Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications

Aidan Shribman[1] and Benoit Hudzia[2]

[1] SAP Research, Ra'anana, Israel
aidan.shribman@sap.com
[2] SAP Research, Belfast, UK
benoit.hudzia@sap.com

Abstract. Virtualization technology provides a means for server consolidation, reducing the number of physical servers required for running a given workload. Virtual Machine (VM) live migration facilitates the transfer of a running VM between physical hosts while appearing transparent to the running application. Memory intensive applications tend to obstruct the original pre-copy live migration process and may result in the failure of the migration process due to its inability to transfer memory faster than memory is dirtied by the running application. The focus of this paper is to present several techniques that can be applied to both pre-copy live migration and post-copy live migration to better support migration of memory intensive applications.

Keywords: Operating Systems, Hypervisors, Virtual Machines, Live Migration, Pre-Copy, Post-Copy, RDMA, QEMU, Linux/KVM.

1 Introduction

Virtualization is widely used in enterprise data centers as a means to reduce operational costs and increase operational flexibility. Adoption of this technology has surged following recent advances in x86 architecture such as multi-core and introduction of hardware assisted x86 extensions supporting full virtualiztion. Live migration is a core capability of modern hypervisors helping increase operational flexibility by serving as a foundation for capabilities such as High Availability (HA), Disaster Recovery (DR) and dynamic policy based migration of workloads for reducing power consumption in data center.

Live Migration Approaches. The live migration process handles the transfer of memory, CPU and hardware device state of the running VM. While the CPU and devices state are relatively small, typically in the size order of several KBs, size of memory may be in the order of many GBs. 1 GB of DRAM requires about 10 seconds to be transferred over a 1 GbE interconnect while 1 KB could be transfered over a 1 GbE in only 10 microseconds. Live migration schemes differ according to the order of state transfer: **Pre-Copy Live Migration:** Implemented in most hypervisors such as VMWare [4], Xen [1] and KVM

I. Caragiannis et al. (Eds.): Euro-Par 2012 Workshops, LNCS 7640, pp. 539–547, 2013.

[2], presented by Brandford at el. [5] is both simple in design and does not require a fast interconnection between physical hosts. Memory is transfered to the destination host in a succession of iterations until the remaining dirty memory can be transferred in a short enough *stop and copy* phase which will not cause prolonged VM downtime. **Post-Copy Live Migration:** CPU and device state are transfered immediately to the destination host followed by transfer of execution control to the destination host. Source host memory is transfered in the background or fetched on-demand if needed by the running VM on the destination host. This migration scheme reduces the downtime and total migration time but incurs service degradation due to page faults which must be resolved over the network by the source host. **Hybrid Post-Copy Live Migration:** Adds a bounded pre-copy phase before entering the post-copy phase. The additional pre-copy phase reduces the number of future network bound page faults as a large portion of the VM memory is already pre-copied.

Live Migration Metrics. We evaluate the various live migration schemas by using the following metrics: **Downtime:** Is the amount of time in which the VM is suspended on the source host and until it becomes available again on the destination host. **Total Migration Time:** The overall time elapsed from the initiation of the live migration operation and until all resources on the source host are released. **Total Transfered Bytes:** The total number of bytes transfered over the network from the source host to the destination host. **Service Degradation:** The degree at which the VM's operation was slowed down due to the live migration process compared to uninterrupted VM execution.

Our Contribution. Is the design and implementation of several improvements to live migration to better support memory intensive applications:

Pre-Copy Live Migration: The fast memory dirtying rate of memory intensive applications relative to available network bandwidth challenges the pre-copy live migration scheme. In this paper we propose using a page reordering policy such that Least Recently Used (LRU) pages, with lower chances of being re-dirtied, are migrated earlier. Additionally, we propose using a fast delta encoder, Xor Binary Zero Run Length Encoding (XBZRLE), which reduces the cost of a page re-send.

Post-Copy Live Migration: Post-copy live migration has two major advantages over pre-copy live migration: (1) shorter downtime as only CPU and device state, several KB in magnitude, need to be migrated while the VM is stopped; (2) total migration time is short and deterministic as pages are not re-dirtied on the source host during live migration. We mitigate the challenge of service degradation by several means: (1) a Remote Direct Memory Access (RDMA) stack for low-latency resolution of network-bound page faults; (2) demand pre-paging (a form of pre-fetching) for reducing the overall number of page faults; (3) integration of page faulting mechanism with the Linux Memory Management Unit (MMU) so that only the thread waiting on the page fault is paused while other threads continue execution; (4) pre-copy post-copy hybrid live migration scheme which helps further reduce the number of page faults when entering the post-copy phase.

2 Design and Implementation

This work introduces modifications to QEMU, originally developed by Fabrice Ballard [3] and KVM originally developed by Avi Kivity [2] on the Linux OS. Our enhancements are to be released as open source under GPL and LGPL licenses. A XBZRLE patch [8] was released during 2011 and is in the process of being integrated into the mainline QEMU code base.

2.1 Least Recently Used Page Reordering

Pre-copy page re-ordering can potentially reduce the total number of pages sent over the network by reducing the number of page re-sends. Checconi at el. [6] have proposed using LRU to dictate the order in which pages are transfered over the network to the destination host. In certain applicative workloads using such a policy may help reduce the page re-dirtying rate and hence reduce page re-sends. In turn downtime and total migration time are much reduced. In our implementation we propose subtle enhancements to the existing work:

QEMU constructs and maintains the LRU order according to when the VM pages were dirtied by the running application. Page dirty events are tracked by KVM in the Linux kernel using a dirty bitmap array. In each observation interval the dirty bitmap array indexed by page virtual address is cleared. The dirty bit is set for all pages dirtied in the observation interval. At the end of the observation interval there is no way of knowing what is the chronological order in which new pages were dirtied. Assuming address-based order (or any other order) would cause a bias in our LRU ordering we therefore propose using Fisher-Yates shuffle [18] to randomize dirty page order events before updating the LRU order.

As the pre-copy live migration scheme transfers memory in consecutive iterations we can use the LRU ordering not only to determine the order in which pages are sent within the interval but also for determining which pages need not be sent at all in the current interval as they will probably be re-dirtied again soon. Any pages at the tail of the LRU, namely the Most Recently Used (MRU) pages, are pages we do not send in the current interval. In our implementation we used an empiric value: the first 75 % LRU pages are transfered; the remainder 25 % MRU pages are not transferred. A more self-tuning approach for determining what is the percentage of pages to be sent in each interval may better suite real applicative workloads.

2.2 XBZRLE Delta Encoder

Reducing the number of page re-sends, via LRU page reordering, is speculative in nature and may have an adverse effect if future memory dirtying pattern can't be derived from past memory dirtying pattern. A different approach to optimizing pre-copy live migration is to reduce the cost of each page re-send by using delta encoding for a compact representation of page updates. General purpose delta compression algorithms such as ZDelta are relatively slow reaching rates of about 50 MB/s [10]. Hudzia et el. [7] proposed the use of Xor Binary Run Length Encoding (XBRLE) for delta encoding of dirty pages for pre-copy live migration which reached rates of about 2 GB/s but had a relativly low compression ratio.

In this paper we present XBZRLE which provides a more compact encoding representation than XBRLE while additionally improving encoding speed. XBZRLE is comprised of two phases: (1) binary XOR for encoding the delta between the old memory page content and the new memory page content - XOR is both fast and is able of capturing the in-place memory changes - represented by the XOR as a sequence of non-zeros; (2) Zero Run Length Encoding (ZRLE) attempts to compress only the zero runs using Run Length Encoding (RLE) but does not attempt to encode non-zero runs. (as non-zero sequences have little chance for character repetition). Both phases XOR and ZRLE can utilize CPU Single Instruction Multiple Data (SIMD) instructions taking advance of multi-byte parallelism by using input and output characters of 8-bytes (=64 bit) rather than executing on byte sized characters. The XBZRLE delta encoder requires as input both the new page contents and the old page contents. The old page contents is cached each time a page is sent to the destination host. We currently use a rudimentary LRU cache but plan in future to use an advanced self-tuning cache such as ARC [11] or CAR [12] which would provide a higher hit-ratio without any increase in cache size. Cache size is determined by optimizing both cache hit-ratio and cache memory overhead incurred on host - for most workloads a 10 % of the VM memory size allocated for the cache - should provide ample hit-ratio.

In [9] we presented comparative results for both encoding speed and encoding size of various page encoders. We evaluated several encoders using 4 scenarios: sparse memory update pattern (step 1111 bytes alter 12 bytes), medium (step 701 alter 33), dense (step 203 alter 41) and very dense (step 121 alter 43). In order to compare compression algorithms candidates we first run a XOR phase (for getting a non compressed delta representation), following we run our candidates: LZO (xblzo), Google Snappy (xbsnappy), RLE (xbrle) and ZRLE (xbzrle). Results demonstrate that XBZRLE outperforms the other candidates in encoding speed achieving a 2200 MB/s compared 200 MB/s for byte-wise XBRLE, and 500 MB/s - 1200 MB/s for xblzo and xbsnappy. While fast - XBZRLE - does incur a 50% increase in compressed size compared with the slower xblzo and xbsnappy encoders.

2.3 Post-Copy Live Migration

In this subsection we present our implementation of post-copy live migration of VMs across an RDMA interconnect. While post-copy live migration is not new, our QEMU and KVM based implementation has incorporated several innovations which make it especially capable of handling memory intensive applications.

Core to a high performing post-copy implementation is the efficient resolution of network bound page faults - which has also been a relative barrier to implementing post-copy live migration in general.

MMU Integration: Hirofuchi and Yamahata in Yabusame project [17] disclose post-copy enhancement of QEMU/KVM using a character device and page faulting transport via user mode. Our approach was to extend the Linux MMU such that network bound page faults are handled directly in the kernel much like swap in of page from a disk-based swap device incurring no context switches into user mode. By introducing a new flag in the *page table entries* we indicate that the page is backed by memory residing on a remote host. While Yabusame causes a full VM freeze during remote page fault - our implementation only causes the requesting thread to pause while all other VM threads continue uninterrupted.

RDMA Interconnects: The concept of remote DRAM backed swap was proposed by Comer and Griffioen [15], however, only the recent advancement in network interconnects performance and accompanying standardization in the RDMA stack have enabled to unleash its power. Especially notable is OFED and its promotion of standard RDMA semantics for all underlying hardware and software stacks such as InfiniBand, IWARP and RDMA over Converged Ethernet (RoCE). While 40 Gbps InfiniBand and 10 Gbps Ethernet provide excellent performance it is also possible to use a pure software stack such as softiWARP atop 1 GbE making post-copy live migration feasible in many cost-conscious environments without the need for up-front hardware expenses. While RDMA is an enabler to post-copy live migration it can also benefit pre-copy live migration as presented by Huang at el. [13].

Demand Pre-paging: Was first presented by Kaplan at el. [14] who demonstrated that by altering the amount of pre-fetched pages according to the load characteristics reduced the percentage of page faults in specific scenarios while not incurring increased service degradation in other scenarios. By adding a simple pre-fetcher that demands the 40 pages surrounding the faulted page we managed to reduce service degradation. Hines at el [16] demonstrated the effectiveness of pre-paging for reducing network-bound page faults to be within 21% of the VM working set size.

Hybrid Post-Copy: We additionally implemented hybrid post-copy allowing for a pre-defined bound pre-copy phase to proceed the post-copy phase as a means for reducing the number of future page faults.

3 Evaluation

While initially considering realistic applications such as the SAP Sales & Distribution Benchmark as used by Hudzia et al [7], it became apparent that these may vary in behavior greatly between runs thus making comparative testing of live migration schemes difficult. We thus focused on artificial benchmarks which may not fully represent real memory intensive applications but are highly deterministic in behavior.

Pre-Copy Evaluation. We built a synthetic multi threaded workload generator *appmembench*, which dirties different groups of pages in different frequencies using 8 threads, a working set of 512 MB, and a workload characterized by a memory dirtying rate of 1 GBps (adding the full 4 KB page size as dirty memory for each page partially dirtied). Workload was run on a guest VM of: 4 x vCPU; 1 GB RAM; Each physical host with 2 cores, 8 GB RAM, while hosts were interconnected with 1 Gbps Ethernet network, we limitted network bandwidth to only 30 MB/s or 240 Mbps to maintain a high *application-dirty-rate / network-transfer-rate* ratio - characteristic to real memory intensive workloads.

Figure 1 compares three pre-copy live migration implementations: (1) the original QEMU implementation - 'original'; (2) Page LRU reordering modification - 'prio'; (3) XBZRLE modification - 'xbzrle'. The chronological sequence of events presented in the chart: 0 seconds - start running appmembench; 30 seconds - start live migration; 150 seconds - force migration (by setting a high max downtime value); 180 seconds - end of measurements.

The original live migration incurs a short downtime right after live migration initiation at 30 seconds, continues to work without any degraded service, but does not manage to complete the migration and transfers control only after 150 seconds and in the process sustaining a 6 seconds downtime. XBZRLE causes a severe service degradation - indicating that the application and XBZRLE are competing on limited compute resources - but it succeeds in completing the live migration after 105 seconds. (Although due to degraded service level XBZRLE had to cope with a much lower rate of page dirtying than the original live migration which did not cause service degradation). Finally LRU page reordering caused a 50% service degradation, also due to CPU overcommit, and only migrated after 150 seconds but has a slightly shorter downtime of about 4 seconds compared with the original implementation.

The appmembench benchmark ran 8 threads on 4 vCPUs atop 2 physical CPU cores, causing severe CPU pressure. As both XBZRLE and LRU page reordering use compute to mitigate network bandwidth strain the result in our case was an adverse effect on overall system performance due to competition between the running VM and the live migration process itself. Also highlighted by this test is the absence in our PoC of an automatic disabling mechanisms during CPU pressure.

Post-Copy versus Pre-Copy. So as to highlight post-copy's ability to cope with very high workloads we use Google Stress Application Test (SAT) - a scalable high-performance benchmark. Google SAT performs large bit-wise inverse operations which may not fully represent the types of operations done by real applications but creates a high multi-threaded workload which can saturate physical resources. For our tests we used the following guest VM specification: 2 x vCPU ; 1, 2, 4, 8, and 16 GB of memory; Google SAT benchmark with 1 GB working set; 1 Gbps Ethernet Network between hosts.

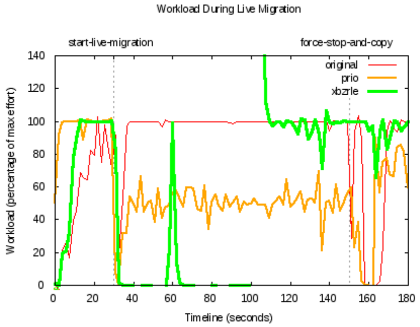

Fig. 1. Pre-Copy Evaluation

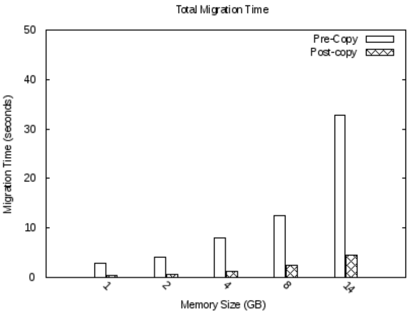

Fig. 2. Post-Copy vs. Pre-Copy

Figure 2 depicts total migration time for both pre-copy original implementation and post-copy live migration for various VM sizes. Post-copy live migration completed the live migration in about 10% of the time required for the original pre-copy implementation, indicating that the original pre-copy sends page updates over and over again in average 10 times each compared to post-copy which sends each memory page exactly once.

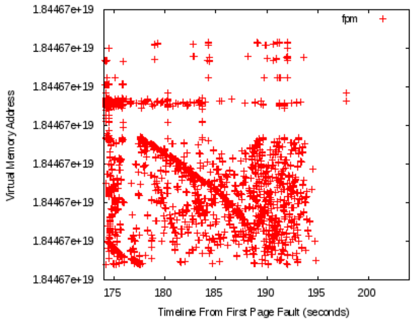

Fig. 3. Full Post-Copy

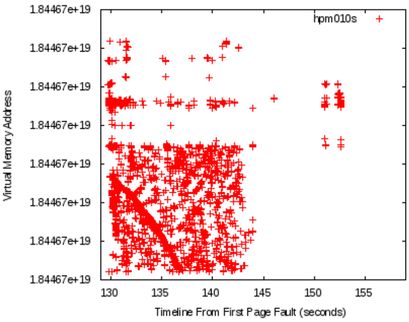

Fig. 4. Hybrid Post-Copy

Hybrid Post-Copy Evaluation. We used the following guest VM specification: 2 x vCPU ; 4 GB of memory; 1 GB Google SAT working set; 1 Gbps Ethernet Network between hosts. Network bound page faults are the main cause of degraded service right after transfer of control to the destination host and therefore their pattern and timespan can help assess the severity of this period. In figures 3, 4, individual network bound page fault events were recorded: Using virtual address as y coordinate and using relative time from first page fault as x coordinate. The figures depict that the pre-copy phase helps reducing page faults induced application service degradation - as can be seen when comparing full post-copy figure 3 versus hybrid post-copy 10s (a 10 second pre-copy phase) figure 4. The contribution of the pre-copy phase is nonetheless finite and does not go beyond a certain point as was determined when comparing our results for the hybrid post-copy 10s case to hybrid post-copy 20s and hybrid post-copy 40s.

4 Conclusions and Future Work

Migrating large memory intensive workloads challenges the existing live migration implementations. In this work we presented our enhancements and optimization over the original pre-copy live migration KVM QEMU implementation. We presented XBZRLE and LRU page reordering which support live migration of applications with high memory dirtying rate relative to available network bandwidth. Next we disclosed our enhanced post-copy live migration implementation which by subsuming demand pre-paging, fast RDMA interconnects, MMU integration and hybrid post-copy can provide a fast and predictable live-migration over fast interconnects maintaining minimal page fault incurred service degradation. In the future we plan extending our work on live migration to handle transfer of CPU and memory state between a cluster of hosts serving as a single **Virtual Distributed Shared Memory** system.

Acknowledgments. This work was funded by grant from the Invest Northern Ireland regional economic development agency covering work done by John Stuart, David Leib and Benoit Hudzia. Additional funding supported by the EU FP7 project VISION Cloud covering Aidan Shribman's work. We thank Chaim Bendelac and Eliezer Levy for their invaluable feedback.

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, New York (2003)
2. Kivity, A.: KVM: Kernel-based Virtualization Machine, `http://www.linux-kvm.org`
3. Ballard, F.: QEMU: Open Source Processor Emulator, `http://wiki.qemu.org`
4. VMWare Inc.: VMWare, `http://www.vmware.com`
5. Bradford, R., Kotsovinos, E., Feldmann, A., Schiberg, H.: Live wide-area migration of virtual machines including local persistent state. In: VEE 2007 Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 169–179 (2007)
6. Checconi, F., Cucinotta, T., Stein, M.: Real-Time Issues in Live Migration of Virtual Machines. In: Lin, H.-X., Alexander, M., Forsell, M., Knüpfer, A., Prodan, R., Sousa, L., Streit, A. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 454–466. Springer, Heidelberg (2010)
7. Hudzia, B., Svard, P., Tordsson, J., Elmroth, E.: Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines. In: VEE 2011 Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 111–120 (2011)

8. Shribman, A., Hudzia, B., Svard, P.: PATCH v4: XBZRLE delta for live migration of large memory apps, http://patchwork.ozlabs.org/patch/108868/
9. Shribman, A., Hudzia, B., Svard, P.: Microbenchmarks of page delta encoders, http://lists.gnu.org/archive/html/qemu-devel/2011-08/msg00214.html
10. Memon, N., Suel, T.: The zdelta-2.0 experimental results, http://cis.poly.edu/zdelta/results.shtml
11. Megiddo, N., Modha, D.S.: ARC: A Self-Tuning, Low Overhead Replacement Cache. In: FAST 2003 Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 115–130 (2003)
12. Bansal, S., Modha, D.S.: CAR: Clock with Adaptive Replacement. In: FAST 2004 Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 187–200 (2004)
13. Huang, W., Gao, Q., Liu, J., Panda, D.K.: High performance virtual machine migration with RDMA over modern interconnects. In: CLUSTER 2007 Proceedings of the 2007 IEEE International Conference on Cluster Computing, pp. 11–20 (2007)
14. Kaplan, S.F., McGeoch, L.A., Cole, M.F.: Adaptive Caching for Demand Prepaging. In: ISMM 2002 Proceedings of the 3rd International Symposium on Memory Management, pp. 114–126 (2002)
15. Comer, D.E., Griffioen, J.: A New Design for Distributed Systems: The Remote Memory Model, http://docs.lib.purdue.edu/cstech/830
16. Hines, M.R., Gopalan, K.: Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning. In: VEE 2009 Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 51–60 (2009)
17. Hirofuchi, T., Yamahata, I.: Yabusame: Postcopy Live Migration for QEMU/KVM. In: KVM Forum 2011 (2011)
18. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research, 3rd edn., pp. 26–27. Oliver & Boyd, London (1938, 1948), OCLC 14222135