of VM adoption, and limitations of storage and network devices of the past.

Today, many of these assumptions no longer hold. Data center networks are much faster with 10 Gbps Ethernet being common, and 40/100 Gbps adoption on the way. Tunneling and network virtualization technologies [14, 20, 22] are alleviating network mobility limitations. Different workloads and high performance SSDs have made local shared-nothing storage architectures increasingly common. Finally, VM deployments at the scale of tens of thousands of physical hosts across multiple sites are being seen. Consequently, the ability to migrate across clusters and data centers is increasingly viable and compelling.

Some have attempted to build wide area live migration by combining techniques to migrate storage e.g., copying across additional storage elements, proprietary storage array capabilities, or software storage replication in conjunction with existing technologies for migrating memory and device state. These ad-hoc solutions are often complex and fragile—incurring substantial penalties in terms of performance, reliability, and configuration complexity compared with doing migration locally. Consequently, these systems are deployed at a very limited scale and only by the most adventuresome users.

We present XvMotion, an integrated memory and storage migration system that does end-to-end migration between two physical hosts over the local or wide area. XvMotion is simpler, more resilient to variations in network and storage performance, and more robust against failures than current ad-hoc approaches combining different storage replication and memory migration solutions, while offering performance, i.e., workload impact and service downtimes, comparable to that of a local migration.

XvMotion has been in use with customers for over a year, and seen wide spread application at the data center and metro-area scale, with link latencies in the tens of milliseconds. Our results with moving VMs between Palo Alto, California and Banglore, India demonstrate that migrations on links with latencies in the hundreds of milliseconds are practical.

We begin by examining how current local migration architectures developed and survey ad-hoc approaches used to support wide area migration. Next, we explore our design and implementation of XvMotion: our bulk transport layer, our approach to asynchronous storage mirroring, our integration of memory and storage movement, our workload throttling mechanism to control page dirtying rates—allowing migration across lower bandwidth links, and our disk buffer congestion control mechanism to support migration across hosts with heterogeneous storage performance. We then present optimizations to minimize switch over time between the old (pre-migration) and new (post-migration) VM instance, cope with high latency and virtualized networks, and reduce virtual disk copy overhead. Our evaluation explores our local and long distance

migration performance, and contrasts it with existing local live storage migration technologies on shared storage. After this, we discuss preventing split-brain situations, network virtualization, and security. We close with related work and conclusions.

2 The Path to Wide Area Migration

Wide area live migration enables a variety of compelling new use cases including whole data center upgrades, cluster level failure recovery (e.g. a hardware failure occurs, VM's are migrated to a secondary cluster and then migrated back when the failure has been fixed), government mandated disaster preparedness testing, disaster avoidance, large scale distributed resource management, and test to production data center migration. It also allows traditional mainstays of VM migration, like seamless hardware upgrades and load balancing, to be applied in data centers with shared-nothing storage architectures.

Unfortunately, deploying live migration beyond a single cluster today is complex, tedious and fragile. To appreciate why current systems suffer these limitations, we begin by exploring why live migration has historically been limited to a single cluster with a common subnet and shared storage, and how others have tried to generalize live migration to the wide area. This sets the stage for our discussion of XvMotion in the next section.

Live memory and device state migration, introduced in 2003, assumed the presence of shared storage to provide a VM access to its disks independent of what physical host it ran on and a shared subnet to provide network mobility. In 2007, live storage migration was introduced, allowing live migration of virtual disks from one storage volume to another assuming that both volumes were accessible on a given host. This provided support for storage upgrades and storage resource management. However, the ability to move a VM between two arbitrary hosts without shared storage has largely been limited to a few research systems [9,28] or through ad-hoc solutions discussed later in this section.

Why not migrate an entire VM, including memory and storage, between two arbitrary hosts over the network? Historically, hardware limitations and usage patterns explain why. Data centers of the previous decades used a mix of 100 Mbps and gigabit Ethernet, with lower cross sectional bandwidths in switches than today's data centers. High performance SAN based storage was already needed to meet the voracious IO demands induced by consolidating multiple heavy enterprise workloads on a single physical host. Network mobility was difficult and complex, necessitating the use of a single shared subnet for VM mobility. Finally, common customer installations were of modest size, and live migration was used primarily for hardware upgrades and load balancing, where limiting mobility to a collection of hosts sharing storage was an acceptable constraint.

Many of these historical limitations no longer apply. Data center networks are much faster with 10 Gbps Ethernet being common, and 40/100 Gbps adoption on the way, with a correspond growth in switch capacity. The introduction of tunneling and network virtualization technologies, like Cisco OTV [14], VXLAN [20] and OpenFlow [22], are alleviating the networking limitations that previously prevented moving VMs across Ethernet segments. With changes in workloads and the increased performance afforded by SSDs, the use of local shared-nothing storage is increasingly common. Finally, with users deploying larger scale installations of thousands or tens of thousands of physical hosts across multiple sites, the capacity to migrate across clusters and data centers becomes increasingly compelling.

To operate in both the local and wide area, two primary challenges must be addressed. First, existing live migration mechanisms for memory and device state must be adapted to work in the wide area. To illustrate why, consider our experience adapting this mechanism in ESX.

In local area networks, throughputs of 10 Gbps are common, as our migration system evolved to support heavier and larger workloads we designed optimizations and tuned the system to fit local area networks. In contrast, metro area links are in the range of a few gigabits at most, and as distances increase in the wide area throughputs typically drop below 1 Gbps with substantial increases in latency. On initial runs on the wide area our local live migration system didn't fare well; we often saw downtimes of 20 seconds or more causing service interrupting failures in many workloads, migrations were frequently unable to complete, and network bandwidth was underutilized. To understand why, lets briefly recap how live memory and device state migration work.

Production live migration architectures for memory and device state all follow a similar iterative copy approach first described by Nelson *et al.* [24]. We initially mark all pages of the VM as dirty, and begin to iterate through memory, copying pages from source to destination. After a page is copied, we install a write trap and mark the page as clean. If the write trap is triggered, we again mark the page as dirty. We apply successive "iterative precopy" passes, each pass copying remaining dirty pages. When the remaining set of pages to be copied is small enough to be copied without excessive downtime (called convergence), we suspend VM execution, and the remaining dirty pages are sent to the destination, along with the device state. Finally, we resume our now fully consistent VM on the destination and kill our source VM.

Unfortunately, naively applied, this approach does not behave well over slower and higher latency networks. Workloads can easily exceed the memory copy throughput, preventing the migration from every converging and causing large downtimes. To cope with this, we introduced workload throttling (§ 3.5) to limit the downtime.

Compounding that issue our initial local migration system used a single TCP connection as transport that suffers from head-of-line blocking and underutilizes network throughput. An improved transport mechanism(§ 3.2) and TCP tuning(§ 4.3) were required to fully utilize high bandwidth-delay links.

The other major challenge to long distance migration is how to move storage. Today, users with the desire to migrate VMs within or across data centers rely on three different classes of ad-hoc approaches: sharing storage using a network file system as a temporary "scratch" space for copying the VM, proprietary storage array based replication, and software based replication (e.g., DRBD) done outside the virtualization stack. Each approaches exhibits particular limitations, understanding these helps to motivate our approach.

In the first approach, a user exploits a network file system (e.g., NFS or iSCSI) to provide a temporary scratch space between two hosts, first doing a storage migration to this scratch space from the source host, then doing a second storage migration from the scratch space to the destination host. A live memory migration is done in between to move memory. This approach has several unfortunate caveats.

Moving data twice doubles total migration time—the performance impact is generally worse than this as the storage migration is not coordinated with memory and device state movement. Running the VM temporarily over the WAN while its disk is located on the scratch volume further penalizes the workload. Finally, hop-by-hop approaches are not atomic, if network connectivity fails, the VMs disk could end up on one side of the partition, and its memory on the other—this state is unrecoverable and the VM must be powered off.

Seeing the limitations of this approach, storage vendors stepped in with proprietary solutions for long distance storage migrations, such as EMC VPLEX [8]. These solutions use synchronous and asynchronous replication applied at a LUN level to replicate virtual disks across data centers. These solutions typically switch from asynchronous to synchronous replication during migration to ensure an up to date copy of the VM's storage is available in both data centers to eliminate the risk that a network partition will crash the VM, unfortunately, this imposes a substantial penalty on workloads during migration. These approaches are not cross-host migration as they do not support shared nothing storage configurations, instead these migrations are between two compatible storage arrays in different data centers.

The last approach relies on software replication, such as DRBD [2], to create a replicated storage volume. DRBD is a software replication solution, where typically each disk is backed by a DRBD virtual volume. To migrate one would back the VM by a DRBD volume, then replicate it to the desired destination machine. Next, a live migration would migrate the VM's memory/execution to the destination. Once complete the DRBD master is switched to the destination and the replication is terminated. A case study of a DRBD deployment [25] uses asynchronous replication during the bulk disk copy and synchronous storage replication for the duration of the migration, as noted above, this synchronous replication phase imposes an additional overhead on the VM workload. Another issue encountered with this approach is the lack of atomicity of the migration, i.e., the storage replication and memory migration system must simultaneously synchronize and transfer ownership from source to destination. Otherwise, this extends the window for a network partition causing the VM to power-off or be damaged.

3 XvMotion

XvMotion provides live migration between two arbitrary end hosts over the local or wide area. Many of limitations of previous approaches are eliminated by being a purely point-to-point solution, without the need for support from a storage system or intermediate nodes.

Our primary goals are to provide unified live migration (memory and storage), with the critical characteristics of local migration—atomic switch over, negligible downtime, and minimal impact on the VM workload, that can operate in the local and wide area.

We present XvMotion as follows. We begin with an overview of the XvMotion architecture in § 3.1. Next, we examine our bulk transport layer *Streams* in § 3.2. In § 3.3 we explore our implementation of asynchronous storage mirroring and storage deduplication. The coordination of memory and storage copying is described in § 3.4. § 3.5 presents SDPS, a throttling mechanism used to limit a VM's page dirtying rate to adapt to available network bandwidth and ensure that a migration converges. § 3.6 describes disk buffer congestion control, which allows XvMotion to adapt to performance differences between disks on the source and destination hosts.

3.1 Architecture Overview

XvMotion builds on the live migration [24] and IO Mirroring [21] mechanisms in ESX. Live memory migration is implemented using an iterative copy approach described in the previous section. ESX uses synchronous mirroring to migrate virtual disks from one volume to another on the same physical host. We augment this with asynchronous IO mirroring for migrating virtual disks across hosts while hiding latency.

Figure 1 depicts the XvMotion architecture. The *Streams* transport handles bulk data transfers between the two hosts. The *Live Migration* module is responsible for the many well-known tasks of VM live memory migration, such as locating VM memory pages for transmission, and appropriate handling of the VM's virtual device state. It enqueues its pages and relevant device state for transmission by *Streams*.

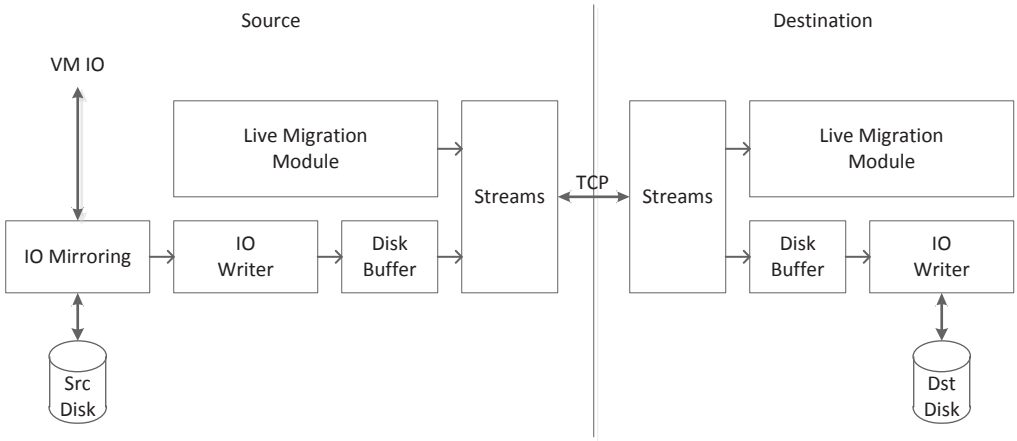

Figure 1: Architecture Overview: shows the main components involved in cross host migration for both the source and destination. IO Mirroring pushes data synchronously into the disk buffer through the migration IO Writer interface. The streams transport is responsible for bulk transfer of data between the hosts. Finally, on the destination the data is written to the disk.

The *IO Mirroring* module interposes on all virtual disk writes and mirrors these to the XvMotion *IO Writer* module. The *IO Mirroring* module is also responsible for the bulk reading of storage blocks during the initial copy i.e., synchronizing the mirror. The IO Writer module on the destination is responsible for dequeuing and writing blocks to the destination virtual disk.

The IO Writer then enqueues its data for the Disk Buffer. Disk buffering enables for asynchronous IO mirroring by storing a copy in RAM until it reaches the destination. While buffer space is available disk mirroring will acknowledge all mirror writes immediately, which eliminates the performance impact of network latency.

3.2 Streams Transport Framework

Streams is a bulk transport protocol we built on top of TCP for transferring memory pages and disk blocks. Streams creates multiple TCP connections between the hosts, including support for multipathing across NICs, IP addresses, and routes. Additional discussion of how Streams uses TCP is given in § 4.1 and § 4.3.

Using multiple connections mitigates head of line blocking issues that occur with TCP, leading to better utilization of lossy connections in the wide area [16, 17]. In the local area, the multiple independent connections are used to spread the workload across cores to saturate up to 40 Gbps links.

Streams dynamically load-balances outgoing buffers over the TCP connections, servicing each connection in round-robin order as long as the socket has free space. This allows us to saturate any number of network connections, up to PCI bus limitations. As a consequence of this approach, data is delivered out-of-order. However, there are no ordering requirements within an iteration of a memory copy: we just need to copy all pages once in any order. Any ordering requirement is expressed using a *write barrier*. For example, we require write barriers between memory pre-copy iterations.

Write barriers are implemented in the following way. Upon encountering a barrier request, the source host transmits a barrier message over all communication channels to the destination host. The destination host will read data on each channel up to the barrier message, then pause until all channels have reached the barrier message. Barriers are an abstraction provided to memory migration and storage migration, but do not necessarily prevent Streams from saturating the network while waiting for all connections to reach the barrier.

The general lack of ordering in Streams impacts our storage migration design. As we see in the next section, barriers are important as ordering is sometimes necessary for correctness.

3.3 Asynchronous IO Mirroring

Storage migration works by performing a single pass, bulk copy of the disk, called the *clone process*, from the source host to the destination host. Concurrently, IO mirroring reflects any additional changes to the source disk that occur during the clone process to the destination. When the clone process completes, the source and destination disks are identical [21]. Both clone and mirroring IOs are made asynchronously by using the Disk Buffer and Streams framework. This minimizes the performance impact on our source VM, despite higher and unpredictable latencies in long distance migrations.

Our clone process is implemented using a kernel thread that reads from the source disk and issues writes to the destination. This clone thread proceeds linearly across the disk, copying one disk region at a time. As usual, the remote write IOs are first enqueued to disk transmit buffer, then transferred to the destination host by the Streams framework, and then written to disk on the destination.

The clone process operates independently of the IO Mirror. This introduces a potential complication. A VM could issue a write while a read operation in the clone process is in progress, thus, the clone process could read an

outdated version of the disk block. To avoid this, the IO Mirror module implements a synchronization mechanism to prevent clone process IOs and VM IOs from conflicting.

Synchronization works by classifying all VM writes into three types relative to our clone process: (1) writes to a region that has been copied (2) writes to a region being copied (3) writes to a region that has not yet been copied. No such synchronization is required for reads, which are always passed directly to the source disk.

For case (1), where we are writing to an already-copied region, any write must be mirrored to the destination to keep the disks in lock-step. Conversely, in case (3), where we are writing to a region that is still scheduled to be copied, the write does not need to be mirrored. The clone process will eventually copy the updated content when it reaches the write's target disk region.

More complex is case (2), where we receive a write to the region currently being copied by the clone process. In the case of local storage migration, where writes are synchronously mirrored, we defer the conflicting writes from the VM and place them into a queue. Once the clone process completes for that region, we issue the queued writes and unlock the region. Finally, we wait for in-flight IOs as we lock the next region, guaranteeing there are no active writes to our next locked region [21]. However, for long distance migration, network latency makes it prohibitively expensive to defer the conflicting VM IOs until the bulk copy IOs are complete. Instead, we introduce the concept of a *transmit snapshot*, a sequence of disk IOs captured with a sliding window in the transmit buffer.

As we receive IO, either clone or mirror IO, we fill the disk's active *transmit snapshot*. When the active *transmit snapshot* reaches capacity (1 MB by default), the snapshot is promoted to a *finalized snapshot*. This promotion process pushes the *transmit snapshot* window forward to cover the next region of the disk's transmit buffer, the new empty *transmit snapshot*. As soon as all of our clone process's IOs for a given region have been queued in a *transmit snapshot*, we unlock the copy region and move on to the next region.

The Streams framework searches for *finalized snapshots*, transmitting them in the order they are finalized. A write barrier is imposed between the transmission of each *finalized snapshot*, ensuring that snapshot content is not interleaved. Between these write barriers, source-side snapshot sector deduplication, and destination-side IO conflict chains, we ensure that all IOs are written to the destination in the correct order. See § 3.6 for more discussion of snapshot handling and IO conflict chains.

We perform source-side deduplication by coalescing repeated writes to the same disk block. This ensures correctness as blocks within a *transmit snapshot* may be reordered by Streams, and reduces bandwidth consumption. Upon receipt of new IO, we search our transmit snapshot for any IO to the same target disk sector. If there was a previous write, we coalesce the two into a single write of the most recent data. This is safe, as we know clone process IO and mirror IO will never conflict, as explained in our discussion of synchronization.

Scanning our transmit snapshot for duplicate disk IO can be expensive, as each disk's active snapshot could contain 1 MB worth of disk IO. We avoid such lookups with a bloom filter that tracks the disk offsets associated with each sector present in each disk's active snapshot. With 8 KB of memory dedicated to each disk's bloom filter, the disk sector as the key, and 8 hashes per key, we achieve a false positive rate of less than 6 per million blocks.

Our data showed that the use of the bloom filter and this optimization halved migration time and cut CPU usage by a factor of eight in some cases. This result is highly workload dependent, and the CPU utilization benefits of the bloom filter offer a large portion of the gains.

3.4 Memory and Disk Copy Coordination

There are several copying tasks for us to coordinate: copying our disk, IO mirroring, copying of the initial memory image, and iterative copying of memory data.

When we start and how we interleave these processes can have a significant impact. If we have insufficient bandwidth available to keep up with page dirtying, we may be forced to throttle the workload. If we do not efficiently use available bandwidth, we increase overall migration time. While low intensity workloads with relatively low dirty rates do not create significant contention for bandwidth, contention can become an issue with higher intensity workloads.

We begin our disk copy first as this is usually the bulk of our data, while enabling IO mirroring. For a while this "fills-the-pipe." After our clone process completes, we begin our initial memory copy, then begin our iterative memory copy process with IO Mirroring still enabled.

Our rationale for this is that storage mirroring generally requires lower bandwidth than memory copying, so overlapping these is less likely to lead to contention than if we try to perform our disk copy process while memory is being copied. Also, our disk copy generally takes longer to complete, thus, we prefer to start it first, to minimize the overall migration time.

Our intuition around memory tending to show an equal or higher dirtying rate, i.e., rate at which pages or disk blocks change, leading to a higher cost for our iterative copy than for IO mirroring, comes from our observation of databases workloads and other enterprise applications. Locks, metadata and other non-persistent structures appear to be a significant source of dirtying.

During the switchover time, where we switch from source to destination, we pause the VM for a short period to copy any remaining dirty memory pages and drain the storage buffers to disk. Draining time is a downside

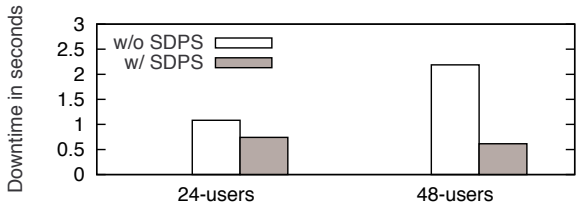

Figure 2: Shows downtime for two intensities of our DVD Store 2 workloads with and without SDPS. We show less downtime variation with SDPS than without.

of asynchronous mirroring as it impacts switchover time, which is not an issue for local synchronous mirroring.

While our heuristic approach seems to work well it may not be ideal. In an ideal implementation we would like to start the memory copy just before the disk copy has completed and only use unutilized bandwidth. Unfortunately, without predictable bandwidth it is hard to decide when we should begin the memory copy as bandwidth may suddenly become scarce, we could be forced to throttle our workload. We imagine using underlying network QoS mechanisms or better techniques for bandwidth estimation could be helpful.

3.5 Stun During Page Send (SDPS)

Transferring the VM's memory state is done using iterative pre-copy, as discussed in § 3.1. Iterative pre-copy converges when a VM's workload is changing page content (dirtying pages) more slowly than our transmission rate. Convergence means that the migration can tradeoff longer migration time for ever smaller downtimes, and thus the migration can be completed without noticeable downtime. Unfortunately, this is not always the case, especially in long distance migrations.

Historically, live migration implementations have solved the problem of low bandwidth by stopping pre-copy, halting VM execution, and transmitting all remaining page content during VM downtime. This is not desirable, as the VM could remain halted for a long time if there are many dirty pages, or if the network is slow.

To solve this problem, we introduce Stun During Page Send (SDPS). SDPS monitors the VM's dirty rate, as well as the network transmit rate. If it detects that the VM's dirty rate has exceeded our network transmit rate, such that pre-copy will not converge, SDPS will inject microsecond delays into the execution of the VM's VCPUs. We impose delays in response to page writes, directly throttling the page dirty rate of any VCPU without disrupting the ability of the VCPU to service critical operations like interrupts. This allows XvMotion to throttle the page dirtying rate to a desired level, thus guaranteeing pre-copy convergence.

Figure 2 shows the downtime for our DVD Store 2 workload that is described in Section 5. We see less downtime variability when SDPS is turned on than without. This technique is critical to allowing live migration to support slower networks and handle intense workloads.

3.6 Disk Buffering Congestion Control

XvMotion supports migrating a VM with multiple virtual disks located on different source volumes to different destination volumes with potentially differing IO performance. For example, one virtual disk could be on a fast Fibre Channel volume destined for a slower NFS volume, and another on a slower local disk destined for a faster local SSD.

Disparities in performance introduce several challenges. For example, if we are moving a virtual disk from a faster source volume to a slower destination volume, and we have a single transmit queue shared with other virtual disks, the outgoing traffic for our slower destination can tie up our transmit queue, starving other destinations.

We address such cross-disk dependencies with queue fairness constraints. Each disk is allowed to queue a maximum of 16 MB worth of disk IO in the shared transmit queue. Attempts to queue additional IO are refused, and the IO is scheduled to be retried once the disk drops below the threshold. In effect, each virtual disk has its own virtual transmit queue.

Tracking per-disk queue length also allows us to solve the problem of the sender overrunning the receiver. To avoid sending too many concurrent IOs to the destination volume, we have a soft limit of 32 outstanding IOs (OIOs) per volume. If the number of OIOs on the destination exceeds this, the destination host can apply back pressure by requesting that the source host slow a given virtual disk's network transmit rate.

There is a final challenge the destination host must attend to. Since the source deduplicates the transmit snapshots by block number, we know that IOs within a given snapshot will never overlap, and can thus be issued in any order. However, there may be IO conflicts i.e., IOs to overlapping regions across snapshots. In such cases, it is important for correctness that IOs issue in the order that the snapshots are received. However, we don't want to limit performance by forcing all snapshots to be written in lockstep. Instead, we make the destination keep track of all in-flight IOs, at most 32 IOs for each disk. Any conflicting new IOs are queued until the in-flight IO is completed. This is implemented with a queue, called a conflict chain, that is associated with a given in-flight IO.

Fairness between clone and mirror IOs: We implement fairness between clone and mirror IOs in a similar way. Each disk's virtual queue is split into two, with one portion for the clone and the other for mirror IOs. These queues are drained in a weighted round-robin schedule. Without this, the guest workload and clone can severely impact one another when one is more intense than the other, possibly leading to migration failures.

4 Optimizations

4.1 Minimizing Switchover Time

The switchover time is the effective downtime experienced by the VM as we pause its execution context to move it between hosts. It is the phase we use to transfer the remaining dirty pages and the state of the guest's virtual devices (i.e., SVGA, keyboard, etc.). Today's applications can handle downtimes over 5 seconds, but an increasing number of high availability and media streaming applications can make sub-second downtimes noticeable. For these reasons, we implemented some optimizations to battle the following problems:

Large virtual device states: Some of the virtual devices can be very large, for example the SVGA (the video card buffer) can be hundreds of megabytes. The problem is that this state is not sent iteratively, so it has to be sent completely during downtime. Because of limited bandwidth at high latency, we implemented a solution where the state of the larger devices is stored as guest memory. Now, the state is sent iteratively, is subject to guest slowdown (i.e., SDPS), and, most importantly, is not sent in full during downtime.

TCP slow start: There are several TCP sockets used during downtime: some to transfer the state of the virtual devices, and some for the Streams transmission. The problem is that each time a TCP connection is used for the first time or after a delay, TCP is in slow start mode where it needs time to slowly open the windows before achieving full throughput (i.e., several hundreds of milliseconds). We implemented some TCP extensions where we set all the sockets windows to the last value seen before becoming idle. This change reduces many seconds of downtime at 100 ms of latency.

Synchronous RPCs: We found that our hot migration protocol had a total of 11 synchronous RPCs. At 100 ms round-trip time (RTT), this adds up to 1.1 seconds of downtime. After careful analysis, we concluded that most of these RPCs do not need to be synchronous. In fact, there are only 3 synchronization points needed during switchover: before and after sending the final set of dirty pages, and the final handshake that decides if the VM should run on the destination or the source (e.g., there was a failure).

We modified the hot migration protocol to only require these three round-trips. The idea was to make the RPCs asynchronous with the use of TCP's ordering and reliability guarantees. Specifically, we used the following 2 guarantees. (1) If the source sends a set of messages and the destination receives the last one, TCP ensures that it also received all the previous messages. (2) Moreover, if the source receives a reply to the last message, TCP ensures that the destination received all the previous messages. We added the three required synchronization points such that the source sends messages, but only waits for the reply of those three points.

4.2 Network Virtualization Challenges

To saturate a high latency TCP connection we must avoid inducing packet loss and reordering within the virtualized network stack. Due to network virtualization, a hypervisor's networking stack is complex, with multiple layers, memory heaps, and asynchronous contexts: all of them with their own queuing mechanisms and behavior when queues are full. When queue limits are exceeded, packet loss and reordering can occur, disrupting TCP streams.

We found that Virtual Network Switches on the source host were the main source of packet loss. Multiple XvMotion sockets with large buffers were constantly overflowing the virtual switches' queues, resulting in packet loss, which substantially impacted TCP performance. We tried adding Random Early Detection (RED [13]) to the Virtual Network Switch, which drops packets from the head of queues rather than the tail so the sender could detect packet loss sooner. However, we found that the resulting packet losses still significantly reduced throughput. The solution we settle upon was to allow virtual switches to generate back pressure when their queues were nearly full by sending an explicit notification to the sending socket(s). This allowed our sockets to check for possible overflows, stop growing the transmit windows, and ultimately avoid the drops.

A second problem is the reordering of packets. Packets are moved from a virtual switch queue into a NIC ring, and whenever the NIC is full, packets are re-queued into the virtual switch. We found this to be a common source of packet reordering. We noticed that a single packet reorder caused a performance hit from 113 Mbps to 76 Mbps for a 1 GB data transfer over a 100 ms RTT link. After fixing these problems, we can get full throughput at 1 Gbps and up to 250 ms of round-trip time. Tests also showed that we were able to achieve 45% of bandwidth utilization at a loss rate of 0.01%.

4.3 Tuning for Higher Latency

High latency networks require provisioning larger TCP send socket buffers. One restriction on our platform, however, is the strictness of resource management and the desire to minimize memory usage for any given VM migration. We satisfy both goals by detecting the round-trip-time between the source and destination host in the initial migration handshake, dynamically resizing our socket buffers to the bandwidth-delay-product.

In the course of experimenting with socket buffer resizing, we discovered some performance issues surrounding our congestion control algorithms. We switched to the CUBIC congestion control algorithm, which is designed for high bandwidth-delay product networks [15]. Experimentation showed that we should enable congestion control with Accurate Byte Counting [7]. This improved performance when leveraging various offload and packet coalescing algorithms. It helps grow the congestion window

faster and handles delayed acknowledgments better.

4.4 Virtual Disk Copy Optimizations

XvMotion implements several storage optimizations to remain on par with live storage migration between locally accessible volumes. Local live storage migration makes heavy use of a kernel storage offload engine called the Data Mover (DM) [21].

The DM is tightly coupled with the virtual disk format, file system, and storage array drivers to perform optimizations on data movement operations. The DM's primary value is its ability to offload copy operations directly to the storage array. However, it also implements a number of file system specific optimizations that are important for storage migration. Because the DM only works with local storage we cannot use the DM, instead, we reimplemented these optimizations in XvMotion.

Metadata Transaction Batching: In VMFS, metadata operations are very expensive as they take both in-memory and on-disk locks. To avoid per-block metadata transactions, we query the source file system to discover the state of the next 64 FS blocks, then use that data to batch requests to the destination file system for all metadata operations.

Skipping Zero Block Reads: Each file in VMFS is composed of file system blocks, by default 1MB each. A file's metadata tracks the allocation and zero state of those blocks. Blocks may not be allocated, may be known to be zero-filled, or allocated but still to-be-zeroed (TBZ) blocks. By querying the FS metadata, we can skip blocks that are in one of these zero states.

Skip-Zero Writes: On the destination, if we know we will write over the entire contents of a file system block, we can have the file system skip performing its typical zeroing upon the first IO to any FS block. We bias writing to entire FS blocks when possible to leverage such skip-zero opportunities.

5 Evaluation

Downtime: All live migration technologies strive to achieve minimal downtime i.e., no perceptible service disruption. Our results show that XvMotion delivers consistent downtimes of ≈ 1 second—what the convergence logic targets—in the face of variable round trip latency (0-200 ms) and workload intensity (1-16 OIOs), as depicted in Figures 3 and 4.

Migration time: Our results show that our migration times are stable with respect to latency up to 200 ms—demonstrating that wide area migration need not be any more expensive than local area migration. XvMotion shows only a small increase in migration time beyond the sum of local memory and storage migration time.

Guest Penalty: Guest penalty as a percentage of the reduction of guest performance (IOPS or Operations/sec.) during migration should be minimized. Our results show that guest penalty is nearly constant with respect to latency, and only varies based on workload intensity.

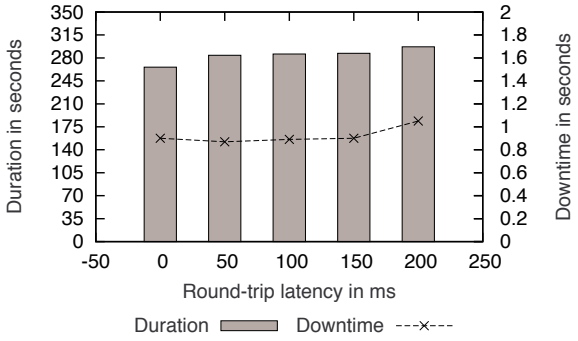

Figure 3: Shows the migration time and downtime for a VM running the OLTP workload at varying latency.

Our XvMotion tests were run on a pair of Dell R610 server running our modified version of ESX. Each had dual-socket six-core 2.67 GHz Intel Xeon X5650 processors, 48 GB of RAM, and 4 Broadcom 1 GbE network adapters. Both servers were connected to two EMC VNX5500 SAN arrays using an 8 Gb Fibre Channel (FC) switch. We created a 200 GB VMFS version 5 file system on a 15-disk RAID-5 volume on each array.

We used the Maxwell Pro network emulator to inject latency between the hosts. Our long distance XvMotions were performed on a dedicated link between Palo Alto, California and Banglore, India with a bottleneck bandwidth of 1 Gbps.

We used three workloads, an idle VM, OLTP Simulation using Iometer [5], and the DVD Store version 2 [3] benchmark. The Idle and Iometer VMs were configured with two vCPUs, 2 GB memory of RAM, and two virtual disks. The first disk was a 10 GB system disk running Linux SUSE 11 x64, the second was a 12 GB data disk.

During the migration both virtual disks were migrated. Our synthetic workload used Iometer to generate an IO pattern that simulates an OLTP workload with a 30% write, 70% read of 8 KB IO commands to the 12 GB data disk. In addition, we varied outstanding IOs (OIOs) to simulate workloads of differing intensities.

DVD Store Version 2.1 (DS2) is an open source online e-commerce test application, with a backend database component, and a web application layer. Our DVD Store VM running MS Windows Server 2012 (x64) was configured with 4 VCPUs, 8 GB memory, a 40 GB system/db disk, and a 5 GB log disk. Microsoft SQL Server 2012 was deployed in the VM with a database size of 5 GB and 10,000,000 simulated customers. Load was generated by three DS2 client threads with no think time.

5.1 Downtime (Switchover Time)

Figure 3 shows the downtimes for our OLTP workload with network round-trip latencies varying from 0-200 ms. All XvMotions show a bounded downtime of roughly one second.

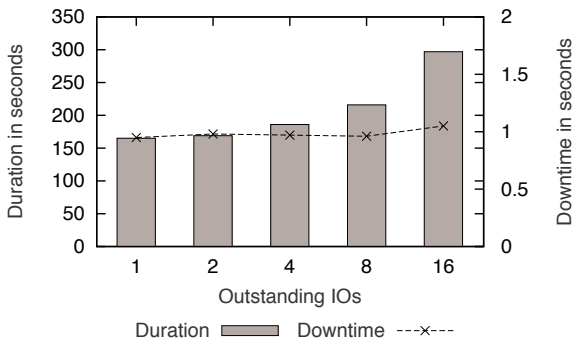

Figure 4: Migration time and downtime for varying OIO on the OLTP workload at 200 ms of round-trip latency.

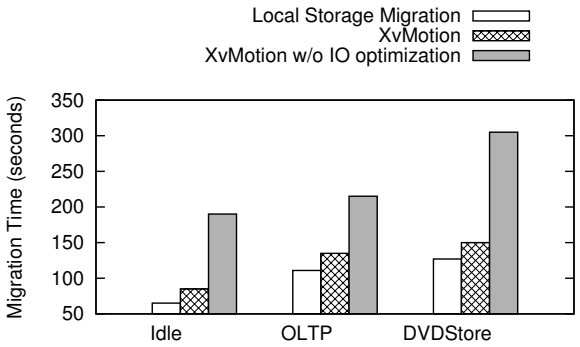

Figure 5: Shows the migration time for three workloads. The remote migration cases are approximately 10% slower for OLTP and DVDStore.

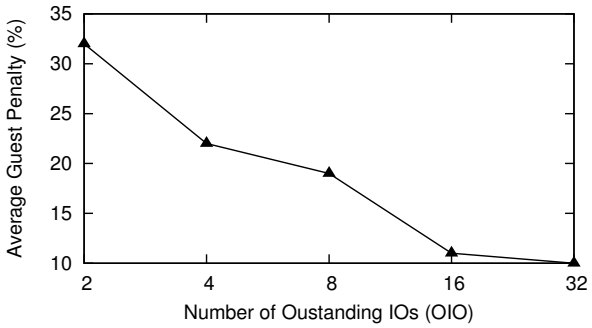

Figure 6: Guest workload penalty as workload intensity (OIO) increases. As workload intensity increases, the average guest penalty decreases, as our transmit queue scheduler gives increased priority to Guest IOs over clone process IOs. workload IOs over clone process IOs.

The XvMotion protocol iteratively sends memory pages: at every iteration it sends the pages written by the guest in the previous iteration. It converges when the remaining data can be sent in half a second, which is the value used in these experiments. Thanks to our SDPS optimization, even if the network bandwidth is low and the guest is aggressively writing pages, the remaining data is bounded to be sent in half a second.

When we measured downtime versus storage workload intensity (varying OIO) we discovered that the switchover time remained nearly constant. Figure 4 shows an average downtime of 0.97 seconds, with a standard deviation of 0.03. This shows we have achieved downtime independent of storage workload.

5.2 Migration Time

We compared an XvMotion between two hosts over 10 Gbps Ethernet, to local live storage migration in Figure 5, to approximate a comparison between XvMotion and local live migration.

Local live storage migration copies a virtual disk between two storage devices on the same host. While this is a bit of an apples to oranges comparison, we choose it for a few reasons. First, storage migration overhead generally dominates memory migration overhead. Second, storage migration is quite heavily optimized, so it provides a good baseline. Third, initiating a simultaneous local memory and storage migration would still not provide an apples to apple comparison as there would be no contention between memory and storage migration on the network, and the copy operations are uncoordinated.

As expected, live storage migration is the fastest. The data shows about 20 to 23 seconds difference between local live storage migration and XvMotion. Part of that overhead is from memory and device state migration that required about 2 seconds to migrate memory state in the Idle and OLTP scenarios, and 8 seconds in DVD Store scenario. Additional reasons that these differences exist are contention between memory migration and IO mirroring traffic, and greater efficiencies in the data mover. However, the encouraging result here is that local migration between hosts is not appreciably more expensive than migrating storage from one volume to another on the same host.

A last point illustrated by Figure 5 is the importance of our storage optimizations described in § 4.4. Migration time nearly doubles in all the scenarios without our disk copy optimizations.

In Figure 4 we measured migration time as we vary workload intensity. We see that migration time increases with guest OIO. This occurs because clone IO throughput decreases as guest IOs take up a larger fraction of the total throughput.

Figure 3 presents how the migration time of the OLTP workload changes as we vary round-trip time. We see that migration time increases by just 10% when increasing latency from 0 to 200 ms. We observed that most of the overhead came from the memory copy phase. In this case, TCP was not able to optimally share the 1 Gbps available bandwidth to both the memory and the disk mirror copy sockets. Additionally, TCP slow start at the beginning of every phase of the migration also add several seconds to the migration time.

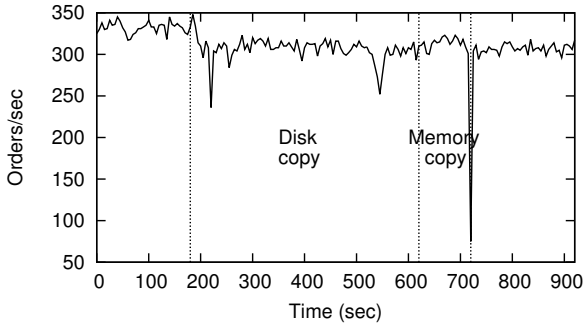

Figure 7: Illustrates the various phases of the migration against a plot of DVD Store Orders/sec for XvMotion.

5.3 Guest Penalty

Figure 6 shows average guest penalty decreases as workload intensity increases. This ensures that the impact of migration on guest performance is minimized.

The average guest penalty drops from 32% for 2 OIOs to 10% for 32 OIOs. This is because both clone process and mirror IOs compete for transmit queue space on the source as discussed in § 3.6. The clone process has at most 16 OIOs on the source to read disk blocks for the bulk copy. As the guest workload has more OIOs it will take a larger share of transmit queue.

Our transmit queue schedules bandwidth between clone process and mirror IOs. Without such a scheduler, we measured a 90% average guest penalty for the 8 OIO scenario. To further investigate, we measured the effect of varying latency for a fixed intensity workload. We observed the penalty for the OLTP workload with 16 OIO is fixed at approximately 10% for RTT latencies of 0 ms, 100 ms and 200 ms.

Figure 7 shows a dissection of DVD Store's reported workload throughput during an XvMotion on a 1 GbE network link with 200 ms round-trip latency. The graph shows the impact on the SQL server throughput during disk and memory copy, and shows a quick dip for one data point as the VM is suspended for switchover. This is an example of a successful migration onto a slower destination volume, where the workload is gradually throttled to the speed of the destination.

5.4 XvMotion on the Wide Area

We explored XvMotion's behavior on the wide area by migrating a VM from Palo Alto, CA to Bangalore, India. These two sites, about halfway across the globe, are separated by a distance of roughly 8200 miles. The WAN infrastructure between these two sites was supported by a Silverpeak WAN accelerator NX 8700 appliance. The measured ping latency between the sites was 214 ms over a dedicated 1 Gbps link.

Our source host in Palo Alto was a Dell PowerEdge R610 with a high performance SAN storage array. Our destination in Bangalore was a Dell PowerEdge R710 with local storage. To virtualize the network, we stretched the layer-2 LAN across these two sites using an OpenVPN bridge. This enabled the VM migrate with no disruption in network connectivity.

We successfully ran two workloads, Iometer and DS2, with minimal service disruption.

In the Iometer test, before the migration, the ping latency between the client and the OLTP VM was less than 0.5 ms since both were running on the same local site. After the migration the ping latency between the client and the OLTP VM jumped to 214 ms. The ping client observed just a single ping packet loss during the entire migration.

The OLTP workload continued to run without any disruption, although the guest IOPS dropped from about 650 IOPS before migration to about 470 IOPS after migration as the destination host was using low performance local storage.

The total duration of the migration was 374.16 seconds with a downtime of 1.395 seconds. We observed 68.224 MB/s network bandwidth usage during disk copy, and about 89.081 MB/s during memory pre-copy.

In the DS2 test, the DS2 clients ran on two client machines, one client machine located in Palo Alto and the other in Bangalore. The DS2 clients generated a substantial CPU and memory load on the VM. The migration was initiated during the steady-state period of the benchmark, when the CPU utilization of the virtual machine was little over 50%.

The DS2 workload continued to run without any disruption after the migration. However, the throughput dropped from about 240 ops before migration to about 135 ops after migration as the destination host was using low performance local storage.

The total duration of the migration was 1009.10 seconds with a downtime of 1.393 seconds. We observed 49.019 MB/s network bandwidth usage during disk copy, and about 73.578 MB/s during memory pre-copy. Some variation in network bandwidth was expected between the tests as the WAN link between sites is a shared link.

5.5 Summary

Our results show that XvMotion exhibits consistent behavior i.e., migration time, downtime, and guest penalty, with varying workload intensity and latency. We saw for all experiments, maximum downtime is around one second, even for latencies as high as 200 ms and data loss up to 0.5%. One of the nice features of our system is that for long distance migrations the VM migration was bottlenecked by the network bandwidth, and thus our storage throttling mechanism slowed the VM down gradually.

6 Discussion

We discuss potential split-brain issues when handing off execution from the source VM to the destination VM, then briefly survey networking and security considerations for wide area migration.

6.1 Split-Brain Issues

A live migration ends with the exchange of several messages between the source and destination hosts called the resume handshake. In the normal case, if the source receives the resume handshake, it can mark the migration successful, and power off. Upon sending the resume handshake, the destination can resume and mark the migration complete. With migration using shared storage, file system locking ensures only one VM can start during a network partition. Each host will race to grab locks on the virtual disks and metadata file, with only one winner.

With XvMotion there is no good solution and yet we must prevent both hosts from resuming concurrently. Both the source and destination wait for the completion of the resume handshake. The source will power on if it never sees a resume request. A network partition after the resume request message causes both hosts to power-off. If the destination receives approval to resume, it will from that point on. Finally, when the source receives the resume completed message it will power-off and cleanup. Anytime the hosts cannot resolve the issue, a human or the management software must power on the correct VM and delete the other VM.

6.2 VM Networking

Long distance migration presents a challenge for the VM's network connectivity. A VM may be migrated away from its home subnet and require a form of network tunneling or routing. Several classes of solutions are available to enable long distance network migrations. First, there are layer two networking solutions that tunnel traffic using hardware or software solutions such as Cisco Overlay Transport Virtualization (OTV) [14] and VXLAN [20]. For modern applications that do not depend on layer two connectivity there are several layer 3 solutions such as Locator/ID Separation Protocol (LISP) [12] that enable IP mobility across the Internet. Another emerging standard being deployed at data centers is OpenFlow [22], which enables generic programming and configuration of switching hardware. Using OpenFlow, several prototypes have been constructed that enable long distance VM migration.

6.3 Security Considerations

Any live migration technology introduces a security risk as a VM's memory and disk are transmitted over the network. Typically, customers use physically or logically isolated networks for live memory migration, but this is not sufficient for migrations that may be over the WAN. Today customers address this through the use of hardware VPN solutions or IPSec. While some customers may desire other forms of over the wire encryption support, we regarded this as outside the scope of our current work.

7 Related Work

Live Migration: Live VM memory migration has been implemented in all major hypervisors including Xen [10], Microsoft Hyper-V [1], and VMware ESX [24]. All three systems use a pre-copy approach to iteratively copy memory pages from source host to destination host.

Hines *et al.* [18] proposed a post-copy approach for live VM memory migration. Their approach essentially flips the steps of an iterative copy approach, instead of sending the working set at the end of the migration, it is sent at the beginning. This allows execution to immediate resume on the destination, while memory pages are still being pushed, and missing pages are demand paged in over the network. Memory ballooning is used to reduce the size of the working set prior to migration. Post-copying offers lower migration times and downtimes, but often induces a higher guest penalty and gives up atomic switch over. Luo *er al.* [19] used a combination of post-copy and pre-copy approaches to lower downtime downtime and guest penalty, but also gives up atomicity. Both of these approaches are unacceptable for wide area migration because of increased risk to losing the VM. Our SDPS technique offers a safer approach, reducing downtime without the loss of atomicity.

Storage Migration: ESX live storage migration has evolved through three different architectures, i.e., snapshotting, dirty-block tracking and IO mirroring [21]. Live storage migration of VMs using IO mirroring is explored in Meyer et al. [23]. The latest storage migration implementation in Microsoft Hyper-V [4], VMware ESX, and Xen with DRBD [6], are all based on IO mirroring.

WAN Migration: Bradford *et al.* extends the live migration in Xen to support the migration of a VM with memory and storage across the WAN [9]. When a VM is being migrated, its local disks are transferred to destination volume using a disk block level iterative pre-copy. The write IO workload from the guest OS is also throttled to reduce the dirty block rate. Further optimizations for pre-copy based storage migration over WAN are explored by Zheng *et al.* [28].

While the iterative pre-copy approach is well suited for memory migration, it suffers from several performance and reliability limitations for storage migration as shown in our prior work [21]. In contrast, we propose to seamlessly integrate memory pre-copy with storage IO mirroring for long distance live VM migration.

CloudNet addresses many of the shortcomings of Bradford's work by using DRBD and Xen to implement wide area migration along with a series of optimizations [27]. The system used synchronous disk replication rather than asynchronous replication used by XvMotion. Their *Smart Stop and Copy* algorithm tuned the number of iterations for memory copy, thus trading off downtime versus migration time. ESX used a similar algorithm internally, but downtimes were still sufficiently high even with this mea-

sure that we introduced SDPS. SDPS and asynchronous disk buffering allows XvMotion to target a specific downtime at the cost of increased guest penalty.

SecondSite is the first solution to use software fault tolerance to implement seamless failover of a group of VMs over a WAN [26]. This solution is built on Remus [11] a fault tolerance solution built on Xen's live migration infrastructure. SecondSite and Remus provide the destination periodically with consistent images of the VMs memory and disk. This is done while the VM is running on the source and the migration only completes when the source host dies.

8 Conclusion

We have presented XvMotion, a system for memory and storage migration over local and wide area networks. By integrating memory and storage movement we were able to achieve an atomic switchover with low downtime. Our use of asynchronous storage replication provides good storage performance in the presence of high latency links. We also introduced mechanisms to increase memory migration tolerance to high latency links, and make storage migration robust to diverse storage speeds.

Our OLTP tests show that an XvMotion between two separate hosts over 10 Gbps Ethernet, performed only 10% slower than a storage migration on a single host between two locally attached disks, demonstrating live migration on a shared-nothing architecture is comparable to live migration with shared storage. We also showed that while increasing the latency of the network to 200 ms we saw downtimes lower than one second, which are unnoticeable to most applications, demonstrating the live migration is viable over the wide area. We also showed that our system is well behaved under heavy load, as increases in guest workload do not effect downtime.

Higher bandwidth networks, network virtualization, large scale virtualization deployments, geographically separated data centers and diverse storage architectures are all increasingly important parts of data centers. Given these trends, we believe the ability to simply, reliably, and efficiently move a VM between two hosts afforded by XvMotion will enable new use cases, and help simplify existing situations where VM mobility is demanded.

9 Acknowledgements

We would like to thank all the people who gave us feedback and support: Michael Banack, Dilpreet Bindra, Bruce Herndon, Stephen Johnson, Haripriya Rajagopal, Nithin Raju, Mark Sheldon, Akshay Sreeramoju, and Santhosh Sundararaman. This work was partially funded by DARPA CRASH grants N66001-10-2-4088 and N66001-10-2-4089.

References

[1] Windows Server 2008 R2 Hyper-V Live Migration, June 2009. http://www.microsoft.com/en-us/download/details.aspx?id=12601.

[2] Distributed Replicated Block Device (DRBD), May 2012. http://www.drbd.org/.

[3] DVD Store version 2 (DS2), May 2012. http://en.community.dell.com/techcenter/extras/w/wiki/dvd-store.aspx.

[4] How does Storage Migration actually work?, Mar. 2012. http://blogs.msdn.com/b/virtual_pc_guy/archive/2012/03/14/how-does-storage-migration-actually-work.aspx.

[5] Iometer, May 2012. http://www.iometer.org/.

[6] Chapter 13: Using Xen with DRBD, May 2014. http://www.drbd.org/users-guide/ch-xen.html.

[7] ALLMAN, M. TCP Congestion Control with Appropriate Byte Counting (ABC), 2003.

[8] ASPESI, J., AND SHOREY, O. EMC VPLEX Metro Witness Technology and High Availability. Tech. Rep. h7113-vplex-architecture-deployment.pdf, EMC, Mar. 2012.

[9] BRADFORD, R., KOTSOVINOS, E., FELDMANN, A., AND SCHIÖBERG, H. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. VEE '07.

[10] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live Migration of Virtual Machines. NSDI'05.

[11] CULLY, B., LEFEBVRE, G., MEYER, D., FEELEY, M., HUTCHINSON, N., AND WARFIELD, A. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 161–174.

[12] FARINACCI, D., FULLER, V., MEYER, D., AND LEWIS, D. Locator/ID Separation Protocol (LISP). Tech. rep., IETF, Aug. 2012. Work in progress.

[13] FLOYD, S., AND JACOBSON, V. Random Early Detection Gateways for Congestion Avoidance. *Networking, IEEE/ACM Transactions on 1*, 4 (1993), 397–413.

[14] GROVER, H., RAO, D., FARINACCI, D., AND MORENO, V. Overlay Transport Virtualization. Internet-Draft draft-hasmit-otv-03, Internet Engineering Task Force, July 2011. Work in progress.

[15] HA, S., RHEE, I., AND XU, L. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev. 42*, 5 (July 2008), 64–74.

[16] HACKER, T. J., ATHEY, B. D., AND NOBLE, B. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM* (2002), IEEE, pp. 434–443.

[17] HACKER, T. J., NOBLE, B. D., AND ATHEY, B. D. Improving throughput and maintaining fairness using parallel tcp. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* (2004), vol. 4, IEEE, pp. 2480–2489.

[18] HINES, M. R., DESHPANDE, U., AND GOPALAN, K. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev. 43*, 3 (2009), 14–26.

[19] LUO, Y. A three-phase algorithm for whole-system live migration of virtual machines. CHINA HPC '07.

[20] MAHALINGAM, M., DUTT, D., AND ETC. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Tech. rep., IETF, Feb. 2012. Work in progress.

[21] MASHTIZADEH, A., CELEBI, E., GARFINKEL, T., AND CAI, M. The Design and Evolution of Live Storage Migration in VMware ESX. USENIX ATC'11.

[22] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[23] MEYER, D. T., CULLY, B., WIRES, J., HUTCHINSON, N. C., AND WARFIELD, A. Block Mason. In *Proceedings of the First Conference on I/O Virtualization* (Berkeley, CA, USA, 2008), WIOV'08, USENIX Association, pp. 4–4.

[24] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast Transparent Migration for Virtual Machines. USENIX ATC'05.

[25] PEDDEMORS, A., SPOOR, R., DEKKERS, P., AND BESTEN, C. Using drbd over wide area networks. Tech. Rep. drbd-vmigrate-v1.1.pdf, SurfNet, Mar. 2011.

[26] RAJAGOPALAN, S., CULLY, B., O'CONNOR, R., AND WARFIELD, A. SecondSite: Disaster Tolerance As a Service. *SIGPLAN Not. 47*, 7 (Mar. 2012), 97–108.

[27] WOOD, T., RAMAKRISHNAN, K., VAN DER MERWE, J., AND SHENOY, P. CloudNet: A Platform for Optimized WAN Migration of Virtual Machines. *University of Massachusetts Technical Report TR-2010-002* (January 2010).

[28] ZHENG, J., NG, T. S. E., AND SRIPANIDKULCHAI, K. Workload-aware live storage migration for clouds. VEE '11.