Quick Eviction of Virtual Machines Through Proactive Live Snapshots

Dinuni Fernando*, Hardik Bagdi*, Yaohui Hu*, Ping Yang*, Kartik Gopalan*, Charles Kamhoua†, Kevin Kwiat†
* Computer Science Dept., Binghamton University, Binghamton, NY, USA
Email: {dferna15,hbagdi1,yhu15,pyang,kartik}@binghamton.edu
† Air Force Rome Research Laboratory, Rome, NY, USA
Email: {charles.kamhoua.1, kevin.kwiat}@us.af.mil

***Abstract*—Cloud computing platforms routinely use virtualization to improve service availability, resiliency, and flexibility. Live migration of Virtual Machines (VM) is a key technique to quickly migrate workloads in response to events such as impending failure or load changes. Despite extensive research, state-of-the-art live migration approaches take a long time to migrate a VM (in the order of tens of seconds for moderately sized VMs) which in turn negatively impacts the application performance during migration. We present, Quick Eviction, a new approach to significantly speed up the eviction of a VM from the source host with low impact on VM's performance during migration. Our approach can also improve the effectiveness of VM resilience tools that back up a VM's state in anticipation of system failures. The insight behind Quick Eviction is that majority of time to evict a VM during migration is due to the transfer of memory contents over the network, sometimes repeatedly. Before migration, Quick Eviction regularly snapshots the VM's memory to a destination or a failover node. During the actual migration, Quick Eviction has to transfer only a small amount of dirtied memory resulting in a very short time to completely evict the VM out of the source. The key challenge is to dynamically adapt the snapshot intervals so as to have minimal impact on application performance during migration. Our implementation of Quick Eviction in the KVM/QEMU platform reduces the eviction time of a read-intensive VM by a factor of 15 and that of a write-intensive VM by a factor of 350 compared to traditional pre-copy algorithm.**

I. Introduction

Virtualization plays an important role in cloud computing by enhancing server consolidation, reducing operational costs, and providing isolation. Live migration of a virtual machine (VM) refers to the transfer of a running VM's state from one physical machine (the *source*) to another physical machine (the *destination*). Live VM migration helps to decouple applications running inside a VM from the hardware on which the applications execute, and hence is crucial for activities like failure resilience, load balancing, server maintenance, and energy savings.

The two pre-dominant live migration techniques, namely pre-copy[4] and post-copy[12], and their variants have primarily focused on reducing two key metrics – the *total migration time* and *downtime* (with secondary focus on metrics such as application performance and network overhead). The total migration time is the duration between the start and the end of migration, whereas the downtime is the duration for which the VM is completely paused during migration. Another related metric, called the *eviction time*, refers to the time to evict a VM's state from the source machine [9, 11]. Eviction time may differ from total migration time when, under certain situations, an intermediate host is used to stage the VM's memory during migration. While downtime has been extensively optimized, there is considerable room for improvement in both total migration time and eviction time. Below, we use the term eviction time without loss of generality.

All existing live migration techniques have a key limitation that the eviction time is determined by the total memory size of the VM being migrated. High eviction time may increase the response time to impending failures or degrade the performance of applications running inside the VMs.

In this paper, we present *Quick Eviction*, a technique for reducing the eviction time of a VM with low impact on its performance. The basic idea is to regularly transfer incremental snapshots of the VM to another machine over the network. Upon a triggering event, such as an imminent failure of the source machine, any residual memory of the VM that was updated since the last snapshot is evicted to the destination. The eviction time thus achieved is small, since the size of the final residual memory is generally a fraction of the total memory size of the VM.

Quick Eviction is designed to satisfy the following requirements. First, Quick Eviction should significantly reduce the time to completely transfer the VM out of the source machine, while maintaining a low downtime during migration. Secondly, Quick Eviction should not excessively increase the network load due to snapshot operations. Thirdly, Quick Eviction should not significantly impact the performance of applications inside the VM. To achieve these goals, we develop a technique to dynamically vary the snapshot intervals based on the threshold rate of the pages dirtied (written to) by a VM during its execution, which lowers the network overhead and performance impact of snapshots.

Approved for Public Release; Distribution Unlimited : 88ABW-2016-2139 20160428

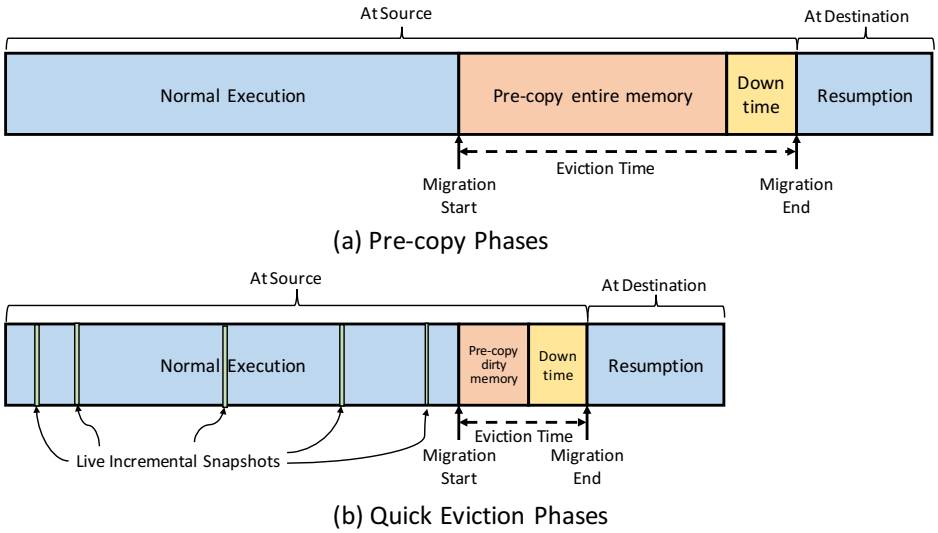

Fig. 1. Phases of live VM migration using (a) Pre-Copy and (b) Quick Eviction.

We have implemented Quick Eviction in the KVM/QEMU virtualization platform and evaluated its effectiveness using a number of benchmarks. Our experimental results show that our Quick Eviction speeds up live migration by a significant factor while maintaining downtime comparable to pre-copy, low impact on application performance, and low network overhead.

The rest of the paper is organized as follows. Section II provides an overview on the pre-copy live migration. Sections III and IV present the design and implementation of Quick Eviction, respectively. Section VI presents related work and Section VII concludes the paper.

II. Background

Pre-Copy migration [4, 22] is a widely used live VM migration approach. As shown in Figure 1(a), when live migration starts, the source machine executes a Pre-Copy phase in which it iteratively transfers the memory pages of a VM to the destination machine. The first iteration transfers all pages and subsequent iterations transfer only the pages dirtied in the previous iteration. When the estimated downtime is less than a threshold, the source machine pauses the VM and transmits the remaining dirty pages, the hardware device state, and the CPU state to the destination. The VM is then resumed on the destination machine. This approach is called live migration because, during the Pre-Copy phase, the guest OS and all applications inside the migrated VM continue execution on the source machine. The time between suspending the VM on the source machine and resuming the VM on the destination machine is called downtime. The eviction time is from the time when migration starts till the time when the VM's state has been eliminated from the source.

Besides downtime, the other primary overhead in the Pre-Copy algorithm arises from the need to track the pages dirtied (written to) by the VM during the Pre-Copy rounds. In each iteration, *dirty page tracking* requires the hypervisor to mark all guest pages as read-only, trap memory writes by the guest, record the dirtied page location, and revert the page back to writable permissions.

III. Design

Quick Eviction speeds up the transfer of a VM's memory state from the source by evicting the bulk of the VM's memory footprint in the background during VM's normal operation. This enables the source machine to rapidly offload a VM's state once the migration command is issued, such as when a failure of the source is imminent or to quickly recover the performance at the source.

Quick Eviction works by regularly transferring incremental snapshots of a VM's memory to a remote destination. The remote destination could be either the final machine where the VM will execute, if known in advance, or an intermediate *staging* machine for the snapshot, if the final destination is not pre-determined. In both cases, the eviction time is measured from the instant the migration command is issued to the instant that the entire state of the VM is transferred out of the source machine.

Individual snapshots in Quick Eviction do not need to be either complete or fully consistent; instead snapshots contribute to progress towards the eventual eviction of a complete VM state. When the migration command is issued, any remaining dirty pages are transferred to the destination. If the snapshots are performed regularly, the number of dirty pages remaining at the source at migration time would be significantly smaller than the number of total memory pages in the VM, resulting in a short eviction time.

Figure 1(b) demonstrates various phases of Quick Eviction algorithm. Prior to migration, the VM runs normally on the source machine. During this normal execution,

the source machine regularly transfers live incremental snapshots of the VM's memory to the destination. In the very first live snapshot, all memory pages of the VM are transferred to the destination. In subsequent live snapshots, only the pages dirtied since the previous snapshot are transferred. Once the live migration is initiated, any pages dirtied since the last snapshot are iteratively transferred to the destination, as in pre-copy migration. During downtime, any remaining dirty pages, VCPU state, and I/O state are transferred to the destination to complete the eviction from source. The VM resumes at the destination if it happens to be the final destination, else the VM is sent to the final destination from the staging node.

We use two techniques to reduce any impact of live snapshots on the performance of application running in VMs: *dynamic snapshot interval* and *transfer limit*.

Dynamic Snapshot Interval: During normal operations, this mechanism dynamically spaces the live snapshots according to nature of the VM's workloads. The VM's memory state is periodically checked to determine the number of pages dirtied since the last snapshot. The next live snapshot operation is initiated once the dirty page count exceeds a threshold and a minimum time has passed since the last snapshot.

Transfer Limit: This mechanism reduces the network overhead during each live snapshot by bounding the number of pages that can be transferred during each snapshot operation. In other words, each live snapshot transfers up to a maximum number of dirty pages; any dirty pages in excess are deferred to the next snapshot.

There are several advantages of the above two mechanisms from the viewpoint of overhead reduction. First, applications that do not dirty memory very frequently pay very little penalty from dirty page tracking and memory transfers. This is because snapshots are not automatically performed with a fixed frequency but deferred until sufficient number of dirty pages have accumulated.

Secondly, by limiting the number of pages transferred in each snapshot, we limit any potential interference of snapshot transfer with any network-bound applications.

Thirdly, when the migration is initiated, the number of dirty pages is guaranteed to be lower than or equal to the threshold used for triggering the snapshot operation. This property significantly reduces the total migration time compared to pre-copy algorithm.

Finally, if a VM dirties too many pages before a snapshot, then the cost of transmitting those dirtied pages is amortized over several subsequent snapshots rather than all at once in the immediate snapshot.

Our experimental results in Section V validate that dynamic snapshot intervals coupled with transfer limits reduces the eviction time, while making the impact of snapshots on network throughput negligible. It should be noted that there is no observable overhead in checking the number of pages dirtied since the last snapshot, as it just involves examining a *dirty bitmap* maintained by the hypervisor by tracking write faults on VM's memory.

IV. Implementation

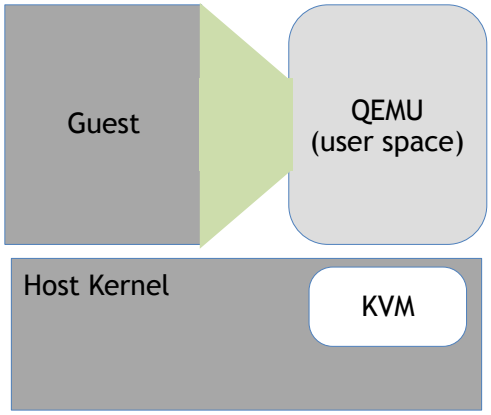

Fig. 2. The positions of the KVM hypervisor and QEMU management process in the KVM/QEMU virtualization platform.

Our Quick Eviction migration technique is implemented by modifying an implementation of Pre-Copy migration in the KVM/QEMU [18] virtualization platform version 2.5.0. The guest OS and applications inside the guest are unmodified. As shown in Figure 2, each VM in KVM/QEMU is associated with a userspace management process, called the QEMU, which performs I/O device emulation and various management functions, including migration and checkpointing. The userspace QEMU process communicates with a kernel-space hypervisor, called KVM, that uses hardware virtualization features to execute the VM in guest mode (or non-root mode). The Pre-Copy migration algorithm is implemented in KVM/QEMU mainly as part of the QEMU process, with some assistance from the KVM hypervisor to perform dirty page tracking.

A key hypervisor feature on which both Pre-Copy and Quick Eviction rely is the *dirty page tracking* mechanism. KVM keeps uses a dirty bitmap to track of pages dirtied by a VM since a given time instant. When the migration process starts, all memory pages are marked as read-only. When a process attempts to write to any of these pages, a write fault is triggered at which point KVM marks the page as read-write and the corresponding bit in the dirty page bitmap is turned on. Successive writes to the same page do not cause any traps or bitmap updates until the bitmap is reset the next time. The QEMU process reads the bitmap and transfers the pages marked as dirty in the bitmap. Once QEMU traverses the entire bitmap, it again reads the bitmap in the KVM. KVM's bitmap is reset whenever QEMU reads the bitmap from it. When QEMU reads and transfers a dirty page to the destination machine, the corresponding dirty bit is cleared.

Algorithm 1 is the pseudo-code for Quick Eviction. To begin with, Quick Eviction sends all memory pages of

Algorithm 1 Quick Eviction Algorithm

```
1: ▷ Initial preparation
2:     Perform initial snapshot of full memory;
3:
4: ▷ Incremental snapshots
5: while (migrate command not issued)
6:     sleep(checking_interval);
7:     if (Number of dirty pages ≥ dirty_page_threshold)
8:         if (Number of dirty pages ≤ max_page)
9:             send all dirty pages to the destination;
10:        else send max_page dirty pages to the destination;
11:        endif
12:    endif
13: endwhile
14:
15: ▷ Live Migration
16: while (Number of dirty pages ≥ downtime_page_limit)
17:    send all dirty pages to the destination;
18: endwhile
19: Suspend the VM and send all remaining pages, VCPU state, and I/O state to the destination;
```

the VM once to the destination, followed by four more iterations to send dirty pages generated during the previous round. This is done only once at the start of Quick Eviction to bring down the dirty page rate to the writable working set of the current load inside the VM. Once done, a migration thread in QEMU periodically counts the number of dirty pages in the memory using a *ioctl* call which reads the dirty page bitmap from the KVM hypervisor. The migration thread then sleeps for a specific interval before checking whether the dirty page count exceeds the dirty page threshold (Line 6). If the dirty page count exceeds the dirty page threshold (Line 7), a snapshot operation is performed which sends at most `max_page` dirty pages to the destination (Lines 7–12). Migration thread starts sending dirty pages as per the bitmap and resets the corresponding bits till the snapshot transfer limit is reached. Any remaining dirty pages are sent in the next snapshot operation.

When the migration is initiated, the migration thread in QEMU retrieves the bitmap from KVM and sends dirty pages to the destination until the number of dirty pages reaches a threshold (`downtime_page_limit` in Line 16). At this point, the VM is suspended and the remaining pages, VCPU, and I/O states are transferred. Finally, the destination migration thread resumes the VM at the destination host. The administrator can customize all parameters in Algorithm 1 including `dirty_page_threshold`, `checking_interval`, and `downtime_page_limit`, to customize the behavior of Quick Eviction.

V. Evaluation

This section compares the performance of our Quick Eviction technique against the KVM pre-copy migration [4]. The performance is measured using the following metrics:

- *Eviction time:* Time taken to transfer a VM's state out of the source.
- *Downtime:* Duration that the VM is suspended during the migration.
- *Performance degradation:* Any adverse performance impact on applications running inside the VM during migration.
- *Network bandwidth degradation:* Reduction in network bandwidth during VM migration.

Our test environment consists of dual six-core 2.1GHz Intel Xeon machines with 128 GB memory connected through a Gigabit Ethernet switch with 1Gbps full-duplex ports. VMs in each experiment are configured with 1 vCPU and 8GB of memory with page size of 4KB unless specified otherwise. Virtual disks are accessed over the network from an NFS server, which enables each VM to access its storage from both source and destination machines. Each data point reported in this section (except Figures 7 and 8) is an average of 5 runs of experiments.

A. Baseline Comparison of Quick Eviction and Pre-copy

Figure 3 compares the eviction time of an idle VM using Quick Eviction and pre-copy. The size of the VM ranges from 1GB to 8GB. The migration phase of Quick Eviction starts after five incremental snapshots. Figure 3 shows that the eviction time of Quick Eviction is very small (between $6ms$ and $246ms$) compared to pre-copy, although it increases linearly for both as the size of the idle VM increases. When using Quick Eviction, almost all dirty pages of the idle VM have been transferred to the destination machine before the migration phase starts.

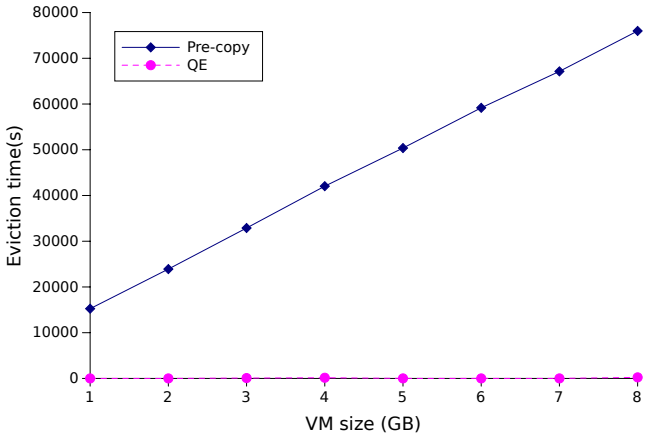

Fig. 3. Eviction time variation for different VM sizes (idle VM)

The downtime (not shown) of Quick Eviction and pre-copy are the same and constant around 5ms. Only a small number of pages and CPU and device I/O states were transferred during downtime.

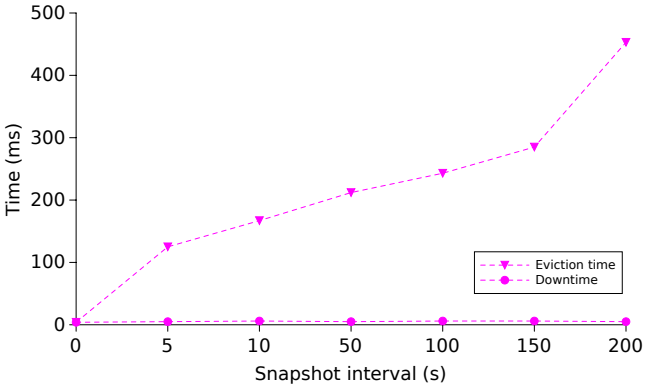

Fig. 4. Effect of varying checking interval on the eviction time using Quick Eviction.

Figure 4 shows the eviction time and the downtime of migrating a 8GB idle VM with different checking intervals, i.e. the time between successive checks of the dirty bitmap. Although the VM is idle, the memory pages of the VM were still very slowly dirtied during Quick Eviction. As a result, the eviction time increases up to 450ms as the checking interval increases to 200s. The downtime remains low and constant at around 5ms.

B. Effect of Memory Write-Intensive Workload

Figures 5 and 6 show the eviction time and the downtime when migrating a VM running write-intensive workloads using Quick Eviction and pre-copy, respectively. The memory-write intensive application we used is a C program that writes random numbers to a large region of main memory. The program starts before the VM was migrated and continues writing to a large region of memory during migration. The size of the working set (i.e., the size of the memory written) ranges from 100MB to 900MB.

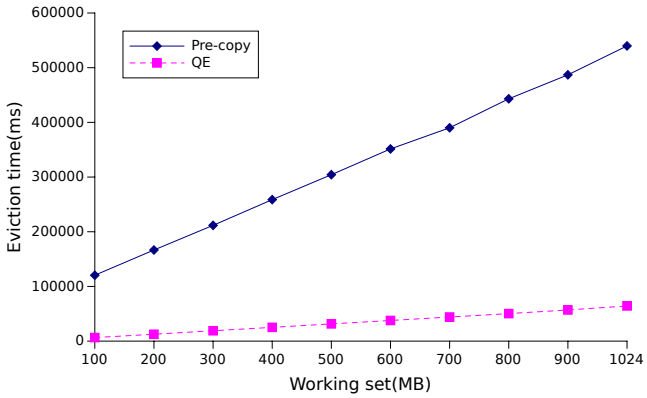

Fig. 5. Eviction time of a VM running a write-intensive application using Quick Eviction and pre-copy.

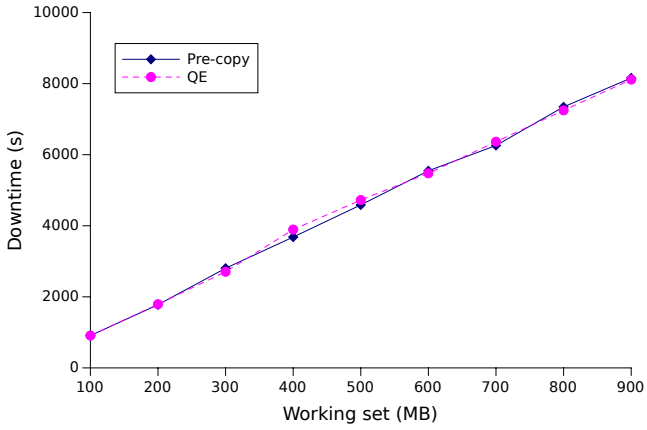

Fig. 6. Downtime of migrating a VM running a write-intensive application using Quick Eviction and pre-copy.

Observe from the figures that the eviction time for pre-copy increases at a much higher rate (by a factor of 8 to 18) than that of Quick Eviction. This is because, with pre-copy, the source machine needs to transfer all memory pages of the VM to the destination, while with Quick Eviction, the number of pages dirtied when the migration phase starts is the same as the size of the working set. The downtime, on the other hand, is almost the same for both Quick Eviction and pre-copy, as both Quick Eviction and pre-copy try to minimize the downtime in the same way.

In addition, for write-intensive applications, we also compared the eviction time and downtime of Quick Eviction while varying the checking interval. We found that the checking interval does not affect the eviction time and downtime because the write-intensive application dirties the memory very quickly and all memory pages in the working set are dirtied in each iteration.

C. Application Performance Overhead During Migration

We measured how Quick Eviction and pre-copy affect the performance of CPU-intensive workloads using two benchmarks – Quick Sort and Kernbench[20]. The Quick Sort program first allocates 1GB of memory using malloc.

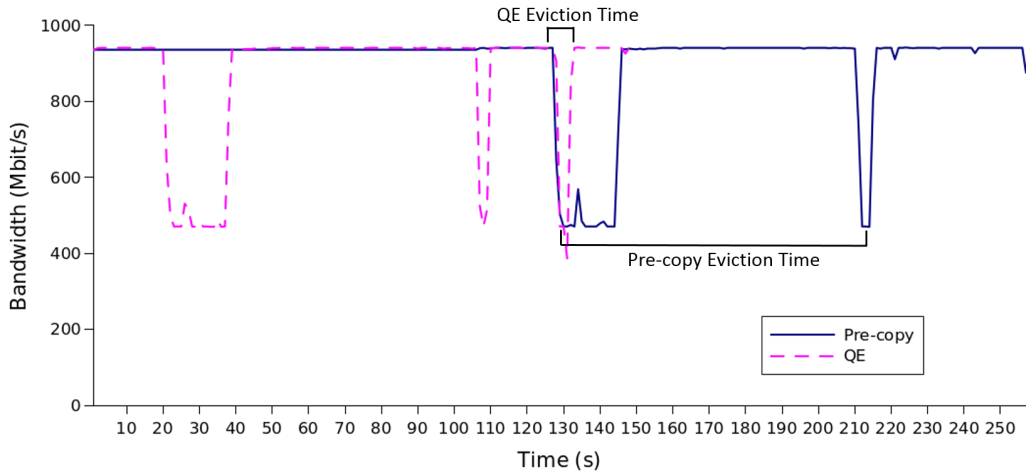

Fig. 7. Network bandwidth when migrating an idle VM using Quick Eviction and pre-copy.

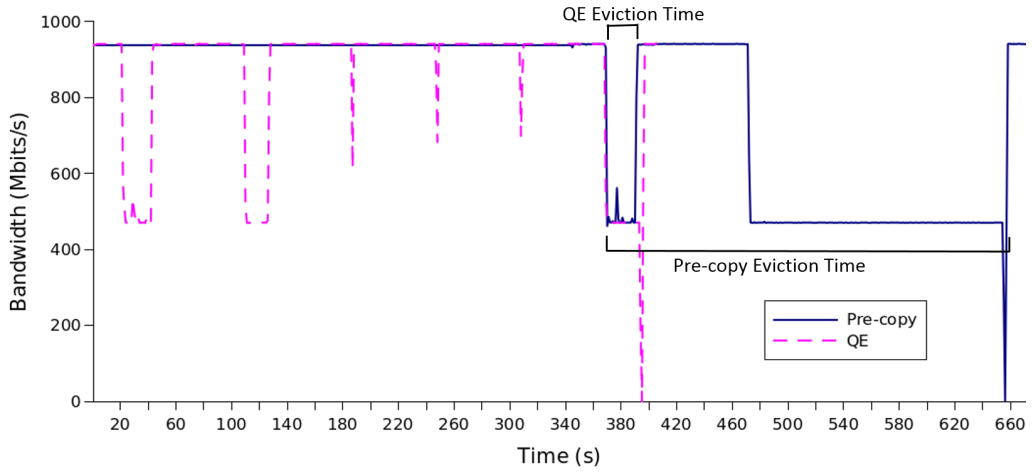

Fig. 8. Network bandwidth when migrating a VM running memory-write-intensive application using Quick Eviction and pre-copy.

The program then takes out a 200KB memory segment at a time, writes random integers to the segment, and sorts the integers using the Quick Sort algorithm. Once the integers are sorted, the program moves to the next 200KB segment. The above process repeats until it reaches the last segment of the 1GB allocated memory, when the program loops back to the first segment. We measured the number of sorting operations performed (i.e., the number of times when random integers are written to the 200KB memory and are sorted) per second, when migrating the VM using pre-copy and Quick Eviction. The Quick Sort program starts before the migration starts and continues writing and sorting integers during the migration. Our experimental results show that the number of sorts performed per second is constant during migration, except that the Quick Sort program stops during downtime. Thus both Quick Eviction and pre-copy do not degrade the performance of Quick Sort except at downtime.

We also compared the time of compiling Linux kernel version 4.2.0 using Kernbench with and without migration. Our experimental results show that pre-copy and Quick Eviction do not increase the kernel compilation time during migration except at downtime, which reinforces our statement that Quick Eviction has no observable impact on CPU performance of applications running inside the VM.

D. Network Overhead During migration

VM Migration itself is a network intensive process. Thus, there is a contention of network resources between the migration process and the applications running inside the VM. Migrating a VM using Quick Eviction or pre-copy may reduce the outgoing network bandwidth of the source machine and the incoming network bandwidth of the destination machine, but not the network bandwidth in another direction.

We used iPerf, a network intensive application, to measure the outgoing bandwidth from the source while migrating the VM using Quick Eviction and pre-copy. The iPerf server runs on an external machine (i.e. neither source nor destination machine) in the same cluster and the iPerf client

resides inside the VM being migrated. During migration, the client continuously sends data to the server through a TCP connection. The VM is configured with 8GB memory and a single vCPU. The network bandwidth is captured using iPerf every second.

Figures 7 and 8 compare the network bandwidth before, during, and after the migration when migrating an idle VM and a VM running a write-intensive application (with $200MB$ working set) using Quick Eviction and pre-copy. Different plots within each graph are aligned on the timing of invocation of the migration command. The checking interval was set to 60 seconds for better visualization of network bandwidth.

These two figures show that the network bandwidth dropped (from 940 Mbps to around 470 Mbps) during the first iteration for both Quick Eviction and pre-copy, when large memory pages were sent to the destination. When the VM is idle, both pre-copy and Quick Eviction do not impose high overhead on network after the first iteration. When migrating a VM running the memory-write-intensive application using pre-copy, the network bandwidth drops by half for around 170 seconds after the first iteration. Quick Eviction does not impose high overhead on network after the first iteration.

Note that, in both figures, when migration starts, after an initial drop of the bandwidth to 470 Mbps, the bandwidth increases back to 940 Mbps for a period of time, and then decreases to 470 Mbps. This is due to an optimization in the KVM pre-copy implementation that avoids transferring a page if all bytes in the page are the same. If all bytes in a page are the same, pre-copy transfers only 8 bytes, instead of the entire page to the destination. In the above two experiments, most memory pages are zero pages or are identical when the migration starts. This is because idle VM dirties very few memory pages and the memory-write-intensive application dirties only 1GB memory segment out of the 8GB memory of the VM. As a result, in the first iteration, pre-copy sends only 8 bytes to the destination for most pages, which results in a drop on bandwidth (sending dirty pages) and then an increase in the bandwidth (sending 8 bytes for each empty/zero pages). In subsequent iterations, pre-copy tracks dirty pages through page faults and very few zero/identical pages are marked dirty, and hence there is no increase in the bandwidth in subsequent iterations.

E. Effect of Dirty-page Threshold

As described in Algorithm 1, the outstanding dirty page count triggers a snapshot operation whenever the count exceeds a predetermined threshold referred to as dirty page threshold. This section evaluates how dirty page threshold affects the performance of Quick Eviction and the number of snapshot taken using Kernbench[20]. During the VM migration, Kernbench compiles Linux kernel version 4.2.0 inside the VM migrated.

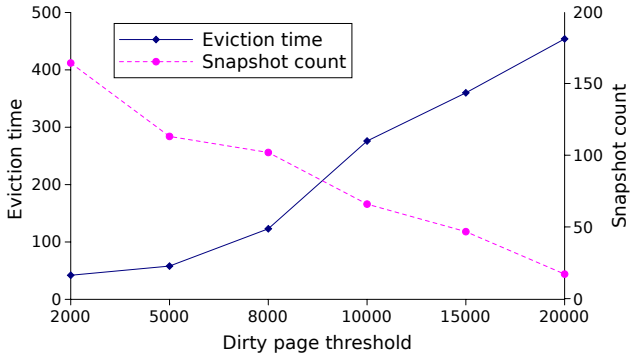

Fig. 9. Eviction time and the number of snapshots performed vs dirty page threshold when migrating a VM running Kernbench using Quick Eviction.

Figure 9 shows how the dirty page threshold affects the number of snapshot operations performed and the eviction time when migrating a VM running Kernbench using Quick Eviction over a duration of 200 second incremental snapshot phase. The results of this figure are averaged over several runs. Observe from the figure that, when the dirty page threshold increases from 2000 to 20000, the number of snapshot operations decreases from 158 to 22 and the eviction time increases from $42ms$ to $454ms$. The dirty page threshold does not affect the downtime as Quick Eviction performs back-to-back dirty page transfer rounds so that downtime phase has minimum possible dirty pages.

It should be noted that the eviction time of Quick Eviction depends on outstanding number of dirty pages at the instant that migration is initiated. The dirty page count is always lower than the snapshot threshold. However, it could be large (closer to the threshold) or small (closer to 0) depending on whether the time between the last snapshot and the migration trigger is large or small, respectively.

VI. Related Work

A number of optimizations have been proposed to speed up VM migration. Content optimizations such as deduplication [6, 7, 17, 23, 31] and compression [7, 15, 24] reduce the amount of data transferred during migration and hence reduce the migration time. Post-copy migration [13, 25] is an alternative live VM migration approach in which each memory page in the VM is transferred only once. Post-copy reduces the migration time for memory-write-intensive applications and usually has smaller downtime than pre-copy. Reactive cloud [14] uses post-copy to react to sudden overloads quickly. Scatter-Gather VM migration [8] allows fast eviction of a VM when the destination machine is resource constrained. VMWare uses a per-VM swap device [1] shared between the source and the destination machines to avoid the transfer of swapped out VM pages during the migration. Jo et.al. [16] avoid transferring cached pages from the source; such pages are fetched directly from the network-attached storage. Jettison [2] proposes partial VM migration in which only the working set of

an idle VM is transferred to the destination, while the remaining memory pages are slowly demand-paged from the source. Agile [10] migration reduces the migration time by swapping the non-working set pages of a VM to a per-VM swap device and migrating only the working set pages using a hybrid pre/post-copy technique. Vecycle [19] stores a checkpoint at each machine which a VM has visited in the past. When migrating a VM back to a previously visited node, the earlier checkpoint on that node can be reused to initiate the VM, which potentially reduces the migration traffic and time. However, since regular snapshots are not performed, over time a VM's memory footprint may diverge significantly from its snapshot on other nodes effectively increasing the migration time. All approaches mentioned reduce the migration time by reducing the amount of data transferred during migration, none of them use regular incremental snapshots in advance of migration to reduce the eviction time from source.

In the realm of fault tolerance for Virtual Machines, Remus [5] proposed a system for ensuring high-availability of VMs against unexpected failures by periodically checkpointing the VM's state, including CPU, memory, and I/O device states to a backup machine. Similar approach was applied by Nagarajan et al.[21] for high performance computing applications such as MPI. VMWare provides a record and replay [28] mechanism by which events external to a VM can be recorded and replayed on a backup VM image in sync. The above approaches activate the backup VM after detecting a failure, which results in strict demands on consistency between primary and backup VMs, i.e., a backup VM must be able to take over operations after a primary VM's failure at *any arbitrary point in execution*. This requirement increases both the complexity and frequency of checkpointing operations, either through VM state transfer or event logging and replay. In contrast, Quick Eviction is designed for a less restrictive operational setting where the VM's migration is triggered by the administrator in anticipation of failure, versus as a reaction to failure. This enables Quick Eviction to transfer very lightweight snapshots that are neither complete nor consistent, allowing the full primary-backup consistency to be deferred until the actual migration time. While Quick Eviction is not a full-fledged solution for a high-availability system, it is suitable as a lightweight solution where VMs must be evicted quickly when failure is imminent.

A number of techniques have been proposed to detect resource pressures and alleviate them through VM migration. VMWare DRS [26] and SandPiper [32] monitor resource usage at the host-level to detect hotspots which triggers VM migration. Zhang et al. [33] use access-bit scanning to estimate the working set of a VM. Chiang et al. [3] propose to resize VMs based on the size of their working set to increase consolidation. Overdriver [30] proposes to resolve transient hotspot with network memory swap and sustained hotspots via migration. An optimization of pre-copy, called SDPS [27], reduces the migration time of write-intensive VMs by slowing down vCPUs, which degrades the application performance during migration [29]. Quick Eviction is orthogonal and could be used in conjunction with the above approaches.

VII. Conclusion

Rapid live migration of VMs is critical in responding to events in a data center, such as an imminent failure or an emerging performance hotspot. We presented Quick Eviction, a new approach to live migration of a VM that significantly reduces the eviction time of a VM from the source. Existing migration techniques such as pre-copy transfer the entire memory of a VM after the triggering event. In contrast, Quick Eviction staggers the memory transfer throughout the lifetime of the VM in a controlled manner, yielding low eviction times, low impact on application performance, and low network overheads. The future work includes addressing simultaneous Quick Eviction of multiple VMs. There is also room for adaptive algorithms that dynamically make snapshot decisions based on workload running inside the VM to balance the tradeoff between network overheads and eviction time.

Acknowledgment

This work is supported in part by the Air Force Rome Research Laboratory and the National Science Foundation through grants 1320689 and 1527338.

References

[1] I. Banerjee, P. Moltmann, K. Tati, and R. Venkatasubramanian. VMware ESX Memory Resource Management: Swap. In *VMWare Technical Journal*, 2014.

[2] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient Idle Desktop Consolidation with Partial VM Migration. In *Eurosys*, April 2012.

[3] J. Chiang, H. Li, and T. Chiueh. Working Set-based Physical Memory Ballooning. In *ICAC*, June 2013.

[4] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI*, 2005.

[5] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI*, 2008.

[6] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan. Gang migration of virtual machines using cluster-wide deduplication. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2013.

[7] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *ACM Symposium on*

High-Performance Parallel and Distributed Computing (HPDC), 2011.

[8] U. Deshpande, Y. You, D. Chan, N. Bila, and K. Gopalan. Fast server deprovisioning through scatter-gather live migration of virtual machine. In *IEEE Cloud*, July 2014.

[9] Umesh Deshpande, Danny Chan, Steven Chan, Kartik Gopalan, and Nilton Bila. Scatter-gather live migration of virtual machines. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.

[10] Umesh Deshpande, Danny Chan, Ten-Young Guh, James Edouard, Kartik Gopalan, and Nilton Bila. Agile live migration of virtual machines. In *IEEE International Parallel and Distributed Processing Symposium, Chicago, IL, USA*, May 2016.

[11] Umesh Deshpande, Yang You, Danny Chan, Nilton Bila, and Kartik Gopalan. Fast server deprovisioning through scatter-gather live migration of virtual machines. In *Proceedings of the 7th IEEE International Conference on Cloud Computing (IEEE Cloud)*, June 2014.

[12] Michael Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), Washington, DC*, March 2009.

[13] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Operating System Review*, 43(3):14–26, 2009.

[14] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. Reactive cloud: Consolidating virtual machines with postcopy live migration. *IPSJ Transactions on Advanced Computing Systems*, pages 86–98, March 2012.

[15] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive, memory compression. In *Proc. of Cluster Computing and Workshops*, August 2009.

[16] C. Jo, E. Gustafsson, J. Son, and B. Egger. Efficient live migration of virtual machines using shared storage. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2013.

[17] S. A. Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, June 2011.

[18] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the linux virtual machine monitor. In *Proc. of Linux Symposium*, June 2007.

[19] Thomas Knauth and Christof Fetzer. Vecycle: Recycling vm checkpoints for faster migrations. In *Proceedings of the 16th Annual Middleware Conference*, pages 210–221, 2015.

[20] C. Kolivas. Kernbench. http://ck.kolivas.org/apps/kernbench/kernbench-0.50/.

[21] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, pages 23–32, 2007.

[22] M. Nelson, B. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 25–25, 2005.

[23] P. Riteau, C. Morin, and T. Priol. Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing. In *Proc. of EURO-PAR*, September 2011.

[24] P. Svard, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *VEE*, 2011.

[25] Takahiro Hirofuchi and Isaku Yamahata. Postcopy live migration for qemu/kvm. http://grivon.apgrid.org/quick-kvm-migration.

[26] VMWare Inc. VMware DRS: Dynamic Scheduling of System Resources, http://www.vmware.com/files/pdf/drs_datasheet.pdf.

[27] VMWare Inc. *VMware vSphere vMotion Architecture, Performance and Best Practices in VMware vSphere 5, https://www.vmware.com/files/pdf/vmotion-perf-vsphere5.pdf*.

[28] VMWare Inc. Protecting Mission-Critical Workloads with VMware Fault Tolerance. White Paper, Revision: 20090507 Item: WP-084-PRD-01-02, 2009.

[29] VMWare Knowledge Base. *Virtual machine performance degrades while a vMotion is being performed, http://kb.vmware.com/kb/2007595*.

[30] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon. Overdriver: handling memory overload in an oversubscribed cloud. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2011.

[31] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2011.

[32] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Intl. Journal of Computer and Telecommunications Networking*, 53(17), 2009.

[33] I. Zhang, A. Garthwaite, Y. Baskakov, and K. C. Barr. Fast restore of checkpointed memory using working set estimation. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2011.