A Hybrid Approach To Live Migration Of Virtual Machines

by

Shashank Sahani, Vasudeva Varma

in

Cloud Computing in Emerging Markets (CCEM), 2012

Bangalore, India

Report No: IIIT/TR/2012/-1

Centre for Search and Information Extraction Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2012

A Hybrid Approach To Live Migration Of Virtual Machines

Shashank Sahni
Search and Information Extraction Lab,
International Institute of Information Technology,
Hyderabad, 500032
Email: shashank[dot]sahni[at]research.iiit.ac.in

Vasudeva Varma
Search and Information Extraction Lab,
International Institute of Information Technology,
Hyderabad, 500032
Email: vv[at]iiit.ac.in

***Abstract*—We present, discuss and evaluate a hybrid approach of live migrating a virtual machine across hosts in a Gigabit LAN. Our hybrid approach takes the best of both the traditional methods of live migration - pre and post-copy. In pre-copy, the cpu state and memory is transferred before spawning the VM on destination host whereas the latter is exactly opposite and spawns the VM on destination right after transferring processor state. In our approach, in addition to processor state, we bundle a lot of useful state information. This includes devices and frequently accessed pages of the VM, aka the working set. This drastically reduces the number of page faults over network while we actively transfer memory. Additionally, on every page fault over the network we transfer a set of pages in its locality in addition to the page itself. We propose a prototype design on KVM/Qemu and present a comparative analysis of pre-copy, post-copy and our hybrid approach.**

I. Introduction

Like electricity, companies dont have to worry about computing and storage infrastructure anymore. With the advent of Virtualization, its been possible to provision any size of infrastructure requirements on demand. Cloud providers like AWS, Google Compute Engine, Rackspace etc. provide IaaS (Infrastructure as a Service) on a pay per use basis. A major concern with such large scale deployments is efficient utilization of resources. Live migration [1] is emerging as an impressive technology to efficiently balance load and optimize VM deployment across physical nodes in a data center.

With the help of live migration, VMs on a physical node can be transferred to another without shutting down the system. This is useful in various scenarios.

- VMs on a failing physical node can be migrated to a healthy one.
- Idle VMs on a node can be migrated to others for optimizing resource utilization.
- VMs on a stressed physical node can be migrated across various nodes for load balancing.

Even with these features, live Migration hasnt been adopted by major cloud providers. AWS doesnt support live migration because of the time and performance overhead of the current methods. In simple terms, the current implementations hinder scalability, degrade performance and hence are not considered by IaaS providers.

In this paper, I propose a hybrid approach to live migration of virtual machines which will improve the time taken to migrate VMs and the performance overhead involved.

Section II will deal with traditional methods of live migration. I propose my method in Section III and present a prototype design in IV. Section V deal with an analysis and comparative study of different approaches.

II. Background

Before the dawn of Virtual Machines, process migration was a budding research field for handling load balancing and efficient resource distribution in a data center. Because of similar features and aims, a majority of the approaches and algorithms in live migration have been heavily motivated from process migration.

On a broad level, live migration process can be classified into two steps - (i) switching the control to the destination (ii) transferring the data(memory/disk) to the destination. The two most common methods of live migration can be easily differentiated based on different iterations of the steps mentioned above.

- pre-copy - Memory is transferred first and then the execution is transferred [1].
- post-copy - Execution is transferred first and then the memory [2].

A. *Pre-copy*

The approach behind pre-copy is to transfer the memory to the destination over a series of iterations. The newly written pages are transferred in each iteration and this process is repeated until either the limit on iteration reaches or the final data is too small for causing any network transfer overhead.

1) Memory and VCPUs are reserved on the destination host.
2) When the migration is issued, a check on page writes is initiated and all the RAM contents are transferred to the destination. This is the first iteration.
3) In the subsequent steps, only the pages that have been dirtied since the last iteration are transferred until the iteration limit is reached or the memory of dirty pages in an iteration is low enough.

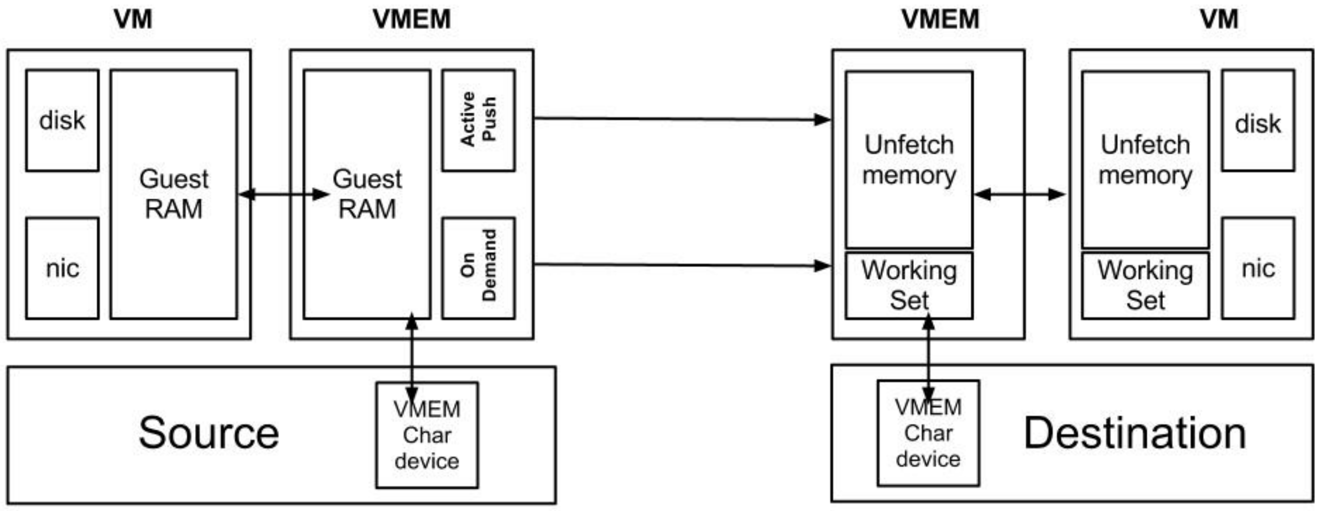

Fig. 1. Proposed prototype design.

4) The execution of VM on source is stopped and CPU state, registers, Virtual devices state and last memory pages are transferred to the destination.
5) VM is resumed at the destination.

Pre-copy is well proven for read intensive workloads but in case of write intensive VMs, large set of dirty pages result in performance degradation [1]. In worst cases, pre-copy wont even work. This is possible when the data write rate is more than the network bandwidth.

B. Post-copy

In contrary to pre-copy, post copy transfers the VCPU and devices state on the destination in the first step and starts the execution on the destination in the second.

1) Stop the VM at the source.
2) VCPU registers and states of devices are copied to the destination VM.
3) Resume the execution at the destination.
4) If the VM tries to access a not yet fetched page, then a network page fault occurs and the page is transferred to the destination(on-demand paging).

This is the most trivial form of post-copy. Network page faults indeed brings down the performance and increases total VM migration time. Various variations have been applied to post-copy to improve performance and reduce time. In addition to the on-demand paging, a bulk copy runs in the background which tries to transfer the memory as soon as possible. Adaptive prepaging is another technique where we intelligently identify the pages which may be accessed by the VM in near future and transfer them beforehand. This helps in quick transfer of the VMs working set.

III. Designing the hybrid approach

Both the traditional approaches defined above have their own weaknesses and strengths. Pre-copy doesnt perform well in write-intensive environment. It shows drastic increase in the total migration time. The no. of iterations may have an upper bound but total migration time is undetermined. It varies with the dirtying rate of pages.

On the other hand, post-copy doesnt perform well with read intensive loads. A read intensive VM will lead to an increase in the number of page faults over network, leading to a degraded response time. It witnesses a huge no. of network faults which incur significantly slow response time in the beginning. On the contrary, here the total migration time is determinable and is directly proportional to the total size of the RAM. This is considering that constant bulk memory transfer is enabled in the background.

A. Working Set

We propose an approach where in addition to transferring the VCPU registers and devices states in post-copy, we also include a small subset of memory which is frequently accessed by the VM. We can address this subset of memory as working set of the VM. This term is usually used in context of processes where it describes the physical pages in memory available to a process. Any addressing to a working set from now on will be with respect to a VM.

Working set in our case is defined as the subset of memory being used by the VM. If this subset is present on the destination when the execution is transferred, then the rate of number of network page faults will be very less as compared to post-copy approach. Due to this, the degraded response time witnessed during read intensive loads will be diminished/minimized.

1) Working Set Estimation: Plenty of work has been done in the field of VM cloning, provisioning and checkpointing that directly or indirectly leverage the benefits of a working set. Well introduce some of the interesting works done in the field and discuss their strengths and weaknesses.

2) VM Fork: Snowflock [3], a rapid VM provisioning system, made use of the post copy approach to spawn identical VMs on the fly. It introduces the concept of VM fork which is similar to process forking. The child VMs inherit the state of an existing parent VM. With various optimizations they were able to perform this stateful transfer and instantiation of virtual machines in sub-second intervals. Unlike process forking, vm forking is capable of spawning childs across different physical hosts making their approach interesting for live migration.

Like post-copy, VM fork works by pausing the source VM and transferring an initial state to the child(s). This initial state, however, isn't just vcpu registers and devices states. Its a condensed form of the VMs memory state, called VM descriptor. This descriptor is created by using paravirtualized guests. When the call is received VM kernel suspends all the I/O activity and deactivates all but one CPU. Once done, it makes a hypercall to the hypervisor issuing VM suspend operation. The suspended memory snapshot is transferred to the destination for execution.

A similar descriptor in case of post-copy can heavily optimize the performance. This condensed memory denotes the set of pages that were being used until the VM was suspended i.e. most frequently accessed pages or working set. With this on the destination VM at the time of execution will result in a significant reduction of page faults, hence better response time.

An obvious limitation of this approach is the dependence on paravirtualization. It produces efficient results because the guest is cooperating, but may not be an appropriate solution for public cloud providers who allow users to have complete control over their images.

3) VM Substrate: [4] describes another technique of working set estimation by the introduction of VM substrates. The aim was to provide a faster method of spawning multiple instances through re-designed VM templates. They minimized the size of the VM templates and kept them in RAM on every host for faster instantiation of VMs. One of their major challenges was reducing the memory footprint of these templates. Once the VMs are instantiated, rest of the memory can be fetched through lazy transfer or bulk prefetching.

They make use of selective checkpointing to identify the minimal subset of memory which will be a part of the substrate. The selection is primarily targeted on ignoring free pages and synchronizing disk caches to the disk, thus minimizing the size of the subset.

For selective checkpointing, they use Xens checkpointing feature which suspends the VM and dumps its entire memory and state in a checkpoint file. This checkpoint file is then used for substrate creation.

Their method cant be directly adopted for our hybrid design, since a frequently accessed disk cache would be considered a working set and shouldnt be synchronized to the disk. But it does provide an interesting insight of using checkpoints for working set estimation.

For VM substrate creation, they only use a single checkpoint. But this method cant be used for efficient working set estimation. For identifying a minimal subset, we need to find the frequently accessed subset of memory which cant be done with one checkpoint. Studying multiple checkpoints will help us render the pattern but the operational overhead is too much.

Hence, we took a different approach of not suspending the VM multiple times, instead study the memory access pattern to identify and mark the pages. Since, we are only interested in identifying the pages, creating multiple checkpoints is not required. Similar approach is adopted in [5]. They made use of access bit scanning to identify the working set.

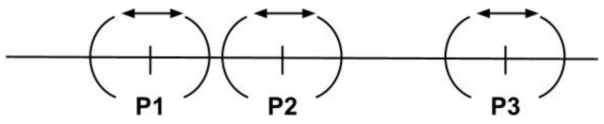

Fig. 2. P1, P2 and P3 represent the position of the pages faulted on the destination. Such requests are responded with an equal no. of pages transferred in both the directions, creating a bubble.

4) Access Bit Scanning: In hybrid live migration, access bit scanning will be the initial phase. We'll use the flags in the page table entries to study the access pattern. We'll perform multiple iterations of looping over the page table to mark most frequently accessed pages and come up with a working set in the end.

Determining the number of iterations is important. If we take a low value, we might not get the efficient result. On the other hand, a higher value may result in an increased preparation time resulting in higher total migration time without guarantee of increased efficiency. An optimum value needs to be addressed. This scenario is similar to pre-copy iterations followed by final transfer. We can apply their learnings here.

At the end of the preparation phase, we'll have a minimal image working set, cpu registers, states of the virtual devices etc. This will be transferred to the destination and the execution will be resumed. This phase is similar to the post-copy.

[2] and [6] describe various approaches for optimizing post-copy migrations. Well use a combination of them to optimize the page transfers and VM's response time in this phase.

- **On-demand Paging:** This method serves the page faults occuring over the network on the pages which have not yet been transferred. We can optimize on-demand paging to make a bulk transfer of the locality too. We can consider the position of the faulted page as a pivot and transfer equal no. of pages in both directions to the destination, in addition to the requested page. The process is briefly shown in figure 2.
- **Active Pushing:** This is a bulk transfer mechanism which keeps on transferring the memory sequentially to the destination host.

TABLE I
PROCEDURAL COMPARISON OF DIFFERENT MIGRATION SCHEMES.

	Preparation	Downtime	Resume
Pre-copy	Iterations and dirty memory transfer	CPU + dirty memory transfer	Reschedule VM
Post-copy (on-demand + adaptive prepaging)	Negligible Preparation time.	CPU + states of virtual devices	Bubbling + Page faults
Hybrid (Working set + optimized on-demand + active push + compression)	Iterations for scanning page table and identifying the working set.	CPU + states of virtual devices + Working set	Bulk locality transfer in Page faults + Active push + compression

- **Adaptive prepaging:** This was proposed by hines and makes use of the faulted page's locality information to predict and transfer pages which can be accessed by the destination VM in the near future. They use bubbling by considering the page's position as pivot and keep on transferring the pages in either direction, i.e. expanding the bubble. This can be considered as active pushing optimized for locality transfer.

If we compare optimized on-demand paging with adaptive prepaging, we're comparing bulk transfer of pages with solitary transfer of the same no. of pages. The latter incurs additional overhead for making multiple network calls. Hence, we'll adopt a combination of optimized on-demand paging and active pushing.

Compression: In order to decrease the transfer time well compress the memory pages while transfer, using the LZO algorithm. This will provide faster transfer rate because of memory compression, but will require additional computation on both source and destination. A major benefit of compression comes in case of zero pages, where the compression ratio will be very high.

An approach for live migration utilized in [7] suggests that compression and decompression introduces approx. 30% CPU overhead, but can be helpful in reducing 68% of the whole transferred data and 27% of the total down time. The figures may defer because of experimental setup and algorithm involved, but it shows the benefits. This feature can be kept optional due to extra computation requirements.

IV. PROTOTYPE PROPOSAL

The first prototype of post-copy was implemented in Xen. Building up on that work will require significant changes to the VMM, leading to slower and more testing to release a production level code. Hence, we will implement the prototype in KVM/Qemu. Figure 1 describes the proposed prototype design.

KVM stands for kernel virtual machine. It enables linux kernel to act as a hypervisor by the addition of a module. The module simply allows the QEMU emualted VMs to leverage the hardware virtualization support provided by Intel and AMD processors. Extending kvm/qemu will mean working on a kernel module or a device driver, easing out the development and testing procedure.

A. Implementation Prototype Design

Interesting work was done by Hirofuchi in implementing post-copy on Qemu/KVM. Their implementation relied mainly on an external character driver and made minor changes in the VMM. We plan to extend their work on QEMU for developing a proof of concept.

Hirofuchi's prototype included an additional char driver, VMEM, into the kernel which shares the qemu process's memory, i.e. VM's memory. Well make use of this char device to loop into page tables and scan the access bits of the pages. Once the working set is identified, the source VM is suspended and it is bundled with cpu registers and devices states for transfer to destination node.

They use threaded approach for handling post-resume jobs - on-demand paging and active pushing. We can use the same methods for including our optimizations.

We can implement our proposal, on-demand paging with locality transfer + active pushing, around their prototype model by making use of the VMEM char device for monitoring the memory and additional threads for performing the post-resume tasks - active pushing and on-demand paging.

V. COMPARISON AND ANALYSIS

For ease of comparison, let us divide the complete live migration into three different phases - preparation, downtime and resume. A comparison between the methods is shown in table I.

- **Preparation:** In this phase, appropriate resources are reserved on the destination VM and various operations are performed on the source which varies with different live migration approaches.
- **Downtime:** This is time during which the VM on the source host is suspended.
- **Resume:** This step involves the instantiation of VM on the destination host with the same state as suspended source VM. The time for the completion of this phase varies with different approaches.

The total time taken in completion of all these phases is called Total Migration time and the time taken in second phase is Downtime.

A comparative study was performed in [2] to describe and compare various elements in different approaches of migration.

We add our approach to the table I mentioning the methods used in each phase.

During the preparation phase, we collect the working set which includes pages which have been accessed for both read and write. Needless to say that our approach performs same for both read and write intensive workloads.

A. Hybrid v/s Pre-copy

Pre-copy transfers everything before resuming the VM on the destination host and hence has no extra overhead in the resume phase. A majority of its time is consumed in preparation. This period involves the transfer of complete RAM once, transferring dirty pages in every iteration and final transfer of writable working set (WWS).

Similar to post-copy, our method transfers the memory only once, not to mention the speed up due to compression of pages during transfer. Our working set involves pages accessed and hence will be greater in size than final WWS, but pre-copy iteratively copies multiple such sets which increases the total migration time.

Hence we can state that, for read-intensive workloads the performance of both will be comparable, but for write-intensive hybrid will perform better.

B. Hybrid v/s Post-copy

Post-copy only transfers the cpu registers and virtual devices states and therefore has the least downtime of all, in any scenario. Almost all of the time is spent in the resume phase where the memory is transferred to the destination.

Post-copy uses on-demand paging and adaptive prepaging to optimize this phase, but we present further optimizations by minimizing the page faults through working set, bulk locality transfer during a page fault and compression of pages while transfer.

For intensive workloads, time consumption in preparation phase increases but it helps reduce the response time at the destination in the final phase.

VI. Conclusion and Future Work

In this paper, we present a hybrid approach to live migration which takes the best from each of the traditional approaches - pre and post-copy. We further optimize it to design a reliable approach for live migrating virtual machines in a data center. Our evaluation shows that pre-copy and post-copy may perform well in certain scenarios but our approach is reliable and produces a comparable result. In scenarios where both fail, were still able to ensure good performance. We now aim to develop a proof of concept of our design and test with various solutions and compare the relative performance. Weve presented a high level design of the prototype, extending the post-copy approach implementation in KVM/Qemu.

References

[1] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[2] M.R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 51–60.

[3] H.A. Lagar-Cavilla, J.A. Whitney, A.M. Scannell, P. Patchin, S.M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 1–12.

[4] K. Wang, J. Rao, and C.Z. Xu, "Rethink the virtual machine template," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2011, pp. 39–50.

[5] I. Zhang, A. Garthwaite, Y. Baskakov, and K.C. Barr, "Fast restore of checkpointed memory using working set estimation," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2011, pp. 87–98.

[6] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Enabling instantaneous relocation of virtual machines with a lightweight vmm extension," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 73–83.

[7] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. Ieee, 2009, pp. 1–10.