HotRestore: A Fast Restore System for Virtual Machine Cluster

Lei Cui, Jianxin Li, Tianyu Wo, Bo Li, Renyu Yang, Yingjie Cao, Jinpeng Huai
State Key Laboratory of Software Development Environment
Beihang University, China
$\{$*cuilei, lijx, woty, libo, yangry, caoyj*$\}$*@act.buaa.edu.cn* $\{$*huaijp*$\}$*@buaa.edu.cn*

Abstract

A common way for virtual machine cluster (VMC) to tolerate failures is to create distributed snapshot and then restore from the snapshot upon failure. However, restoring the whole VMC suffers from long restore latency due to large size snapshot files. Besides, different latencies would make the virtual machines not start at the same time. The prior started virtual machine (VM) thus cannot communicate with the VM that is still restoring, leading to the TCP backoff problem.

In this paper, we present a novel restore approach called HotRestore, which restores the VMC rapidly without compromising performance. Firstly, HotRestore restores the single VM through an elastic working set which prefetches the working set in a scalable window size, thereby reducing the restore latency. Second, HotRestore constructs the communication-induced restore dependency graph, and schedules the restore line to mitigate the TCP backoff problem. Lastly, a restore protocol is proposed to minimize the backoff duration. Additionally, a prototype has been implemented on QEMU/KVM. The experimental results demonstrate that HotRestore can restore the VMC within a few seconds whilst reducing the TCP backoff duration to merely dozens of milliseconds.

1 Introduction

Machine virtualization is now widely used in datacenters and this has led to lots of changes to distributed applications within virtualized environments. In particular, the distributed applications are now encapsulated into virtual machine cluster (VMC) which provides an isolated and scaled computing paradigm [1, 24, 22]. However, failures increasingly become the norm rather than the exception in large scale data centers [17, 35]. The variety of unpredictable failures might cause VM crash or network interruption, and further lead to the unavailability of applications running inside the VMC. There are many approaches for reliability enhancement in virtualized environment. Snapshot/restore [25, 39, 40, 37] is the most widely used one among them. It saves the running state of the applications periodically during the failure-free execution. Upon a failure, the system can restore the computation from a recorded intermediate state rather than the initial state, thereby significantly reducing the amount of lost computation. This feature enables the system administrators to recover the system and immediately regain the full capacity in the face of failures.

In the past decades, several methods have been proposed to create distributed snapshot of VMC, and most of them aim to guarantee the global consistency whilst reducing the overhead such as downtime, snapshot size, duration, etc [11, 25, 14]. However, restoring the VMC has received less attention probably because restoration is only conducted upon failures. As a matter of fact, due to the frequently occurring failures in large scale data centers, restore becomes frequent events accordingly [35]. Worse still, just one VM crash would lead to the entire VMC's restoration with the consistency guarantee taken into account. The frequent restore of multiple virtual machines cause non-negligible overheads such as restore latency and performance penalty. Here, restore latency is referred to the time to load saved states from the persistent storage until the VM execution is resumed.

There are a few well-studied works on improving VM restore. *Working set restore* [39, 40] is one solution proposed recently, and it restores single VM by prefetching the working set. Since the working set reflects the active access locality, this method reduces the restore latency without performance compromising. It seems that the VMC restore could be simply accomplished by restoring the VMs individually with *working set restore* [19]. Unfortunately, several practical drawbacks are still far from settled.

First, the restore latency with *working set restore* is still long. In fact, the latency is proportional to the work-

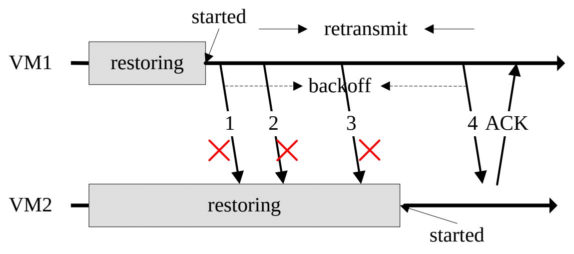


Figure 1: A TCP transmission case during restore. VM1 after start sends packet1 to VM2, this packet will be lost because VM2 which is restoring is currently suspended. VM1 will resend this packet once retransmission timeout (RTO) is reached. The packets 2, 3, 4 are retransmitted packets of packet 1. If packet1 is a SYN to connect VM2, the TCP handshake would fail if the retransmission duration exceeds TCP connection timeout. If the packet1 is a heartbeat packet, the timeout may cause the VM2 be falsely reported as fail which would result in misbehaviour.

ing set size which is directly related to the usage pattern of the workload. For memory intensive tasks, the working set might be the entire memory, making the *working set restore* degrade to *eager restore*[37] which starts the VM after all states are loaded.

Second, due to the various memory sizes and sorts of workloads, VMs cooperated in the VMC may have different working set sizes. This difference results in diverse restore latencies and hence causes VMs being launched at different times. As a result, if the prior started VM sends a packet to a restoring VM, it will not receive any reply within a while. Meanwhile, this might lead to temporary backoff of active TCP connections as illustrated in Figure 1.

Therefore, in the cluster, the practical VM disruption time not only includes the restore latency but is determined by the TCP backoff duration as well. The backoff duration directly depends on the degree of discrepancy among VMs' restore completion times, i.e., start times. What's more, due to the complexity of workloads, these VMs may differ greatly in working set sizes, making TCP backoff duration the dominant factor in disruption.

In this paper, we propose HotRestore, which is capable of restoring the saved states of VMC swiftly and efficiently. Unlike prior VMC restore tools, HotRestore could resume the applications' execution within a few seconds and the applications can regain their full capacity rapidly. Moreover, the distributed applications only suffer transient network interruption during the restoration. Consequently, HotRestore can be naturally adopted into high availability scenarios, where fast recovery is essentially critical to provide reliable services to end users.

To address these challenges, HotRestore proposes two key ideas. On the one hand, it traces the memory access during *post-snapshot*, and records the traced pages in a first-access-first-load (FAFL) queue, which finally constitutes an elastic working set. The motivation behind is that when a VM is restored, it will be roll-backed to the snapshot point. If the execution is deterministic, the VM will re-execute in the same way as that of *post-snapshot*. As a result, the traced memory pages during *post-snapshot* will be touched again and thus can be regarded as working set pages. By only loading the working set pages upon restore, HotRestore reduces the restore latency a lot.

On the other hand, restore line is designed and it depicts the start times of VMs in order to reduce the TCP backoff duration. The basic idea is that for a packet sent from one VM, the associated destination VM must have been started to receive the packet, avoiding the potential backoff. The restore line derives from defacto restore order and causal restore order. The former one is revealed by the calculated working set sizes of VMs while the latter is communication-induced. Since the semantics of the two orders might not match, HotRestore revises the working set sizes to make defacto order be consistent with causal order, and thereafter computes the restore line. Moreover, a restore protocol is designed to guarantee that the VMs can start as indicated by the restore line, thereby significantly minimizing the backoff duration.

Our contribution is three-fold. First, we introduce elastic working set, which is a subset of memory pages in any desired size, in order to restore the single VM rapidly. Second, we propose restore line for virtual machines that cooperate into a cluster, schedule the VMs' start times to minimize the TCP backoff duration. Third, we have implemented HotRestore on our work previous called HotSnap [14] which creates distributed snapshots of the VMC, and conducted several experiments to justify its effectiveness.

The rest of the paper is organized as follows. The next section gives a brief overview of previous works on HotSnap while section 3 presents the concepts and implementation of elastic working set. Section 4 introduces the algorithm to compute the restore line and the restore protocol is also proposed. Section 5 describes several implementation details on QEMU/KVM platform followed by the experimental results in Section 6. Finally we present the previous work related to HotRestore in section 7 and conclude our work in Section 8.

2 A Brief Overview of HotSnap

HotSnap creates a global consistent state among the VMs' snapshots. It proposes a two stage VM snapshot creation approach consisting of *transient snapshot* and *full* *snapshot*. In *transient snapshot*, HotSnap suspends the VM, records the CPU and device state, sets guest memory pages to be write-protected, and then starts the VM. After that, *full snapshot* starts up. HotSnap will save the guest pages in a copy-on-write manner. Specifically, upon a page fault triggered by touching the write-protected page, HotSnap will save the page into snapshot file, remove the write-protect flag and then resume the VM. The recorded snapshot is actually the instantaneous state in *transient snapshot*; therefore *full snapshot* is actually a part of *post-snapshot* stage.

In HotSnap, we tailor the classical message coloring method to suit to virtualized platforms. In the coloring method, the packet color is divided into *pre-snapshot* and *post-snapshot*, so is the VM color. HotSnap intercepts the packet sent and received by the simulated tap device on QEMU/KVM. For a transmitted packet, HotSnap piggybacks the packet with the immediate VM color. For a received packet, if the packet color is *post-snapshot* while the VM color is *pre-snapshot*, the packet will be temporarily dropped to avoid inconsistency; otherwise, the packet will be forwarded to the virtual machine and be finally handled.

The consistency of global state is guaranteed during snapshot creation by HotSnap. Therefore, HotRestore makes no attempt to ensure consistency upon restore, which is different to previous works that focus on consistency as well as avoiding domino effect [16, 34, 36].

3 Elastic Working Set

This section first presents the estimation method of elastic working set, and then describes how to create snapshot and restore the VM when working set is employed.

3.1 Working Set Estimation

An elastic working set should satisfy three requirements. Firstly, the restore latency is as short as possible without comprising application performance. Secondly, high hit rate and high accuracy are achieved after VM starts. High hit rate implies most of the accessed pages hit in the working set, while high accuracy means most of the working set pages will be touched within a short while after restore. Thirdly, the working set size could scale up or scale down without the decrement of hit rate, accuracy and performance. In this manner, the size can be revised on-demand upon VMC restore. To estimate one such working set, we still need to determine: i) which pages should be filled into the working set, and ii) how many pages should be loaded on restoring (namely the working set size).

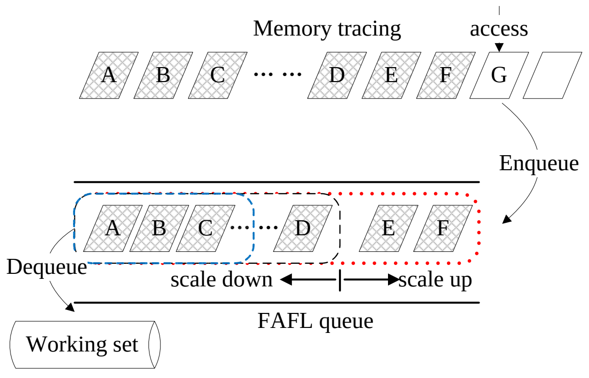


Figure 2: Elastic working set with the FAFL queue.

3.1.1 Working Set Pages

Upon the restoration, the VM will be roll-backed to the saved snapshot state and continue to execute. Ideally, if the execution is deterministic, the VM after restore will re-execute the same instructions and touch the same memory area as that of *post-snapshot*. This insight inspires us to trace the memory access during *post-snapshot* and regard the traced pages as candidate working set pages for producing the final working set.

The optimal selection from the candidate pages has been well studied. LRU and CLOCK are two classical and widely-used page replacement algorithms [23, 41]. However, we argue that the FIFO manner could be better in the mentioned scenario within the paper. The reason is as follows. If the execution is deterministic, the page access order after restore remains the same as that of *post-snapshot*. This implies that the first accessed page during *post-snapshot* will be firstly accessed after restoring. As a result, HotRestore adopts a first-access-first-load (FAFL) manner to determine the working set pages. Specifically, HotRestore saves the traced pages in the FAFL queue during *post-snapshot*, and loads the pages dequeued from the queue upon restore. Figure 2 illustrates the overview of FAFL queue. The traced pages, i.e., A-F, have been stored in the queue in access order, and the working set can be simply produced by dequeuing the pages from the queue.

Elasticity is another crucial feature of FAFL queue. The queue depicts the order of pages which are touched by OS kernel or applications during post-snapshot. On the one hand, given that the VM execution is deterministic, loading more or fewer pages into the working set upon restore will not influence the hit rate or accuracy seriously. Consequently, it enables the working set to scale up/down efficiently. On the other hand, the background load thread after VM is launched could still fetch the pages in the FAFL manner rather than loading the pages blindly. In this way, the probability of page fault could be effectively reduced. It is essentially important to maintain the application performance without severe degradation when the working set size decreases.

The external inputs events or time change make the VM execution be non-deterministic invariably in real world scenarios. Despite this, our experimental results[1] show that elastic working set could achieve high hit rate and accuracy. Meanwhile, it could scale up and scale down with the performance guaranteed.

3.1.2 Working Set Size

The working set size plays an important role when rolling back. Although a small size reduces the restore latency, it incurs numerous page faults, thus aggravating the decreased performance after restoring. On the other hand, a large size could avoid severe performance penalty produced by more loaded memory pages, but at the cost of long latency.

Statistical sampling approach described in ESX Server [38] is frequently used to predict the working set size. However, it lacks the ability to respond to the phase changes of workload in real time which is critical in restoration. Since the working set size upon restore should capture the size of workload activity in a timely manner, we propose a hybrid working set size estimation method. First, we adopt a statistical sampling approach to calculate the working set size during normal execution. This size depicts the average workload activity in a long period, and is referred to as WSS_{sample}. Secondly, we count the touched pages during *post-snapshot*. The page count reflects timely workload activity in part, and is referred to as $WSS_{snapshot}$. The expected working set size is the weighted sum of these two sizes and is determined by: $WSS = \alpha * WSS_{sample} + \beta * WSS_{snapshot}$

In most programs, the WSS remains nearly constant within a phase and then changes alternately. Since the workload keeps steady for quite a while [41], we empirically set α to be larger in HotRestore, i.e., α is 0.7 and β is 0.3. In addition, due to the sound scalability of FAFL queue, we adopt $WSS/2$ rather than WSS as the actual working set size upon restore. The remaining pages will be loaded by the background thread from the FAFL queue or by demand-paging. The experimental results in §6.1.3 demonstrate that the size shrinks and only incurs negligible performance overhead.

3.2 Snapshot and Restore

This section describes the VM snapshot and restore when elastic working set is employed.

Snapshot. Unlike HotSnap which only traces the memory write operations, HotRestore traces all accesses and records them in the FAFL queue to estimate the working set. Therefore, HotRestore adopts a snapshot approach that consists of copy-on-write and record-on-access. In detail, HotRestore sets the guest memory pages to be non-present to trace all accesses. For write operation, HotRestore records the page frame number into the FAFL queue, saves the page content into persistent storage, and then removes the non-present flag to allow the page to be read or written later on. For read operation, HotRestore only records the page frame number but does not save the page content. This is because read operations occur much more frequently than write operations, and saving the page content will lead to serious performance degradation. Therefore, HotRestore removes the read-protect flag but reserves the write-protect flag. The page content will be saved either in copy-on-write manner once it is written or through the background copy thread.

Restore. Upon restore, HotRestore firstly loads the CPU state and device state, and it fetches the working set pages into the guest memory. Afterwards, it sets the page table entries of unrestored pages to be non-present before starting the VM. The unrestored pages will be loaded by i) on-demand paging due to touching the non-present page after VM starts, or ii) background load thread running concurrently which ensures that restore finishes in a reasonable period of time.

4 Restore of Virtual Machine Cluster

The key of VMC restore is to mitigate the TCP backoff problem. In TCP, after sending a packet, the sender will wait for the *ack* from the receiver. It would resend the packet once timeout occurs to ensure that the packet is indeed received. Motivated by this, the basic idea to avoid TCP backoff is to ensure the receiver start before the sender.

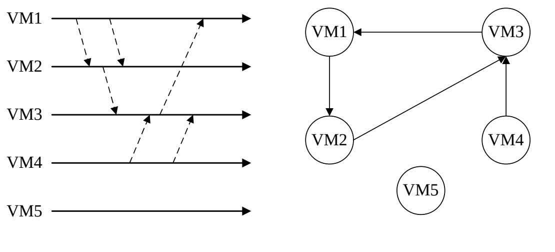
This section presents our solution to VMC restore. We start by describing the communication-induced restore dependency graph (RDG). Based on RDG, we compute the causal restore order[2] of VMs, and then schedule the restore line by revising the working set sizes of VMs. Finally, we introduce the restore protocol which ensures the VMs to start as indicated by the restore line.

4.1 Restore Dependency Graph

We define "VM_i depends on VM_j" or $VM_i \rightarrow VM_j$ if VM_i sends a packet to VM_j. If VM_i sends a packet to VM_j during snapshot, it will resend the packet after it is restored to the saved snapshot point. Therefore, the dependency

[1]§6.1 will explain the results.

[2]The restore order here describes the order of completion times of the working set restore, or namely VM start times, rather than restore start times.



(a) Time-space execution (b) Restore dependency graph

Figure 3: Communication-induced RDG.

can be reserved upon restore (we ensure the dependency by deterministic communication in §5.1). This motivates us to construct the RDG via the dependency among VMs. In RDG, each node represents one VM, and a directed edge is drawn from VM_i to VM_j if VM_i depends on VM_j. The name "restore dependency graph" comes from the observation that if there is an edge from VM_i to VM_j and VM_i is to be restored, then VM_j must be restored no later than VM_i to be ready to receive the packet and make the reply. Restore here means the "restore end" or "VM start", rather than "restore start". The dependency in RDG is transitive, i.e., if $VM_i{\rightarrow}VM_j$ and $VM_j{\rightarrow}VM_k$, then $VM_i{\rightarrow}VM_k$.

Figure 3(b) demonstrates a RDG yielded by the time-space diagram in Figure 3(a). RDG only depicts the dependency between different VMs, i.e., space, but has no concept of time whether it is physical time or global virtual time. This is because the packet order and send/receive time may change after restore due to non-deterministic execution. The lack of time semantics allows the existence of dependency ring, e.g., VM1, VM2 and VM3 form a ring. We define $VM_i \leftrightarrow VM_j$ if VM_i and VM_j are in a ring. VM5 is an orphan node; it neither depends on any VM, nor is depended by other VMs. The orphan node reflects the case that there is no packet sent or received associated with the node.

4.2 Restore Line Calculation

Restore line depicts the desired start times of VMs while guaranteeing the causal restore order of VMs. The causal restore order can be calculated from the RDG: if $VM_i \rightarrow VM_j$ in RDG, then VM_j should be ahead of VM_i in causal restore order. Moreover, the VM start time (or restore latency) is related to the working set size. The different working set sizes of VMs form the defacto restore order. This motivates us to compute the restore line by the causal restore order as well as the defacto restore order.

4.2.1 Causal Restore Order

The causal restore order is the semantics of logical start time; each dependency edge represents one elapsed clock. It is insufficient to obtain a unique causal restore order solely on RDG. First, the RDG is not necessarily a complete graph, e.g., there is no edge between VM2 and VM4, implying VM2 and VM4 can be restored independently after VM3. Second, the existence of the dependency ring and orphan node make a lot of choices on the order. As a result, rather than compute a unique order, we instead give the following rules to construct one feasible causal restore order through RDG.

Rule 1. The VM, which doesn't depend on other VMs but is depended, is prioritized in causal restore order.

Rule 2. VM_i should restore after VM_j, if VM_i depends on VM_j while VM_j does not depend on VM_i.

Rule 3. The VM that depends on several VMs will not restore until all the depended VMs are restored.

Rule 4. The VMs can restore independently if no dependency exists between them.

Rule 5. The VMs in the ring restore at the same time, i.e., they are placed together in causal restore order.

Rule 6. An orphan node may execute independently and therefore will be free in causal restore order.

Rule 1 seeks the VMs that should be restored first. Once found, the VMs that depend on prior restored VMs will join in the causal restore order by Rules 2 and 3. Rule 4 implies that the VMs can restore independently if there is neither direct nor transitive dependency between them. Based on these rules, we can reach two conclusions. First, after the completion of loading the working set, the VM can start if it satisfies any one of Rules 1, 4, 5, and 6. Second, after one VM is started, the VM can start accordingly if it depends on this started VM and satisfies the Rule 2 or Rule 3 as well.

4.2.2 Defacto Restore Order

The previously calculated working set sizes of VMs are always different due to the phase changes of workload, varieties of workloads or VM heterogeneity. These different sizes lead to different restore latencies and further form the defacto restore order which can be regarded as the semantics of physical start time.

The problem here is that the VM that is prior in causal restore order may have a larger working set size and thus start later than the latter VM, making the defacto restore order be inconsistent with the causal restore order. The inconsistency results in the TCP backoff problem as mentioned above. As a result, revising the working set sizes to match the two semantics becomes one crux of the problem in restoring the VMC. First, the defacto restore order after revision should be consistent with the causal restore order, to avoid TCP backoff. Second, the

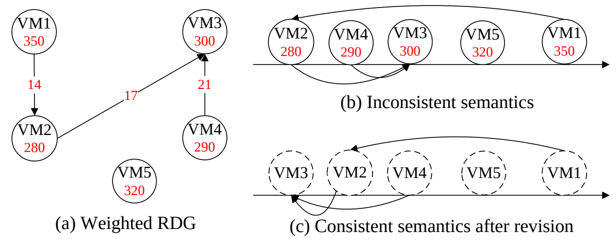


Figure 4: The revision of working set sizes. (a) is derived from Figure 3, but the dependency between VM1 and VM3 is removed.

revised working set size should not impose significant effects on latency or performance for single VM, i.e., minimum change on previous working set size.

4.2.3 Working Set Size Revision

The basic idea of revision is simple. Given that a case that $VM_i \rightarrow VM_j$, but the working set size of VM_j, i.e., S_j, is larger than S_i, we can decrease S_j or increase S_i, or do both to achieve the matching.

System Model. We consider network intensive applications which is common in nowadays data centers, such as distributed database, scientific computing, web services [1]. Their characteristic is that the communicating VMs send and receive packets frequently. We transfer the RDG to weighted RDG with the assignment to node value and edge weight, as shown in Figure 4(a).

In weighted RDG, the node value S_i is referred to as the previous calculated working set size of VM_i. The edge weight $W_{i,j}$ denotes the number of the captured packets sent from VM_i to VM_j during snapshot. Here, $W_{i,j}$ has no practical meaning, it is just used to ensure that the revised S_i, i.e., S_i^*, be larger than S_j^* if VM_i depends on VM_j. We denote this relation by $S_i^* \geqslant S_j^* + W_{i,j}$. Moreover, this inequation implies that if $VM_i \rightarrow VM_j$ and $VM_j \rightarrow VM_k$, then $S_i^* \geqslant S_k^* + W_{i,j} + W_{j,k}$. $W_{i,j}$ is minor compared to S_i^* or S_j^*, hence causes no significant effect on the revised size. The dependency ring is particular, because the VMs should be provided with equivalent size to start at the same time. Therefore, for $VM_i \leftrightarrow VM_j$, we determine the sizes by:$S_i^* - S_j^* = 0$.

Figure 4(b) demonstrates an inconsistent semantics graph yielded by the weighted RDG in Figure 4(a). The horizontal line shows the defacto restore order, and the arrow depicts the causal restore order. The defacto restore order is [VM2, VM4, VM3, VM5, VM1], while there exist several candidates for causal restore order, e.g., [VM3, VM4, VM2, VM1, VM5], or [VM3, VM2, VM4, VM5, VM1]. Our goal is to reorder the nodes in the horizontal line with the latest movement, to ensure consistency between the causal order and the revised defacto order, such as the example shown in Figure 4(c).

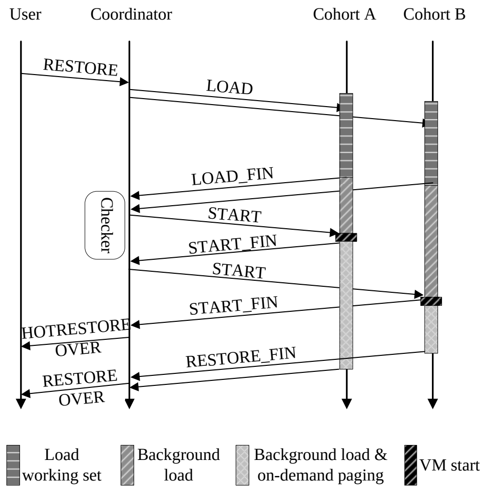


Figure 5: Restore protocol. We assume that B depends on A, so that B starts later.

Problem Formulation. We assume that the previous calculated working set size is optimal. This is reasonable since the previous size achieves well tradeoff between restore latency and performance loss for single VM, as demonstrated in the experimental results in §6.1.3. Given n VMs, let $S = \{S_1, S_2, ..., S_n\}$ be the previous working set sizes of VMs and $W = \{W_{i,j} | VM_i \rightarrow VM_j\}$ be the set of edge weight of each two communicating VMs. We aim to find a minimum revised working set size S^* to guarantee the consistency between the revised defacto restore order and causal restore order. The formulation is denoted by:

$$\begin{array}{lll} \min & \sum_{i=0}^{n} \mid S_i^* - S_i \mid & \\ s.t. & S_i^* - S_j^* \ \geqslant W_{i,j} & (W_{i,j} \in W) \\ & S_i^* - S_j^* \ = 0 & (VM_i \leftrightarrow VM_j) \end{array}$$

We use the classical Linear Programming approach [26] to solve this optimization problem. Consequently, the desired restore line is computed through arranging the VMs in ascending order of the revised working set sizes.

4.3 Restore Protocol

The purpose of restore protocol is to ensure that the virtual machines start as indicated by the restore line. The restore protocol is entirely software based and does not depend on any specialized hardware. We make the following assumptions about the distributed VMC system. First, the VMs, the underlying physical servers as well as the network devices are failure-free during restore. This is reasonable since the restore procedure lasts for a short time compared to the mean time to failure of hardware or software. Second, the network latency is minor, i.e., the messages can be received within a relatively short period. Otherwise, the restore latency would be directly related to the network latency on waiting for the messages of protocol. This assumption is satisfied in nowadays data center networks which always utilize 1Gbps or even 10Gbps Ethernet.

There exist two roles in restore protocol, Coordinator and Cohort, as shown in Figure 5. Each VM is associated with one Cohort, and only one VM is regarded as the Coordinator. The roles of Coordinator consist of constructing the restore dependency graph, scheduling the restore line, broadcasting the working set sizes of VMs and notifying Cohorts to load the working set or start to execute. Besides, the Coordinator employs Checker to determine whether the associated VM can start after receiving the LOAD_FIN reply from the Cohort, and which VMs can start correspondingly once receiving the START_FIN reply from one Cohort, as described in §4.2.1. The Cohort restores the VM through three steps: i) load the working set pages after receiving the LOAD command and then reply LOAD_FIN, ii) load the pages in background until receiving the START command, and iii) start the VM after receiving the START command, reply the START_FIN command, and then load the remaining pages through on-demand paging along with background load.

Disruption will disappear after all the VMs are started, meanwhile the hot restore is completed. The whole restore procedure will not finish until all the VMs reply RESTORE_FIN after the completion of loading the remaining memory pages in snapshot file.

5 Implementation Issues

HotRestore is implemented on qemu-kvm-0.12.5 and KVM [28] in Linux kernel 2.6.32.5-amd64, it does not require the modification of the guest OS. HotRestore utilizes HotSnap to guarantee global consistency of snapshot state. Some optimizations such as compression of zero pages are provided by HotSnap and are also employed in HotRestore. Since the page saved in background belongs to the working set if it is accessed later during snapshot procedure, HotRestore stores the working set pages in the snapshot file but creates a FAFL file which stores the guest frame number to index these pages. The FAFL file makes it convenient to fetch working set pages in any desired size. The rest of this section describes the sub-level parts and optimizations in detail.

5.1 Packets Used to Construct RDG

In reliable communication, the receiver will reply *ack* to the sender. The capture of *data/ack* packets make sender and receiver depend on each other and further form a ring in RDG. The dependency from receiver to sender is compelled, so that it should be removed when constructing the RDG. One possible approach is to identify the roles by analyzing the packets, however it is extremely complicated. Therefore, we resort to the on-the-fly packets, which cannot be received by the destination VM during *transient snapshot*. It cannot be received due to two reasons: i) the receiver VM is suspended to create *transient snapshot* or ii) the packet violates the global consistency, i.e., it is sent from a *post-snapshot* VM to *pre-snapshot* VM. HotRestore logs the on-the-fly packets on the receiver side, and constructs the RDG through these packets. Logging these packets produces two benefits. First, the on-the-fly packet will not be received or handled, and hence avoid two-way dependency for one transmission. Second, replaying these packets after restore can ensure deterministic communication, and hence the dependency is preserved.

UDP packets are also preserved. Although UDP is unreliable, the applications may support the reliability themselves through retransmission. Therefore, the network communication will be interrupted if UDP packets are lost upon restore. In conclusion, on constructing the RDG, we employ TCP and UDP packets whose source and destination are both the VMs within the VMC.

5.2 Optimizations on Restore

HotRestore adopts several optimizations to reduce the restore latency as well as performance overhead.

Sequential loading of working set pages. In HotRestore, the working set pages scatter here and there in the snapshot file and are mixed with the pages saved in background. Upon restore, HotRestore dequeues the guest frame number (gfn) from FAFL queue to index the working set page. However, the gfn order in FAFL queue is not consistent with that in the snapshot file. For example, page *A* is firstly saved in background, then page *B* is saved and recorded due to memory write operation, finally *A* is recorded due to memory read. In this case, *A* is stored in front of *B* in the snapshot file, but is behind *B* in FAFL queue. Fetching the working set pages in FAFL order would seek the pages back and forth in the disk file, thereby incurring longer latency. To solve this problem, we rearrange the working set pages by their file offset once the revised size S^* is known, so that the pages can be loaded sequentially from the snapshot file. Besides, the pages that are neighboring in the snapshot file can be loaded together to further reduce the amount of file read operations.

DMA cluster. DMA pages are touched frequently for IO intensive workloads, e.g., we observe about 280,000 DMA access in 30 seconds execution under Gzip for the VM configured with 2GB RAM. DMA page access exhibits two characteristics: First, the page will be accessed repeatedly (may be hundreds or even thousands times), therefore the amount of touched pages is actually only a few hundreds or thousands. Second, the accessed pages are always neighboring. These observations inspire us to load multiple neighboring pages (HotResotre loads 4 pages) for each on-demand DMA paging to reduce the occurrence of demand-loading, thereby reducing the performance loss incurred by page faults.

5.3 Real World I/O Bandwidth

In real world scenarios, the restore latency is related to not only the working set size, but also the available I/O bandwidth. The snapshot procedures may contend for I/O bandwidth, making the latencies various even if the file sizes are identical. As a result, the practical start times may be not as expected in restore line. In our test environments, the working set size is small while the bandwidth is sufficient, so that the restore latency is less affected. However, we believe that this problem will get worse for memory intensive workloads, especially in I/O intensive data centers. VM placement [30] and I/O schedule layer [32] are alternative approaches to mitigate the problem, and we leave this as our future work.

5.4 Non-deterministic Events

There exist many events that lead to non-deterministic system execution, they fall into two categories: external input and time [13]. The external input involves the data sent from another entity, such as the network packets from the web server, or user operations (e.g., booting a new application). Time refers to the point in the execution stream where the internal or external event takes place, for example, the receiving time of network packets. The combination of the two influences the results of several strategies resides in CPU scheduler, IO scheduler, and TCP/IP stack, and thus the system execution is diverged even the system is restored from the same snapshot point. Fortunately, compared to the individual desktop which involves multiple tasks and continual interactive operations, the distributed applications running in the virtual machine cluster are always monotonous and involves less user interaction, making the execution of the virtual machine be always deterministic.

6 Evaluation

We conduct the experiments on eight physical servers, each configured with 8-way quad-core Intel Xeon 2.4GHz processors, 48GB DDR memory and Intel 82576 Gbps Ethernet card. The servers are connected via switched Gbps Ethernet. We configure 2GB memory for the VMs. The operating system on physical server and virtual machine is debian6.0 with 2.6.32-5-amd64 kernel. We save the snapshot files in local disk, and then restore the virtual machines from snapshot files to evaluate HotRestore.

6.1 Elastic Working Set

We evaluate the elastic working set in terms of hit rate, accuracy, working set size and scalability, under several applications. The applications include: 1) Compilation, a development workload which involves memory and disk I/O operations. We compile the Linux 2.6.32-5 kernel. 2) Gzip is a compression utility, we compress the */home* directory whose size is 1.4GB. 3) Mummer is a bioinformatics program for sequence alignment, it is CPU and memory intensive [5]. We align two genome fragments obtained from NCBI [2]. 4) Pi calculation, a CPU intensive scientific program. 5) MPlayer is a movie player. It prefetches a large fraction of movie file into buffer for performance requirements. 6) MySQL is a database management system [6], we employ SysBench tool [9] to read (write) data from (to) the database. We conduct the experiments ten times and report the average as well as the standard deviation.

6.1.1 Hit Rate and Accuracy

We first measure the hit rate and accuracy under different working set sizes. We start the VM after the working set with specified size is loaded, and disable the background load thread to trace all memory accesses. If the traced page is in the working set, then a hit occurs, and the hit count increases by 1. The traced page will not be traced again, since multiple hits on the same page would increase the hit rate and accuracy. Once the count of the traced pages reaches the given size, we calculate the hit rate by the ratio of hit count to the given size. The accuracy calculation is a little different. We observe that most of the untouched working set pages will be touched within a short period. Therefore, we trace an extra 1/5 size, and calculate the accuracy by the ratio of hit count to 1.2 times given size.

Table 1 demonstrates the hit rate and accuracy of FAFL compared to LRU and CLOCK under Linux kernel compilation. It can be seen that the hit rate of FAFL is higher, e.g., for a 15K size, the hit rate of FAFL is 94.4%

	Hit Rate			Accuracy		
Size	**FAFL**	**LRU**	**CLOCK**	**FAFL**	**LRU**	**CLOCK**
5K	0.816	0.778	0.814	0.859	0.817	0.838
10K	0.845	0.875	0.749	0.945	0.918	0.926
15K	0.944	0.857	0.868	0.952	0.954	0.952
17K	0.912	0.918	0.822	0.958	0.955	0.955
20K	0.889	0.888	0.828	0.963	0.962	0.923
25K	0.870	0.861	0.869	0.962	0.963	0.970

Table 1: Hit rate and accuracy for working set with various sizes. Here, the size refers to page count, the calculated *WSS* is 17K. STDEV is minor and thus is removed.

	Hit Rate			Accuracy		
Workloads	**FAFL**	**LRU**	**CLOCK**	**FAFL**	**LRU**	**CLOCK**
Gzip	0.806	0.768	0.883	0.974	0.979	0.966
MySQL	0.947	0.655	0.912	1	0.998	1
Mummer	0.931	0.835	0.812	1	0.971	0.909
Pi	0.628	0.562	0.589	0.702	0.682	0.793
MPlayer	0.890	0.825	0.862	0.926	0.923	0.892

Table 2: Hit rate and accuracy under various workloads. STDEV is minor and thus is removed.

compared to 85.7% of LRU and 86.8% of CLOCK. This improvement is mainly due to the FAFL queue which captures the access order of VM execution more accurately. An interesting result is that the hit rate decreases as the size grows, e.g., the hit rate with FAFL decreases from 94.4% to 88.9% while the size increases from 15K to 20K. We suspect that this is because the execution suffers from larger deviation after long execution time. Despite this, the hit rate is still high. Besides, the accuracy of the three manners exceeds 95% in most cases. This fact proves that the *post-snapshot* memory tracing method captures the memory accesses accurately.

We also measure the hit rate and accuracy under other workloads. Table 2 illustrates the results when the calculated working set size is adopted upon restore. We can see that the hit rate for Gzip workload is low, this is because Gzip is I/O intensive and involves large amounts of DMA operations which always diverge after the VM is restored. For MySQL, Mummer and MPlayer workloads, the hit rate and accuracy is high due to two reasons: i) these workloads consist of a large amount of pages that hit the buffer cache which facilitate working set estimation and ii) their executions are almost deterministic. The high rate for Pi workload is poor too, i.e., 62.8%, due to the dynamic memory allocation during execution. Fortunately, the associated working set size is small, it is less than 1K pages, so that the performance loss is insignificant after the VM is rollbacked. The accuracy of these three methods are high, this means that most of the loaded working set pages will be touched by the applications after the VM is rollbacked. The accuracy for Pi workload is low, we guess that this is due to the non-deterministic execution of Pi. On average, FAFL increases the hit rate by 15.3% and 4.92% respectively compared to LRU and CLOCK.

6.1.2 Working Set Size

Here, the working set size is referred to as the total amount of loaded state including the CPU state, device state, page content and extra information such as page address. Zero page compression used in most snapshot technologies may achieve 50% reduction of snapshot size [21]; however, the reduction is specific to workload and application execution time. As a result, this optimization is disabled in this experiment. Table 3 compares HotRestore with *working set restore* [39] which calculates the size through the statistical sampling approach.

As expected, HotRestore loads less pages upon restore. Compared to *working set restore*, HotRestore reduces the working set size by 50.95% on average, therefore the restore latency is supposed to be halved accordingly. Although the restore requires extra time to sort the working set pages as well as set protection flags, the extra time is minor. Compared to the default restore method in QEMU/KVM which takes about 60 seconds to load a 2G snapshot file, HotRestore can restore the VM within 3 seconds for most workloads.

Modes	**Compilation**	**Gzip**	**Mummer**	**Pi**	**MPlayer**	**MySQL**
HotRestore	72.0(2.88)	60.9(15.3)	347.5(30.7)	1.5(0.09)	37.2(6.4)	42.4(8.9)
Working Set Restore	153.0(7.92)	113.5(21.2)	836.3(117.9)	2.76(0.17)	75.3(9.4)	87.8(11.4)
Reduction	52.94%	46.34%	58.45%	45.65%	50.6%	51.71%

Table 3: Comparison of working set sizes. The unit is MBytes, STDEV is also reported

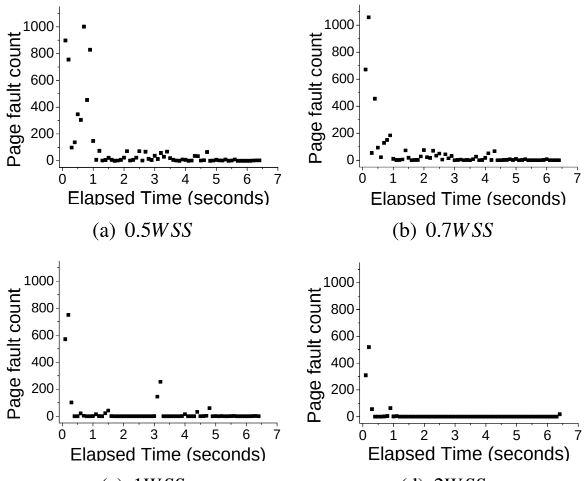
6.1.3 Scalability

The results in Table 1 have shown that the hit rate and accuracy remain high regardless of the set sizes, they therefore prove that the elastic working set can scale up or scale down without compromising the hit rate and accuracy.

On the other hand, however, scaling down the size will bring more page faults due to demand-paging and thus impose performance overhead. Therefore, the performance loss or the count of page faults should be measured and reported. We trace the page faults after restore and record the count in 100ms interval. Figure 6 shows the count on different working set sizes: the calculated WSS and its two variants, $0.5WSS$ and $0.7WSS$. WSS is 18327 pages in this experiment. As we can see, decrease of the working set size indeed incurs more page fault-s. Specifically, the numbers are 2046, 3539 and 5690 for WSS, $0.7WSS$ and $0.5WSS$ respectively during the 7 seconds tracing period. Intuitively, the increased page fault count should be equal to or approximate the saved size upon restore. However, our results show that they are not. For example, $0.7WSS$ loads 5498 less pages upon restore but incurs only 1493 more page faults compared to WSS. This is mainly because the pages stored in the FAFL queue depict the page access order. If the WSS used upon restore is less than the calculated WSS, the remaining working set pages can still be dequeued and loaded by the background load thread. This FAFL method effectively improves the hit rate compared to the approach which loads pages blindly.

The result for $2WSS$ is also given, as shown in Figure 6(d). It can be seen that the count of page faults can be further reduced due to the increase of WSS, e.g., it decreases to 958. However, we believe that the reduction is insignificant. This is because that one page fault incurs about 200us interruption which involves the time to exit to VMM, load the page from local disk, remove protection flag and resume the VM. Therefore, the overall overhead for handling thousands of page faults is negligible compared to the gain of 50% reduction on the working set size or the restore latency.

These results show that the working set can scale up/down without significant performance loss. This gives us a hint that slight working set size revision is allowable upon VMC restore. The results under other



(a) $0.5WSS$ (b) $0.7WSS$ (c) $1WSS$ (d) $2WSS$

Figure 6: Comparison of page fault count after restore.

workloads are similar, so that they are ignored due to s-pace constraints.

6.2 Restore of VMC

In this section, we evaluate HotRestore in the VMC and discuss the restore latency as well as the TCP backof-f duration. The native restore method of QEMU/KVM incurs dozens of seconds disruption, so the result is neglected here. We conduct the experiments with three restore mechanisms:

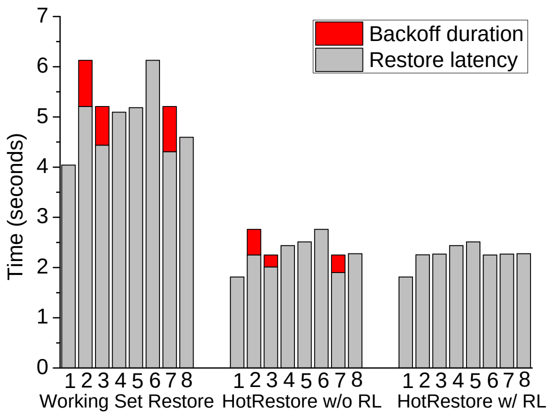
Working Set Restore. It reduces the restore latency by prefetching the working set for a single VM.

HotRestore without Restore Line (HotRestore w/o RL). It reduces the restore latency through elastic working set, but does no further work on VMC restore.

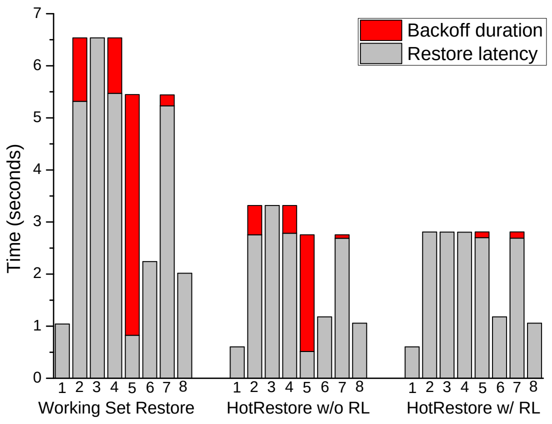
HotRestore with Restore Line (HotRestore w/ RL). It exploits the elastic working set and utilizes restore line to reduce the TCP backoff duration.

6.2.1 Detailed View of Disruption

We first illustrate the disruption duration consisting of restore latency and TCP backoff duration in detail. We setup the VMC with 8 VMs, and apply two representative network applications: i) Distcc [3] which is a compilation tool that adopts client/server architecture to dis-



(a) Disruption under Distcc



(b) Disruption under Elasticsearch

Figure 7: Comparison of disruption time in the VMC.

tribute the tasks across the nodes, and ii) Elasticsearch [4] which is a distributed, de-centralized search server used to search documents. The nodes play equivalent role in Elasticsearch. The practical TCP backoff duration is hard to depict unless modifying the guest OS, therefore, we approximate the value by the difference between start times of communicating VMs. If the VM is communicating with multiple VMs, the maximum time difference will be adopted as the duration. We use NTP to synchronize the times of physical server, so that the difference among physical times of servers is within only a few milliseconds.

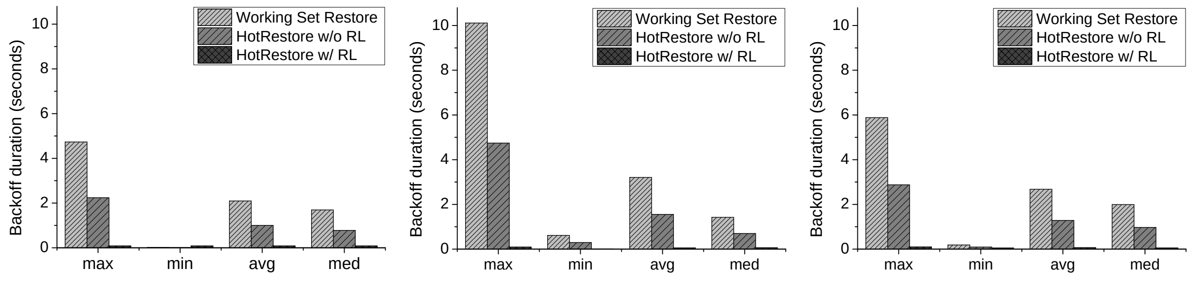
Figure 7 demonstrates the detailed disruption duration of VMs of HotRestore compared to Working Set Restore. The gray bar illustrates the restore latency, while the red bar depicts the TCP backoff duration. It can be seen that the restore latencies of VMs are various and thus cause the backoff problem. Take Distcc as an example, the latencies of VM2 and VM6 are 5.21 seconds and 6.13 seconds respectively. Since VM2 depends on VM6, hence the backoff duration of VM2 is 0.92 seconds. It can be seen that the restore latencies of VMs with Working Set Restore are much longer than that with HotRestore methods. The HotRestore w/o RL method reduces the restore latency by employing a smaller working set size, so that the backoff duration decreases accordingly. The results in Figure 7(a) show that the HotRestore w/o RL reduces the disruption by 54.1% on average compared to Working Set Restore. The HotRestore w/ RL method reduces the disruption further. Although there is no significant decrease on restore latency compared to HotRestore w/o RL, the backoff is eliminated due to the restore line. Generally, it achieves a 56.7% reduction on disruption compared to Working Set Restore under Distcc workload.

The disruption under Elasticsearch workload shows similar results. Compared to Working Set Restore, HotRestore w/o RL reduces the average disruption duration by 48.6%. It is worth noting that the TCP backoff appears in HotRestore w/ RL, e.g., the backoff duration of VM5 and VM7 are 0.1 and 0.12 seconds respectively. This is because that VM4, VM5 and VM7 form a dependency ring and are given identical working set size to be started simultaneously. However, due to the fluctuation of disk IO speed, the practical start times of VMs are different. In this experiment, VM5 and VM7 start earlier than VM4, as a result, they suffer from TCP backoff. On average, HotRestore w/ RL reduces the disruption by 53.6% compared to Working Set Restore.

6.2.2 Details on TCP Backoff Duration

The above experiments present an overview of backoff duration. We can see that some VMs do not experience TCP backoff, while others suffer from long backoff duration, e.g., the duration of VM5 is 4.49 seconds under Elasticsearch. In this section, we will exhibit the details on backoff duration, especially for the complicated network topology as the VMC scales out. We evaluate HotRestore for a VMC configured with 8, 12, 16 VMs under Elasticsearch workload, and reports the details on VMC backoff duration. There is no direct metric to measure the backoff duration for the whole VMC, therefore, we calculate all the backoff duration between each two communicating VMs, and report the maximum, minimum, average and median value.

Figure 8 demonstrates the results on backoff duration. As expected, HotRestore w/ RL achieves the least duration. Compared to the Working Set Restore approach which incurs 2.66 seconds backoff duration on average, HotRestore w/ RL reduces the average duration to less than 0.07 seconds. As explained earlier, the duration with HotRestore is mainly due to the existence of dependency ring as well as the difference of VMs' start times. Besides, we can see that the maximum backoff duration with Working Set Restore exceeds 10.1 seconds in Fig-



(a) 8 VMs (b) 12 VMs (c) 16 VMs

Figure 8: Comparison of TCP backoff duration.

ure 8(b). This fact may make the application inside the VM experience a long freeze stage even if the VM actually has already started. HotRestore solves the problem through the restore line. As we can see, even the maximum duration is less than 0.14 seconds in HotRestore. As a result, the VM after restore is able to execute without perceivable interruption.

These results demonstrate that HotRestore w/ RL reduces the TCP backoff duration to milliseconds and scales well for larger scale VMC. Besides, for low latency network where the restore protocol is not suitable, the HotRestore w/o RL approach can still work and bound the duration in a few seconds.

It is worth noting that the TCP backoff duration here is simply calculated by the difference between the start times of communicating VMs. In practice, however, the backoff duration increases twofold for each timeout, making the practical backoff duration of the Working Set Restore be much larger than the value shown in Figure 8. In other words, HotResotre would performs much better than Working Set Restore in practical scenarios.

6.3 Performance Overhead

This part will measure the incurred performance overhead of HotRestore. The overhead comes from two aspects. One is the overhead on snapshot. Compared to traditional snapshot that only traces write operations, HotRestore traces both read and write operations, so that it incurs extra overhead. Another is the overhead after restore. The demand-paging will trigger page faults, making the VM exit to VMM layer to load the desired page from disk file.

6.3.1 Overhead during Snapshot

We measure the overhead in terms of the traced page count during snapshot. Table 4 compares the count of page faults; the baseline shows the count in HotSnap approach. It can be seen that the increase incurred by HotRestore ranges from 11.3% to 124%.

Each trace causes a page fault, so that the VMM suspends the VM and handles the page fault. The overhead to handle the fault mainly consists of two parts. First, the VM exits to VMM which removes the read/write protect flag and then resumes the VM. The exit and entry of VM execution takes 38us in our platform. Second, the VMM saves the traced page into storage for memory write operation. Saving one page (4K) takes about 150us on average. Fortunately, tracing the memory read operations in HotRestore does not require saving the page. As a result, the extra time during snapshot creation incurred by tracing read operations is minor. Table 5 compares the snapshot creation time under various workloads. As an example, the total time to save the VM state in HotRestore increases by 1.3 seconds compared to that of Baseline under Compilation workload.

Modes	**Compile**	**Gzip**	**Pi**	**MPlayer**	**MySQL**
Baseline	25933	53447	1523	21510	12825
HotRestore	34348	59466	3413	30217	17598
Increase	32.4%	11.3%	124%	40.4%	37.2%

Table 4: Comparison of traced page count.

Modes	**Compile**	**Gzip**	**Pi**	**MPlayer**	**MySQL**
Baseline	85.3	79.5	54.2	72.5	77.3
HotRestore	86.6	81.1	54.4	74.2	78.2
Increase	1.3	1.6	0.2	1.7	0.9

Table 5: Comparison of snapshot duration (seconds).

6.3.2 Overhead after Restore

We setup Elasticsearch in the VMC configured with 8 VMs. We measure the performance loss in terms of the response time. Specifically, we fill Elasticsearch with 1 million micro-blogs and launch ten threads to query concurrently. Each query requests different keywords and

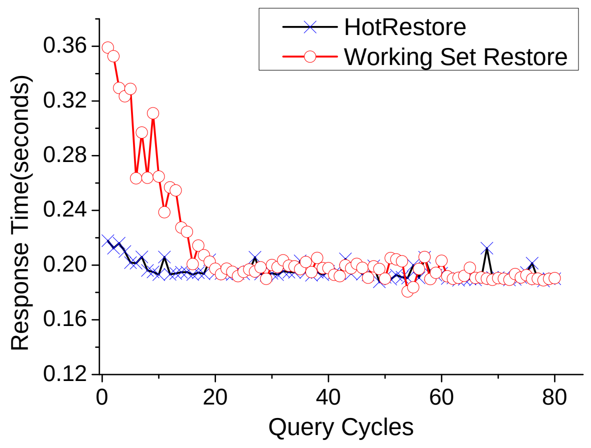


Figure 9: Comparison of response time.

acquires 40 micro-blogs. The average response time for each query is about 0.192s during normal execution.

Figure 9 demonstrates the average response time of ten threads for continuous queries. As we can see, the response time with Working Set Restore is long after restore. Specifically, the latency of the first query is 0.36 seconds, which is about twice the latency of normal execution. The reason is as follows: The Elasticsearch node will handle the query after the associated VM starts. However, some other nodes are still suspended to load the working set, thereby causing the backoff among these nodes. As a result, the requested node cannot reply as fast as in normal execution with fewer peers, especially for concurrent queries from ten threads. The response time decreases for the subsequent queries, due to the coordination with peers that are resumed. HotRestore shows no significant performance loss. The requested node can coordinate with peers in a short period after restore due to the short backoff duration. In practice, the average response time with HotRestore is 0.215 seconds for the first six queries, only a little higher than that of normal execution.

The response time with HotRestore returns to the normal value from the 7th cycle, while it keeps high until the 16th cycle with Working Set Restore. This implies that HotRestore halves the time for Elasticsearch to regain the full capability. Although the set size of certain VM is revised to be smaller and thus degrades the single VM's performance, the overall performance of entire VMC is improved due to the elimination of the network interruption.

7 Related Work

7.1 VM Restore

The idea of fast restore is not new, several approaches have been proposed to fast restore (or start) the processes and the operating systems. Recovery Oriented Computing (ROC) [33] achieves fast recovery of process upon failures by fine grained partitioning and recursive restart. Li et al. [29] track the pages touched by applications during post-checkpoint, and use these touched pages to restart the processes fast. Windows adopts SuperFetch [8] and ReadyBoost [7] to accelerate the application and OS launch times respectively by monitoring and adapting the usage patterns and prefetching the frequently used files and data into memory so that they can be accessed quickly when needed. HotRestore is fundamentally different from these works in that it focuses on the restore latency of virtual machines, rather than processes or operating systems.

There is not much work on improving VM restore. The most simple approach is *eager restore*, which starts the virtual machine after all the state including device state and memory state are loaded [37]. This approach, obviously, incurs long latency for VMs equipped with large size memory. *Lazy restore* [18] reduces the latency to milliseconds through starting the VM after CPU state and device state are loaded and loading the memory state in an on-demand way. It however incurs serious performance loss due to large amounts of demand-paging after restore, especially in the beginning execution after the VM is rollbacked. *Working set restore* [39] addresses the performance issue by prefetching the working set upon restore, at the cost of only a few seconds downtime. Our work on single VM restore shares a similar philology to *working set restore*. The difference is, their method employs working set to reduce the time-to-responsiveness metric, yet we propose an elastic working set for restoring the virtual machine cluster.

7.2 VMC Restore

There have been amounts of work about restoring a distributed system, and the key of them is to guarantee the consistency of the saved state. Several work create the global consistent state through coordinated snapshotting approach, so that the state can be loaded directly from the saved state [27, 20, 12]. Another field assumes that the nodes create snapshots independently, therefore they focus on guaranteeing the global consistency among snapshots upon restore and avoiding domino effect [16, 36, 34]. Several recently proposed snapshot systems for virtual machine cluster create the consistent global state [25, 11, 19] when snapshotting, but make no improvement on restore. We guarantee the consistency while the snapshots are being created by our prior work called HotSnap [14], and our concern within the paper is to reduce the restore latency as well as the TCP backoff duration.

Besides, the TCP backoff problem incurred during snapshot has received attention recently. VNSnap [25] saves the memory state in a temporary memory region and then flushes into disk asynchronously, thereby reducing the discrepancy of snapshot completion times to reduce the backoff duration. Emulab [11] addresses this issue by synchronizing clocks across the nodes to suspend all nodes for snapshot simultaneously. Our work emphasizes on backoff duration upon restore. By exploiting the network causal order, we propose a restore line to minimize the backoff duration.

7.3 Working Set

Working set captures the notion about memory access locality. Denning describes which pages should be swapped in and out of the working set [15]. LRU [31], CLOCK [23] and CAR [10] are improved methods to identify which pages should be replaced into the working set. HotRestore adopts the FAFL queue to record the traced pages as candidate working set pages, and then produces the working set according to the desired size. Besides, several systems adopt the working set to achieve fast restore of process [29] or VM [39]. They employ memory tracing to capture the working set pages and leverage the sampling approach [38] to estimate the working set size. HotRestore is different in that it figures out a hybrid size based on sampling during normal execution as well as statistic during snapshot, and then halves the size as the expected working set size. The size shrink does not impose significant performance loss due to the well scalability of the elastic working set.

8 Limitations and Future Work

There still exist several limitations in HotRestore. First, the proposed elastic working set would perform poor for non-deterministic applications, especially for applications in SMP virtual machines due to the variable vCPU scheduling after restore. Other non-deterministic events, as described in §5.4, will also lead to the divergence of VM execution after restore. As a result, the on-demand paging would occur frequently, and further imposes significant performance loss. Second, for some applications, e.g., long running scientific programs, where fast restore is inessential, HotRestore may incur unnecessary overhead due to extra read traces during snapshot, given that the VMC snapshot is frequently created.

Therefore, our ongoing work contain two directions. The first is to analyze the memory traces for non-deterministic applications in SMP VM for seeking a suitable page replacement method to build an accurate working set, with the aim to reduce the amount of page faults as well as eliminate performance degradation. What's more, given that snapshot is always more frequent than restore, we plan to make a holistic study on performance overhead with multiple snapshots along with one restore operations in real world scenarios, for finding a adaptive snapshot/restore policy to minimize the overall overhead for long running applications.

9 Conclusions

In this paper, we present HotRestore, a restore system which enables fast restore of the VMC without perceivable disruption. HotRestore employs an elastic working set to reduce the restore latency without compromising the performance, and proposes restore line to reduce the TCP backoff duration. The key insight in restore line is that the start times of VMs can be revised to match the network causal order of VMs. We have implemented HotRestore on QEMU/KVM platform. Our evaluation shows that the whole VMC can be restored within a few seconds, what's more, the VMs can regain the full activity rapidly after restore benefiting from the elimination of TCP backoff. We believe that HotRestore will help improve system reliability and performance after failure recovery, especially in the scenarios where failures and restore occur frequently.

Acknowledgements

We would like to thank our shepherd Theophilus Benson and the anonymous reviewers for their valuable comments and help in improving this paper. This work is supported by China 973 Program under Grant No. 2011CB302602, China 863 Program under Grant No. 2011AA01A202 and 2013AA01A213, HGJ Program under Grant No. 2010ZX01045-001-002-4 and 2013ZX01039-002-001-001, NSFC Program under Grant No. 61202424, 91118008, and 61170294, and Fundamental Research Funds for the Central Universities under Grant No.YWF-14-JSJXY-017. Jianxin Li and Tianyu Wo are the corresponding authors of this paper.

References

[1] Amazon ec2. `Http:// aws.amazon.com/ec2/`.

[2] National center for biotechnology information. `ftp://ftp.ncbi.nih.gov/`.

[3] Distcc. `http://code.google.com/p/distcc/`.

[4] Elasticsearch. `http://www.elasticsearch.org/`.

[5] Mummer. `http://mummer.sourceforge.net/`.

[6] Mysql. `http://www.mysql.com/`.

[7] Readyboost. `http://en.wikipedia.org/wiki/ReadyBoost`.

[8] Superfetch. `http://en.wikipedia.org/wiki/Windows_Vista_I/O_technologies`.

[9] Sysbench. `http://sysbench.sourceforge.net/`.

[10] S. Bansal and D. S. Modha. Car: Clock with adaptive replacement. In *Proceedings of USENIX FAST*, pages 187–200, 2004.

[11] A. Burtsev, P. Radhakrishnan, M. Hibler, and J. Lepreau. Transparent checkpoints of closed distributed systems in emulab. In *Proceedings of EuroSys*, pages 173–186, 2009.

[12] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3 (1):63–75, 1985.

[13] P. M. Chen and B. D. Noble. When virtual is better than real [operating system relocation to virtual machines]. In *Proceedings of the HotOS*, pages 133–138, 2001.

[14] L. Cui, B. Li, Y. Zhang, and J. Li. Hotsnap: A hot distributed snapshot system for virtual machine cluster. In *Proceedings of USENIX LISA*, pages 59–73, 2013.

[15] P. J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[16] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[17] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of OSDI*, pages 1–14, 2010.

[18] R. Garg, K. Sodha, and G. Cooperman. A generic checkpoint-restart mechanism for virtual machines. In *CoRR abs/1212.1787*, 2012.

[19] R. Garg, K. Sodha, Z. Jin, and G. Cooperman. Checkpoint-restart for a network of virtual machines. In *IEEE International Conference on Cluster Computing*, pages 1–8, 2013.

[20] A. P. Goldberg, A. Gopal, A. Lowry, and R. Strom. Restoring consistent global states of distributed computations. In *ACM/ONR workshop on Parallel and distributed debugging (PADD)*, pages 144–154, 1991.

[21] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: harnessing memory redundancy in virtual machines. *Communications of ACM*, 53 (10):85–93, 2010.

[22] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference*, pages 113–128, 2008.

[23] S. Jiang, F. Chen, and X. Zhang. Clock-pro: An effective improvement of the clock replacement. In *Proceedings of ATC*, pages 323–336, 2005.

[24] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay infrastructure. In *Parallel and Distributed Processing and Applications*, pages 937–946, 2005.

[25] A. Kangarlou, P. Eugster, and D. Xu. Vnsnap: Taking snapshots of virtual networked environments with minimmal downtime. In *Proceedings of DSN*, pages 87–98, 2011.

[26] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of STOC*, pages 302–311, 1984.

[27] J. Kim and T. Park. An efficient protocol for checkpointing recovery in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):955–960, 1993.

[28] A. Kivity, Y. Kamay, D. Laor, and U. Lublin. Kvm: the linux virtual machine monitor. *Computer and Information Science*, 1:225–230, 2007.

[29] Y. Li and Z. Lan. A fast restart mechanism for checkpoint/recovery protocols in networked environments. In *Proceedings of DSN*, pages 217–226, 2008.

[30] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of INFOCOM*, pages 1–9, 2010.

[31] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD*, pages 297–306, 1993.

[32] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling io in virtual machine monitors. In *Proceedings of VEE*, pages 1–10, 2008.

[33] D. Patterson, A. Brown, P. Broadwell, and et al. Recovery-oriented computing (roc): Motivation, definition, techniques, and case studies. In *Technical Report UCB//CSD-02-1175, UC Berkeley Computer Science*, 2002.

[34] D. L. Russell. State restoration in systems of communicating processes. *IEEE Transactions on Software Engineering*, 6(2):183–194, 1980.

[35] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. *Journal of Physics*, 78: 1–11, 2007.

[36] R. Strom and S. Yemini. Optimistic recovery in

distributed systems. *ACM Transations on Computer Systems*, 3(3):204–226, 1985.

[37] G. Vallee, T. Naughton, H. Ong, and S. L. Scott. Checkpoint/restart of virtual machines based on xen. In *Proceedings of HAPCW*, 2006.

[38] C. A. Waldspurger. Memory resource management in vmware esx server. In *Proceedings of USENIX OSDI*, pages 181–194, 2002.

[39] I. Zhang, A. Garthwaite, Y. Baskakov, and K. C. Barr. Fast restore of checkpointed memory using working set estimation. In *Proceedings of VEE*, pages 534–533, 2009.

[40] I. Zhang, T. Denniston, Y. Baskakov, and A. Garthwaite. Optimizing vm checkpointing for restore performance in vmware esxi. In *Proceedings of USENIX ATC*, pages 1–12, 2013.

[41] W. Zhao, X. Jin, Z. Wang, X. Wang, Y. Luo, and X. Li. Low cost working set size tracking. In *Proceedings of USENIX ATC*, pages 17–22, 2011.