Revision History

Version	Date	Author	Description
0.1	08/14/2013	Le Tian/Hill Zhao	Initial draft.

Acronym and Codename Table

1 Foreword

1.1 Basic

2 vMotion Setup

2.1 Setting up your machines

- You'll need two machines to do VMotion, unless you're doing a self-VMotion, in which case you only need one.
- If you're using two machines:
  - They must have CPUs that belong to the same CPU family.
  - They need a shared VMFS volume (it can be SAN, NFS, or iSCSI).
    - Note: This common datastore must have the same name on each machine, i.e. the relative path to the disk files should be the same.

2.2 Setting up the VMkernel

- Create a vmknic for the kernel -- check out EsxNetworkingSetup for tips on how.
- Be sure to build the VMkernel tcp/ip module (ie, `make PRODUCT=server tcpip`).
- Also be sure to build the migration module (ie, `make PRODUCT=server migration`).
- `load-esx` will automatically load both modules for you if they are built.
- Enable migration with `echo 1 >> /proc/vmware/config/Migrate/Enabled`.
- Do this on both the source and destination machines.

2.3 Setting up the virtual machine

- You need to place the VM on the shared storage you set up previously.
  - Technically, only the disk and swap file need to reside on the shared storage, but in practice it's easier to put everything there.
- No special setup is necessary for the config file for VMotion.

2.4 Other misc. setup

- It's a good idea to add `msg.autoAnswer = "TRUE"` to /etc/vmware/config. This will prevent the VMX from getting stuck in a Msg_Post, which can make it unresponsive.

2.5 Building migrateTest

- To build migrateTest, run make PRODUCT=server migrateTest in your vmkernel-main tree.

2.6 Running migrateTest

- Simply running migrate produces the following help message:

```
Usage: migrate [user1.pass1@host1,]vmkip1:cfg1 [user2.pass2@host2,]vmkip2:cfg2
               [-v #] [-u] [-hipri] [-rt [# of iters]] [-e2d] [-na]
               [-w # of secs] [-r [# of retries]] [-rw # of secs]

Migrate a VM without hostd or VC!

  user    a valid username for the COS
  pass    a valid password for that username
  host    COS hostname or IP address
  vmkip   VMkernel VMotion IP address
  cfg     Absolute path to the VM config file (/vmfs/volumes/...)

VMkernel ip address and config file are mandatory.  Username, password, and
hostname are all optional, although they must either all be omitted or all
specified.  If they are omitted, the host is assumed to be localhost.

  -v #    Sets the verbosity level (default: 0)
  -u      Enables 'unshared swap' VMotion (default: shared swap)
  -hipri  Enables a high-priority VMotion (default: normal priority)
  -rt [#] Perform a roundtrip VMotion, optionally specify the number of
          roundtrips (default: one-way, if only -rt is set: 1 roundtrip)
  -e2d    Perform an ESX 2.5.x -> ESX 3.0.1+ VMotion.  This option only
          applies to one-way migrations and thus is not compatible with -rt.
  -na     No auto-start.  Authd won't start vmx if it's not running
          (default: authd starts the vmx if it's not running)
  -w #    Sets the number of seconds to wait between VMotions (default: 3)
```

```
-r [#]  Sets the number of retries for certain failures (default: 0)

-rw #   Sets the number of seconds to wait between retry attempts (default: 3)
```

- migrate root.pass@destESX,10.10.10.2:/vmfs/volumes/foo/bar/bar.vmx root.pass@sourceESX,10.10.10.1:/vmfs/volumes/foo/bar/bar.vmx

Reference:
https://wiki.eng.vmware.com/HotMigration
https://wiki.eng.vmware.com/UsingMigrate
https://wiki.eng.vmware.com/EsxNetworkingSetup

3 VMotion

3.1 Overview

3.2 Compatibility

Reference:

2007-06-08-joel-baxter-slides-000.ppt

3.3 CBT

Reference:

ESX_CBT.ppt

3.4 FSR

Reference:

Fsr.ppt

4 Scalable VMotion

4.1 Overview

Scalable VMotion is a feature that encompasses a collection of components, all focused on making VMotion perform better. The goal of this work is not only to lessen VMotion's impact on a migrating VM, or to utilize more available bandwidth, but also to add some level of redundancy and general workload-aware optimizations.

4.2 Problems

While we investigate performance issues, we have some ideas on what might be improved:

- Zero copy sends and receives: right now VMotion copies into a bounce buffer before calling into Net_Send/Receive (or whatever the exact call is).
- Multiple send threads: right now VMotion only has a single send world. I've heard that there is some limit to how much a single thread can send and that if we used multiple send threads we might be able to get a higher throughput.
- Batch sends: right now VMotion only sends one page's data at a time. We could certainly batch a bunch of pages up.
- Bigger page list from monitor: currently the monitor sends up a bitmap of 256 pages to send. If we find that the roundtrip to/from the monitor is a bottleneck, we could bump this up to 512 or 1,024.
- Pre-allocate pages on receive side.

4.3 Goal

- 5Gbps bandwidth on 10GbE networks
- Decrease overall VM downtime, with a focus on representative workloads for Microsoft Cluster Service (MSCS) clusters
- Lessen VMotion CPU and memory overhead, improve memory accountability

- Increase likelihood of VMotion success, for even memory intensive operations

Performance and scalability targets can be broken down by category:

- Bandwidth: We are targeting a minimum of 5Gbps for initial pre-copy phases over 10GbE network links.
- Page transmission: HEAT is expected to lessen page retransmission. (e.g. for SPECJBB2005 workload it reduced 50,000 pages on average).
- Aggressiveness: VMotion should be able to aggressively succeed on workloads known to not be VMotionable with today's code.
- Competitive: We intend to analyze and refine our goals based on Hyper-V live migration performance.

4.4 Technical Driver(s)

Here are some of the highlights:

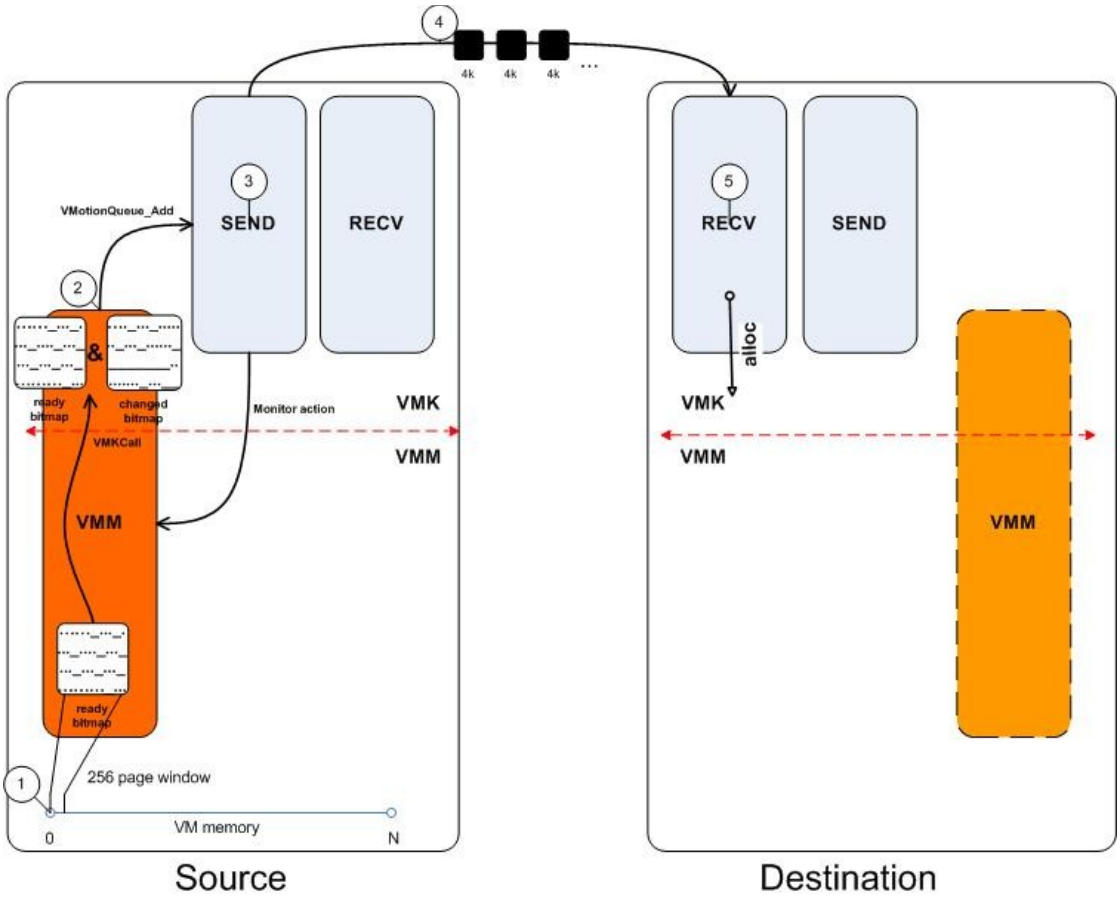
- VMotion performance on 10GbE environments is less than impressive.
- Memory intensive workloads can cause VMotion to fail.
- VM downtime in some (extreme) cases can be fairly large causing VM network disruption.

4.5 Limitations

Broadly speaking, the largest limitation of VMotion is its performance issues. VMotion performance can be broken down into the following metrics:

- VM downtime: Time which the VM needs to be stunned before it is resumed at the destination.
- Overall VMotion time: absolute time it takes from the beginning to the end of the VMotion

Both metrics are a function of the VM memory dirty rate and the VMotion network bandwidth.



The VMotion pre-copy phase has the following steps which are problematic for performance:

1. 256 page window: Each iteration of the pre-copy phase requires that the monitor send a changed page bitmap, covering a 256 page window, to the kernel. Only pages marked as changed in the bitmap are considered for transmission. Early on in the pre-copy process, when this bitmap is largely full, this serves as a useful optimization. In later pre-copy phases, however, the bitmap becomes more sparse. In some cases, it will be entirely empty. Entire send iterations can be wasted if empty or very sparse bitmaps are sent down to the kernel.
2. VMotionQueue_Add/Monitor Action cycle: Each send iteration, as described above, is triggered by a monitor action, which in turn is triggered by an action queue. As such, sending is serialized: an action is not queued until the previous send has completed, and even then is only executed when the monitor triggers the action.
3. Send World: Each VMotion has only a single send world, which executes largely at the mercy of the VMotionQueue_Add/Monitor Action cycle, as described above. Multiple send worlds should help to alleviate the performance cost of serialized transmits.

4. Single page at a time: Each page is sent separately, in a separate packet with its own header. Batching page sends will serve to lower the overhead imposed by page transmission.
5. Receiver World: The receiver world works by allocating memory for a page and receiving the page. Preallocation of pages, along with UIO operations, will help to lessen overhead.

1. preflight 2. precopy 3. save checkpoint 4. stun 5. unstun 6. transfer checkpoint 7. restore checkpoint 8. assume identity 9. release identity

Limitation 2

VMotion has no concept of redundancy. Should a network failure occur, any active VMotions will fail. Introducing redundancy through disk-backed operations should significantly improve VMotion reliability during the critical "switchover" period of VMotion.

Limitation 3

During both the pre-copy and page-in phase, page transmissions are not considered in any particular order. Pages are simply sent in order of address, and resent whenever their contents changes. Utilizing HEAT information, however, should allow us to decide which pages can be sent a single time during pre-copy, and which pages should simply be held off for transmission during page-in.

Limitation 4

Currently, the VMotion model consists at a high level of pre-copy, page-in, and then resume. The destination VM cannot resume until page-in completes, as every changed page will have been provided to the VM from the source. This is a key limitation of VMotion - in many cases, it is feasible to resume the destination VM before all pages have been received, since the VM only requires its working set to continue functioning. Addressing this limitation is coupled tightly with utilization of HEAT information, and the additional redundancy provided by disk-backed operations.

4.6 Functional Description

Scalable VMotion will decrease the overall VMotion time and the VM downtime due to VMotion. This will require a number of improvements in the VMotion design and protocol.

Feature(s) including mapping to PRD requirement

- **Aggressive Resume**: After completion of multiple rounds of page pre-copying, VMotion currently stuns the source VM so that it can transmit all remaining changed pages. The destination VM is not resumed until all pages have completed transfer. Aggressive Resume will be a set of features focused on allowing the destination VM to resume before all changed pages have been received, letting the resumed VM make use of remote page faults to retrieve pages as necessary from the source. Performance/reliability analysis is critical to this feature.
- **HEAT**: Determines working set, and can classify which pages are most likely to be changed. Using this data will allow VMotion to prioritize sending pages based on expected change frequency, reducing the number of page retransmissions.
    - Dependencies
        - For HEAT, we are working with Intel to add A/D bits into the extended page tables and we are expecting they are available in 2012 (confidential information). These A/D bits reduce the HEAT overheads on Nehalem significantly.
        - HEAT is NOT available on software MMU hence any CPUs prior to Barcelona/Nehalem can't take advantage of HEAT information.
    - HEAT adapted CLOCK-Pro with the following differences
        - Classifies in-memory pages
        - Variable number of hot pages
        - Timer driven (as opposed to page fault)
        - Input parameter: scanning speed
            - The number of guest pages to be scanned in certain period

- **Performance Optimizations:** Addresses three key areas of overhead in VMotion. **Page batching**: Rather than step over the bitmap of send pages one bit at a time, sending/receiving each page in turn, this change allows for batched sending/receiving of groups of pages. Significant performance gains have already been observed in prototype implementations. **Multiple Receiver/Sender**: Currently, the architecture calls for one sender/receiver pair per VMotion. Exploring multiple send/receive worlds should have performance benefits, assuming CPU-bound processes**. Window Analysis**: In addition to the send bitmap, VMotion involves a changed bitmap, which is computed in the VMM and sent down to the kernel. This bitmap covers a 256 page window, and necessitates that the monitor be invoked continually to provide more windows into memory.

Exploration of different window sizes, along with alternatives to utilizing the window, should yield significant performance improvements.

4.7 Software Architecture and Design

4.7.1 **Concept**

Aggressive Resume: As outlined, the concept of aggressive resume is simple. A destination VM may be resumed immediately after having loaded its checkpointed state, whether it has all of its pages or not. Changed pages can be demand remote faulted over the network from the source VM. The aggressive resume component is a high level component, encompassing the change to allow VMotion to take advantage of this property. Specifically, it will proceed with VMotion as normal, until pre-copy has succeeded. After the checkpoint is transferred, it will briefly initiate the page-in process. Normally, a VMotion will pause here, with both src and dest stunned, until page-in completes. Aggressive resume, however, only allows page-in to take a certain length of time, after which point the destination VM will be resumed, whether or not it has received its changed pages completely. Page-in continues in the background until the VMotion completes. Pages required as the destination VM runs are demand remote faulted, as described. This feature relies on disk-backed operations for redundancy, otherwise a network failure after aggressive resume would necessarily result in the loss of the VM, as its memory contents would be impossible to determine. As consequence, aggressive resume will be automatically disabled if disk-backed operations are not available.

Disk-backed operations: At the beginning of the restore phase, before page-in begins, the source host will reserve a space on shared storage of sufficient size to hold a multi-megabyte circular buffer. The source host will stamp the file with the current migraiton ID. Immediately thereafter, the destination host will attempt to open the file and match its migration ID against the stamped ID. If the destination host fails to open the file, or if the migration IDs do not match, disk-backed operations will be disabled and aggressive resume switched off. Upon detection of a network failure, the source side will begin streaming pages into the circular buffer, updating a special segment of the file with the buffer head. Likewise, as the destination host reads from the circular buffer, it will write to a tail segment. VMotion cleanup on both sides will attempt to cleanup the file with migration completion, allowing whichever side holds the lock last to successfully remove the file. As this approach requires multiwriter support, it may only be applicable to VMFS.

HEAT: Aims to provide accurate and transparent online VM memory page heat classification at the hypervisor. HEAT periodically goes through the nested/extended page tables to get the dirty bit for each page and if the dirty bit is set it also resets it. It classifies the pages by looking at the history

of dirty bits over a period of time. Awareness of page heat will allow for page transmits to be ordered such that pages expected to change are sent later than cold pages during pre-copy, lowering the number of necessary page retransmits due to content changes. Note that this can also benefit aggressive resume, if the page-in phase is altered to send the hottest pages first. If the aggressively resumed has already received its hottest pages by the time it resumes, it will be very unlikely to need to remote fault additional pages.

Performance Optimizations: In order to improve vmotion throughput, we need to ensure that the send world is keeping the network pipe full and the receiver is receiving the data as fast it can. From the preliminary tests, we have identified the following optimizations that may help improve throughput:

- Sender
  - Reduce the message overhead during precopy by batching multiple page headers in one page.
  - Monitor makes a call down to vmkernel to send 256 pages. If we can increase this number, it will reduce vmm->vmk calls and thus reduce latencies on the sender side.
  - Monitor waits for an action before sending the next page list. This again adds latency and send world's queue is often empty. We should keep multiple lists in queue so that the send world never blocks waiting for the monitor to send the next list. IF we can not keep multiple queues, then the size of a pagelist should be large enough that by the time sender world is done sending it, vmm can prepare the next page list and send it to vmkernel.
  - During the page-in phase, page list is sparse. Send world goes through many empty pages before it can find any page to send. As the VM memory size increases, this becomes even worse. We need to store the modified pages in a list or something similiar so that we do not spend time scanning unwanted pages.
- Receiver
  - Use scatter gather receive so that multiple pages can be received together.
  - Alloc pages in batch. Currently we alloc one page at a time. Allocating in batch might help reduce some latencies.

- Use receive callbacks so that receive can be done in parallel to page alloc.
- Monitor
  - Consider sending a pagelist to vmkernel instead of a bitmap
  - Avoid making a vmkcall when there is no page to send
  - Evaluate and optimize the latency of trace installation
- Other general optimizations
  - Reduce any possible lock contention if there is.
  - Tune the socket buffer size.
  - Evaluate scheduling aspects. Some affinity settings might help.

4.8 Performance Characterization Outline

We will characterize VMotion performance across many dimensions:

- Application type - SPECweb, Exchange Server, SPECJbb, Idle VM.
- VM Working set size - ??
- VM Memory size - upto 64GB ??

Performance Metrics

- Bandwidth
- Downtime, Totaltime
- CPU utilization

Our goal is to improve vmotion throughput for all workloads and achieve at least 5Gbps in the best case. We will also compare vmotion performance to Hyper-V. Tune HEAT parameters based on observed saved page recopy.

VMotion performance, with and without aggressive resume, will need to be compared over a standard array of memory-focused workloads. Analysis of performance results will then be used to establish default aggressive resume configuration values. Various window sizes will be tested and compared to establish optimal defaults.

General performance testing, with and without any of the described components, will be used to characterize overall performance improvement as a result of this feature.

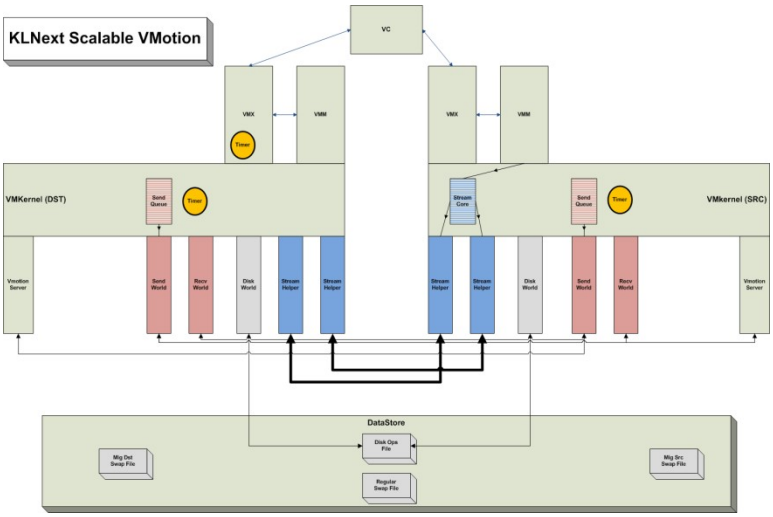
4.9 Limit Enforcement

In order for a migration operation to be able to start on an ESX host, all three limits must be satisfied for that operation. If any of these limits cannot be satisfied, then the operation will be queued until some other operations complete.

Each operation has a per-resource "cost" (C) and each resource has an in-use cost (I) (which is 0 when not in use) and a "max cost" (M) that is defined by the limit. For any operation, a resource's in-use cost increases by the he cost for that operation provided, max cost for that resource is not exceeded. i.e. I + C <= M.
For instance, the cost of a VMotion on an ESX 4.0 host is 4, while the cost of a VMotion on an ESX 4.1 host is 1. Similarly, a Storage VMotion on ESX 4.1 has a cost of 4. Max Cost for an ESX 4.1 host is 8. This means that you could do 8 VMotions, or 4 VMotions and 1 Storage VMotion, or 2 Storage VMotions simultaneously on an ESX 4.1 host.

Reference:

https://wiki.eng.vmware.com/VMotion
https://wiki.eng.vmware.com/VMotion/ScalableVMotionSpec
https://wiki.eng.vmware.com/VMotionPerformance
https://wiki.eng.vmware.com/Performance/DevPerf/Kandinsky/tcpip2TestWorld
https://wiki.eng.vmware.com/VMkernelQA/ResourceManagement/kl_next/TDS_VMotion
https://wiki.eng.vmware.com/VMkernelQA/ResourceManagement/kl_next/TestPlan_VMotion



Reference:
KLNext-ScalableVMotion.ppt

Reference:
MNEducationScalableVMotion.pptx
vmotion_info_guide.pdf

4.10 Concurrent Scalability

4.10.1 **Motivation**

Currently, VC supports a maximum of two concurrent VMotions per host. This is a constant number regardless of the number of VMotion pNICs in the host, the speed of those pNICs, the number of VMs on the host, etc, etc. Obviously as VM/host density increases, it is very desirable to be able to scale the number of concurrent VMotions, especially for situations like maintenance mode and DPM. While we have been working hard to increase single VMotion performance to scale on 10Gb (see VMotionPerformance), in order to achieve much faster overall VMotion times for a group of VMs, more hardware is needed and we'll need to support that additional hardware.

There are a variety of potential solutions for improving VMotion concurrent scalability. They are:

- Increasing the number of supported concurrent VMotions per pNIC
- Supporting multiple pNICs for VMotion using a single IP address
- Supporting multiple pNICs for VMotion using multiple IP addresses

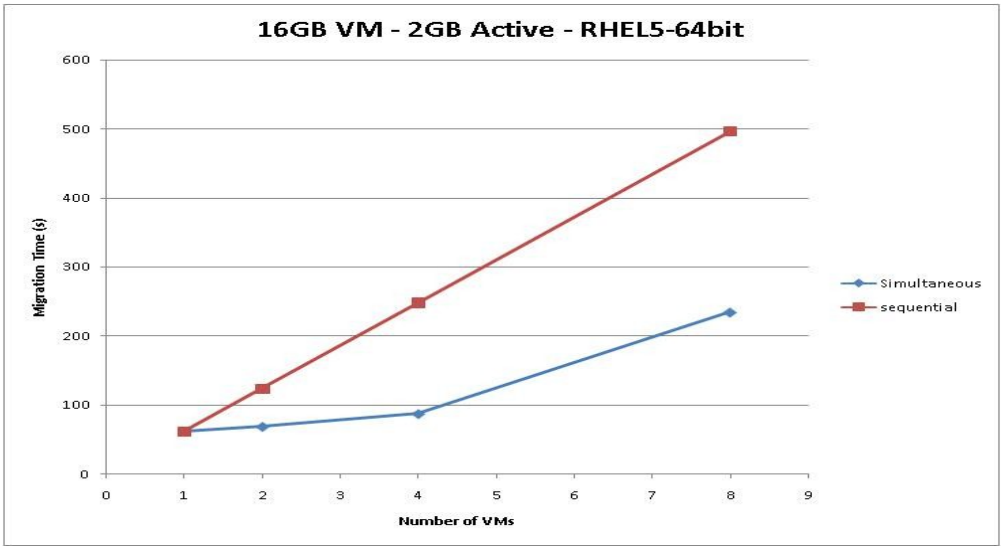
4.10.2 **Solutions**

Currently there are two proposals that have been put forward:

- Increase single-NIC hardcoded limits: This would simply increase the number of VMotions a single NIC can do. Based on the June 25th meeting, this will be our primary focus for K/L and KL.next.
- Support teamed pNICs using IP hash: In theory this would allow VMotions to load-balance across multiple NICs while using only a single IP address. As was discussed in the June 25th meeting, the IP-hash doesn't load-balance very well when all IP addresses are in the same subnet, meaning that more than likely only a single pNIC will be used, defeating the point of NIC teaming. Thus this proposal is not being explored further.

Given the results of our first round of testing, the following updated limits are being considered:

- 1GbE: 4 concurrent VMotions
- 10GbE: 8 or 10 concurrent VMotions



Support customers modifying vpxd.cfg to add the following config options:

- vpxd.ResourceManager.maxCostPerHost = 16/1GbE or 32 or 40/10GbE
- vpxd.ResourceManager.maxCostPerDs = 256 (???)
- config.task.totalThreads = 64

Reference:
https://wiki.eng.vmware.com/VMotion/ConcurrentScalability/KL.next
http://www.vmware.com/files/pdf/virtual_networking_concepts.pdf
http://www.formortals.com/an-introduction-to-vlan-trunking/
https://wiki.eng.vmware.com/VMkernelQA/ResourceManagement/VMotion/Limits/Simultaneous
https://wiki.eng.vmware.com/VMotion/ConcurrentScalability/SupportTeamedPNicsUsingIPHash
https://wiki.eng.vmware.com/wiki/images/9/9d/Concurrent_VMotion_Scalability_-_June_2009.ppt

5 vMotion Anywhere

5.1 Introduction

vMotion anywhere was first coined and presented at [RADIO 2007].

The basic idea is to remove VM mobility restrictions in the private and public cloud as well as to and from clouds. Limitations today include:

- Virtual Center: both source and destination hosts need to be managed by the same virtual center instance
- Shared storage: both source and destination hosts need to be connected to the same SAN.
- Shared network: both source and destination hosts need to be connected to the same IP subnet (for VM network connectivity)
- No group migrations: The fundamental unit of migration has been a VM but the customers applications typically span VMs which all work in concert.
- Distance: both source and destination hosts need to be within a specified distance and need to have a specific latency between each other
- Security domains: both source and destination hosts must "trust" each other and the VM being migrated.

This allows customers:

- Increased agility
- Increased elasticity

- Increased availability

5.2 Long Distance VMotion

5.2.1 **Overview**

This project covers the necessary vMotion optimizations within the larger "vMotion Anywhere" effort that among other things allows vMotions over large distances (metro and trans continental US). This specification will only describe the vMotion performance optimizations needed to provide a strict upper limit of 1 second for switchovers and, opportunistically, fast pre-copy of virtual machines during vMotion assuming some minimum network requirements:

- Maximum round trip time of 100 ms and at least 1 Gbps (to support 4 concurrent vMotions).
- Average Jitter of 0.1 ms. Quick, unexpected increases in latency could exceed our switchover time limit.
- Source and destination hosts should share the same subnet and they should be part of the same security domain. Migration across subnets may be available for 2014 but is part of a different specification.

The 2014 release will support high latency vMotions using shared storage, replicated storage based on VPLEX, and storage migrated with the XvMotion protocol. Integration with VPLEX is specified atLink Here. We do not expect any major change to the storage side of XvMotion needed to support high latencies.

5.3 X-vMotion

X-vMotion is the simultaneous migration of the storage/memory/device state/cpu state of a running virtual machine over the network.

X-vMotion allows for complete migration of a virtual machine from any host to any other host which it is connected to (modulo cpu compatibility, VM IP connectivity, sufficient storage, etc). X-vMotion involves the synchronization of the Storage vMotion and vMotion convergence such that they end up converging almost simultaneously.

The issues here with switchover time can be magnified since the stun for the two operations will be handled together. Workloads which involve memory or I/O exclusively or workloads that shift from I/O intensive to memory intensive in the middle might prove to be difficult to handle.

For more information please read the [functional spec].

[Need to add] If the workload is only being migrated temporarily for a maintenance activity then we will need to keep the storage image at the source so we can leverage on the return trip (maybe we should implement some sort of cache).

Reference:
https://wiki.eng.vmware.com/VMotion/VMotionAnywhere
https://wiki.eng.vmware.com/VMotion/VMotionAnywhere.old
https://wiki.eng.vmware.com/LD-Vmotion_functional_specification
https://wiki.eng.vmware.com/TCP_analysis_for_high_latencies
https://wiki.eng.vmware.com/VcProvisioning/XVC
https://wiki.eng.vmware.com/StorageVMotion/CrossHost2Spec

6 Storage vMotion

6.1 Functional description

Storage VMotion migrates a VM's storage while it is running, without requiring any downtime for the VM. Relocation of powered off and suspended VMs is already mostly handled by VC and will not be discussed in this document.

A Storage VMotion is initiated for a VM either through the Migration Wizard or through the command-line app, as is described below. Storage VMotion functionality is also accessible through the API. Thus any third party could write their own application to initiate a Storage VMotion.

The Storage VMotion operation will migrate the entire VM to a new destination as specified by the user. If the user chooses, various parts of the VM can either be left in their original location or migrated to a different location than the rest of the VM. In addition, the user can also choose to only migrate a single disk of the VM and leave the rest of the VM/the VM itself in its original location. By default, if a new location for the VM as a whole is specified, then all disks, regardless of whether they reside in the same location as the VM before the operation, will be moved to the new location for the VM. Again, in these cases, the user may wish to move these disks to different locations and that functionality is supported. The only individual relocatable parts of the VM are its disks (each individually relocatable) and the VM "home" (which includes the config file, swap file, nvram, log files, etc). [Also note that the swap file may not be located in the VM "home" if, for instance, host-local swap is enabled.] RDM virtual disks themselves cannot be relocated, however the "link" to the RDM in the VM "home" will be relocated.

Storage VMotion supports moving a VM between any type of storage that is supported by ESX. This includes FC, iSCSI, NFS, and local SCSI storage. The source and destination storage devices can be any of storage type - they do not need to be identical. Currently, both the source and destination storage devices need to be visible to the host that the VM currently resides on. However, we may support moving the VM between hosts as well as changing the VM's storage. Thus, in most cases, the VM will not change execution host.

Storage VMotion can be performed on essentially any type of VM. It is guest-os agnostic - all guest OSes supported by ESX are supported by Storage VMotion. Both the guest OS and applications running within the guest will continue to operate uninterrupted as if nothing had happened before, during, and after a Storage VMotion. In addition, Storage VMotion will not negatively affect various VI services such as VCB or DRS; they will continue to operate as normal. Storage VMotion also supports VMs with snapshots and VMs with various disk types (such as non-persistent, independent-nonpersistent, etc).

When a Storage VMotion is executing for a VM, it will appear as a task in the VC task pane, where its progress will be tracked. Its progress is mostly based on how much data has been copied. The Storage VMotion process itself should not degrade the performance of the VM that much. We expect the performance degradation to be much less than 10%. In order to ensure that performance degradation is not too severe, we will restrict the simultaneous number of Storage VMotions per datastore to two. Thus a datastore can be the source or destination of at most two Storage VMotions simultaneously.

If the Storage VMotion fails for any reason, the VM should continue to run. There are some Byzantine failure cases where the VM might die, especially if we don't implement "improved DMotion" (as described below), but those cases involve multiple failures from various components. In general the VM should always be running after a Storage VMotion failure.

6.2 Design

6.2.1 **Virtual Disk Relocation**

In previous releases of Storage VMotion, relocating a VM's virtual disks entailed the use of REDO logs. The REDO log method worked because it was expedient and proven, however there were many downsides to that method. First, it reduced the VM's performance. Second, it took an unknown amount of extra space (as the REDO logs grew, they consumed more datastore space). Finally, the REDO log method forced the VM to rely on both the source and destination datastores simultaneously, if only for a small amount of time, which increased the VM's exposure to physical disk failure.

6.2.2 **Disk Pre-Copy**

Going forward, we would like to do away with the REDO log method. Our new approach is to copy the disks much as we copy the memory of the VM during a VMotion: by pre-copying it. In this approach, we start a simple byte-for-byte copy of the VM's disks from the source datastore to the destination datastore. At the same time, we begin tracking changed disk blocks. After the disk copies finish, we take the list of changed blocks and iterate through it, copying each changed block from its source disk to its destination disk. Again, at the same time, we start tracking a new list of changed blocks. After the second copy phase ends, we take our new list of changed blocks and begin copying it, again starting a new list of changed blocks. Eventually the list of changed blocks will be small enough that we can copy it very quickly, at which point we can suspend the VM, copy the last few changed blocks to the new disks and switch over to using those new disks.

The above algorithm can be written as:

1. Start tracking changed blocks/create changed block bitmap for all the VM's disks.
2. Copy full disks to destination datastore.
3. Check if the number of changed blocks can be copied in a small amount of time.
    a. If so, we're ready to suspend on the source.
4. If not, copy all blocks in the changed block list directly into their corresponding blocks on the destination disk.
5. Goto step 3.
6. The VCB team is working on a changed block tracking mechanism that should work nicely for us: http://vmweb.vmware.com/~ceci/vcb/changedblocks .

Reference:

http://vmweb.vmware.com/~ceci/vcb/changedblocks

6.2.3 **Fast Suspend/Resume**

"Fast suspend/resume" is a new technology being developed by the hosted group. It is similar to VMotion in that it starts a new copy of the vmx and vmm, but instead of copying memory, it "transfers" memory from the original vmx/vmm to the new vmx/vmm. We can use fast suspend/resume to start a new version of the vmx and vmm for the VM with the new config file in the new VM home directory and the VM will automagically start using the new files.

Benefits

- Simple from a vmx point-of-view.
- Will probably be used for other features as well, thus will receive additional testing.
- Fairly easy to reason about.

Problems

- Still requires doubling overhead memory (overhead memory is typically about 1.5% of main memory).
- Because of this, will require DRS interaction.
- Because of this, may hit interesting failure cases.
- Tricky from the vmkernel point-of-view. Need new code to handle memory and memory reservation handoff.
- "Big hammer" approach - coarse-grained fix for a specific problem

6.3 Mirror Storage vMotion

6.4 Storage vMotion Snapshot

6.4.1 Current Limitations

- If a VM is powered on, it's storage can not be relocated as long as there is any snapshot taken on this VM. The customer will have to either delete the snapshots or power off the VM to relocate its storage.
- If a VM is powered off or suspended, it's storage can be relocated only if all its config file, disks, and snapshot files are in the same home directory. And all the VM's files can only be relocated to one destination datastore.
- If a VM has multiple disks on different datastores, all the delta disks after the snapshot is taken will be placed to a single datastore specified by the snapshot directory. This basically offsets the effects of Storage VMotion of VM's disks. All delta disks will still be on the same datastore, and introduce potential availability and IO performance issues.
- When a VM's snapshot directory is changed, the existing snapshot files in that directory can not be moved to the new one. Only the new snapshot files taken after the reconfiguration will be placed in the new directory.

6.5 Storage vMotion Cross-host

6.6 Implementation

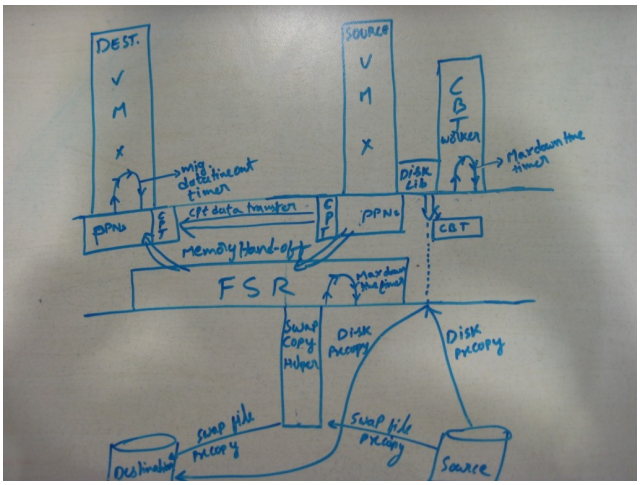
There are a variety of places we could implement the logic for the above design: VC, hostd, the vmx, or the vmkernel. There are tradeoffs to putting the main logic in each of those locations.

- VC: Generally we deemed VC as too far removed from the action to warrant us implementing the logic there. hostd has all the requisite functionality, most notable of which is NFC (network file copy).
- VMkernel: On the side of things, we prefer to not put anything in the vmkernel unless it has to go there. The right place definitely seems like vmx or above, so we're not going with the vmkernel.
- hostd: Originally, it was thought that most of the logic would go into hostd, as it's a very good candidate. However, most of the stat collection code will be in the vmx. And all we need hostd for is copying. So it seemed simpler to have hostd act as a beefed up ftp for this feature.
- vmx: This leaves the vmx, which can do all the stat collection and reasoning itself. It can also snapshot its disks and message hostd to perform the copy (or alternatively do the copy itself over the vmkernel network, but since it doesn't have NFC we're going with hostd for copying for the time being).

With these implementation decisions in mind, we can take a stab at guesstimating time requirements.

Reference:

https://wiki.eng.vmware.com/StorageVMotion/Design
https://wiki.eng.vmware.com/StorageVMotion/SnapshotsSpec
https://wiki.eng.vmware.com/StorageVMotion/CrossHostSpec
https://wiki.eng.vmware.com/StorageVMotion/SVMMirrorSpec
https://wiki.eng.vmware.com/VMotionWithoutSharedStorage



Reference:
KL_SVMotion_CPD_Education.ppt
http://www.vmworld.com/docs/DOC-6289
http://www.vmworld.com/docs/DOC-2354

7 X-vMotion

7.1 Overview

Cross-host Storage VMotion enables a VM to change its datastore and host simultaneously, even if the two hosts don't have any shared storage in common. Another way of looking at this functionality is supporting VMotion without shared storage.

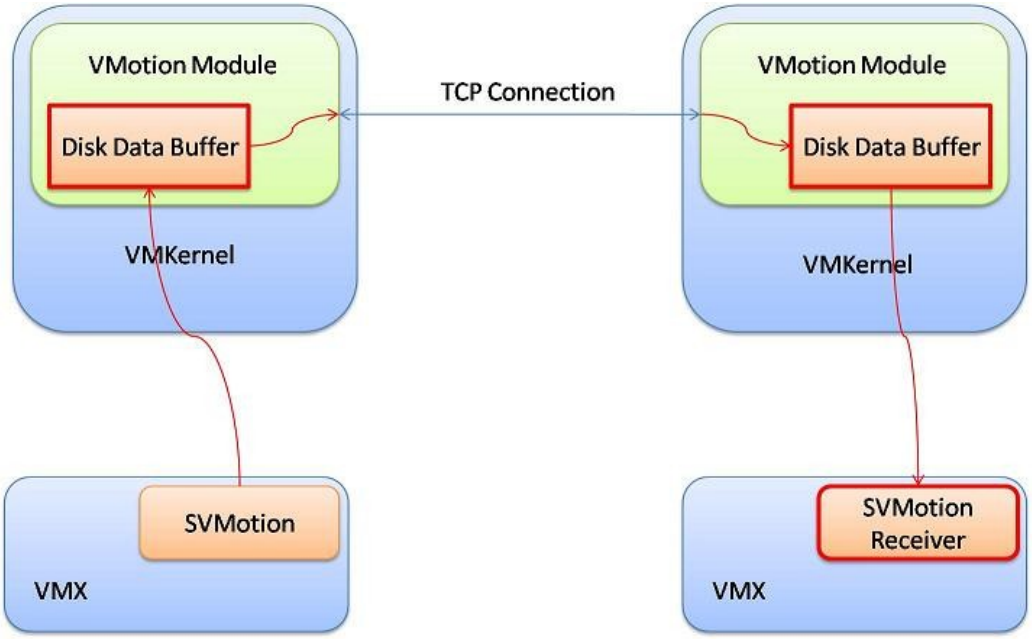
Cross-host Storage VMotion provides a new level of ease and flexibility for live VM migrations. It allows VM migration between clusters in a larger datacenter, which may not have a common set of datastores between them. In addition, by removing the shared storage requirement, it allows for simpler setup and use of local disks, which lowers the barrier to entry for use of VMotion and will thus be very useful for the SMB market. Finally, it is a step towards long-distance VMotion and can enable VM migration between clouds.

This technology essentially combines VMotion and Storage VMotion. The pre-copy phase now copies both the VM's memory and its disk(s) over the network to the destination host. Once it has removed the changed memory and disk data to an acceptably low amount, it will checkpoint the VM and send that data over. The destination VM will begin its resume and

simultaneously the rest of the changed memory and disk data will be sent over. When the changed data is received, the destination VM can finish its resume and the source VM will be powered off. Thus the process is very similar to VMotion or Storage VMotion. The specific differences are described below.

7.2 Design

The design diagram below shows the components that need modification. Everything in red is completely new code. The design works by having a data buffer on both the source and destination. The VMotion protocol will be augmented to support transferring data from this buffer across the network. We refactor the current Storage VMotion code that is in ESX 5.0 to support calling either the standard localhost functions it does today or in the case of a cross-host Storage VMotion a new VMKCall. In the prototype we are building a VMKCall to be issued by the source to transmit a chunk of data to the destination. The call returns asynchronously as long as the source buffer has space in it otherwise it will block. The VMotion module will transmit the data to the destination and into the destinations data block buffer. The destination VMX will sit blocked until it receives a new message in the buffer and will save that to disk. We send acknowledgments when buffer space on the destination is free to allow the source to send data and to know how much to send.



Reference:
https://wiki.eng.vmware.com/StorageVMotion/CrossHost2Spec