Live Gang Migration of Virtual Machines

Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan
Computer Science, State University of New York, Binghamton, NY

1 Introduction

Administrators of virtualized clusters often need to simultaneously migrate a group of co-located virtual machines (VMs), i.e. VMs located on the same physical machine. For example, simultaneous live migration of VMs may be needed before shutting down a physical machine for maintenance, or to perform load-balancing to handle sudden spikes in load. To reduce the performance impact of migration on applications running in the VMs, the total migration time, network traffic overhead, and service downtime should be kept minimal.

State-of-the-art VM migration techniques [2, 4] optimize the migration of a single VM. However these techniques alone are insufficient when tens or hundreds of VMs need to be migrated simultaneously, possibly at a moment's notice. For example, migrating one 16GB idle VM over an uncongested Gigabit network can take more than 2 minutes. Live migration code in QEMU/KVM environment [1] applies simple optimizations such as compressing pages that contain uniform data. Even then, a 16GB VM can still take up to half a minute to migrate. Needless to say, migrating multiple VMs simultaneously can take several minutes of critical time during which the network becomes overloaded and application within the VM suffer from degraded performance.

We present a new approach, called *live gang migration*, that can be used in conjunction with traditional single VM migration techniques to reduce the overhead of simultaneously migrating multiple co-located VMs. Live gang migration is based on the observation that co-located VMs often have significant commonalities, such as the same operating system, application packages, libraries, etc. This leads different VMs to have significant number of identical memory pages. Traditional techniques for single VM migration would independently transfer the entire contents of each page, despite many pages having identical content. In contrast, live gang migration pro-actively detects and tracks the presence of identical pages across co-located VMs, as also within each VM. When migrating one or more VMs, identical pages are transmitted only once, instead of multiple times.

We have implemented a prototype of live gang migration in the QEMU/KVM virtualization platform. Evaluations over Gigabit Ethernet show that gang migration of 24 VMs can achieve more than 60% reduction in network traffic and 40% reduction in total migration time. Our technique also reduces the total migration time of a single 16GB VM by 47%.

The idea of tracking identical pages across co-located VMs has been proposed earlier [7, 3, 5] to reduce memory pressure within a machine and to facilitate higher degree of consolidation. Work in [6], which optimizes single VM migration over slow wide-area networks, uses hashing to avoid sending disk blocks that already exist at the target. To the best of our knowledge, our work is the first to exploit identical memory pages across multiple co-located VMs to improve simultaneous migration of multiple VMs over Gigabit Ethernet.

2 Design and Implementation

In this section, we present the design and implementation of live gang migration in the context of QEMU/KVM virtualization platform. However the basic concepts are applicable to other virtualization platforms as well. The key idea behind live gang migration is to identify and track all identical pages among multiple co-located VMs. As shown in Figure 1, migration controllers at both the source and the target nodes initiate and control live gang migration. The controller sets up a shared memory region that is used to track identical pages among co-located VMs. Live gang migration has two phases, namely *preparation* and *migration*.

Preparation: In this phase, the controller periodically scans the memory of co-located VMs to identify identical pages using the well-known hashing approach. We compute a 32-bit hash value for each page. If the hash value matches a page already stored in shared memory then we proceed to perform a byte-by-byte comparison to confirm that the pages are indeed identical. (More bits in the hash value can further reduce memory comparison overhead). A hash table in the shared memory is used to track identical pages and hash buckets resolve collisions. A $(hash, index)$ pair is used to track the hash value and the position in hash bucket for identical pages in each VM. Contents of a page can change after a scan due to memory writes by VMs. We enable dirty page logging that marks the scanned pages as read-only, traps the first write attempt to each page, and updates a dirty bitmask that records pages dirtied since the last scan.

Migration: Live gang migration transfers only one instance of every identical page across VMs. During migration, each page of a VM is handled in one of the following four categories: (1) *Unique Pages*: These pages are not identical to any other page and their contents must be transferred during migration. (2) *Dirtied Identical Pages*: These pages were identified as identical to others during the preparation phase, but have been dirtied by the VM since then. Currently, we assume that such dirtied pages are unique and transfer the entire contents of such pages. However, one could potentially extract marginal performance gains by re-hashing and re-matching these pages once more. (3) *Unsent Identical Pages*: These are identical pages have not been dirtied since the last scan and not yet been transferred to the target. Again, the entire page content is transferred and accompanied by the $(hash, index)$ identifier so that the page can be uniquely identified and stored by the controller at the target. (4) *Sent*

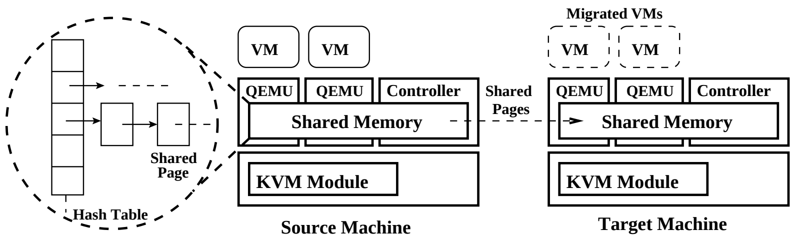

Figure 1: Architecture of live gang migration of VMs in QEMU/KVM.

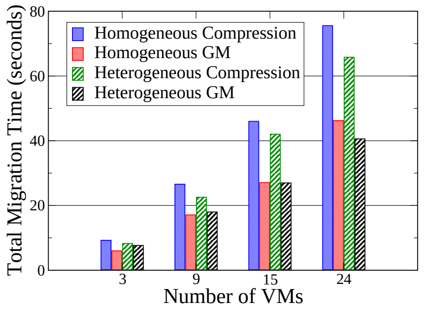

Figure 2: Reduction in TMT.

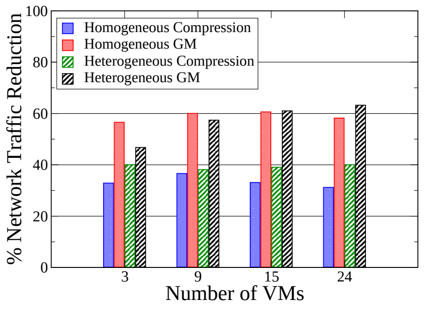

Figure 3: Reduction of network traffic overhead

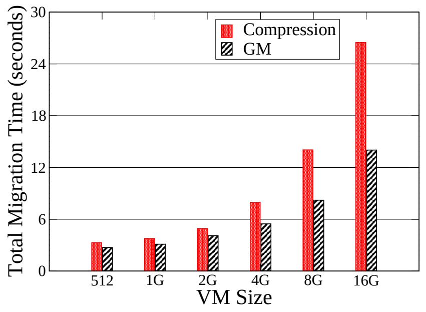

Figure 4: Reduction in TMT for single VM.

Identical Pages: These are identical non-dirtied pages that have already been transferred to target. For such pages we transfer only *(hash, index)* identifier to locate the corresponding page at the target and omit transferring the page content.

We implemented a prototype of live gang migration in QEMU/KVM environment, using Linux 2.6.32 and qemu-kvm-0.12.3 on source and target machines. Our implementation consists of around 1600 lines of code. As mentioned earlier, QEMU/KVM also performs a rudimentary compression of pages that have uniform content. Our implementation of live gang migration replaces this compression phase with code to eliminate duplicate transfers.

3 Evaluation

Here we compare the performance of our implementation of gang migration against the original QEMU/KVM live migration mechanism. The key performance metrics are total migration time (TMT) and the amount of network traffic reduction. Live gang migration has no effect on the downtime of VMs during migration since it augments, not replaces, the basic pre-copy migration strategy in QEMU/KVM. Our experimental setup consists of two machines with dual quad core Intel Xeon processors and 70GB DRAM. We used mix of Ubuntu 9.10, Fedora 12, and Windows XP VMs for heterogeneous tests, and Ubuntu 9.10 for homogeneous tests.

Fig 2 shows that live gang migration of multiple 512MB VMs significantly reduces TMT compared to original QEMU/KVM migration. As the number of VMs being simultaneously migrated increases, the performance gains also increase, with up to 40% reduction in TMT for 24 VMs. Fig 3 compares the reduction in network traffic due compression in original QEMU/KVM migration against elimination of duplicate transfers in live gang migration for multiple 512MB VMs. In most cases, live gang migration achieves almost 60% of reduction in the network traffic against 30 to 40% reduction with just rudimentary compression. Finally, we also observe a significant number of identical pages within a single VM. Fig 4 shows that, even for a single VM, with increasing VM size, elimination of duplicate page transfers yields much lower TMT than compression, and the performance gains increase with larger VM sizes.

4 Future Work

In this paper, we have presented our initial design of live gang migration of VMs and preliminary performance results. Our future work includes more extensive performance evaluations using a range of VM workloads. We are also investigating the benefits of parallelizing both compression and de-duplication of data transfer during migration on multi-core machines. We also plan to investigate the benefits of de-duplicating transfers at the sub-page level, similar to the approach in [3].

References

[1] A. Kivity, Y. Kamay, et. al. KVM: The Linux virtual machine monitor. In *Linux Symposium*, June 2007.

[2] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[3] D. Gupta, S. Lee, et. al. Difference engine: Harnessing memory redundancy in virtual machines. In *OSDI*, 2008.

[4] J.G. Hansen and E. Jul. Self-migration of operating systems. In *Proc. of ACM SIGOPS European Workshop*, 2004.

[5] G. Milos, D.G. Murray, S. Hand, and M.A. Fetterman. Satori: Enlightened page sharing. In *USENIX Annual Technical Conference*, 2009.

[6] C. Sapuntzakis, R. Chandra, and B. Pfaff. Optimizing the migration of virtual computers. In *Proc. of OSDI*, 2002.

[7] C.A. Waldspurger. Memory resource management in VMware ESX server. In *Proc. of OSDI*, 2002.