Efficient Live Snapshot for Virtual Machines: A Multi-layer Coordination Approach

Jingsheng Zheng, Jianxin Li, Lei Cui, Yangyang Zhang
School of Computer Sci&Eng, Beihang University

ACT Lab

Introduction

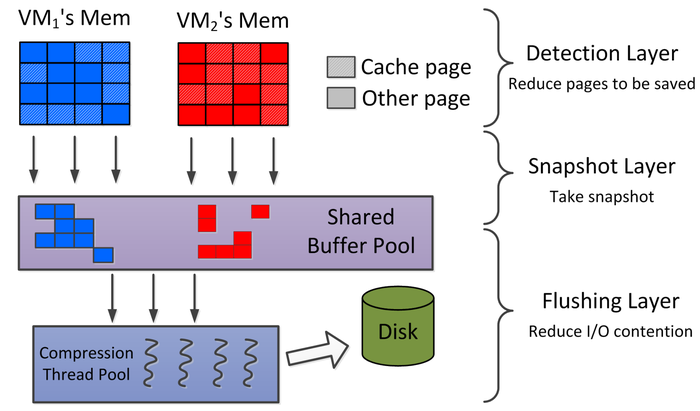
Virtual Machine Live Snapshot: :
- High availability,
- Disaster recovery
- Debugging
- System administration

Problems Statement:
- **Long downtime:** large memory size
- **Long duration:** large memory size and write-intensive workload
- **Significant performance overhead:** I/O contention between the snapshot process and the applications inside VM

Goal:
- Reduce the snapshot size, thus reduce the snapshot duration as well as the occupied storage space
- Mitigate VM performance loss

Approach:
- **I/O tracking**: detect cache pages whose contents are duplicated on disk
- **Copy-on-Write based snapshot:** save memory pages on demand and only once
- **Asynchronous compression**: avoid large amounts of I/O operations

Design Principle:

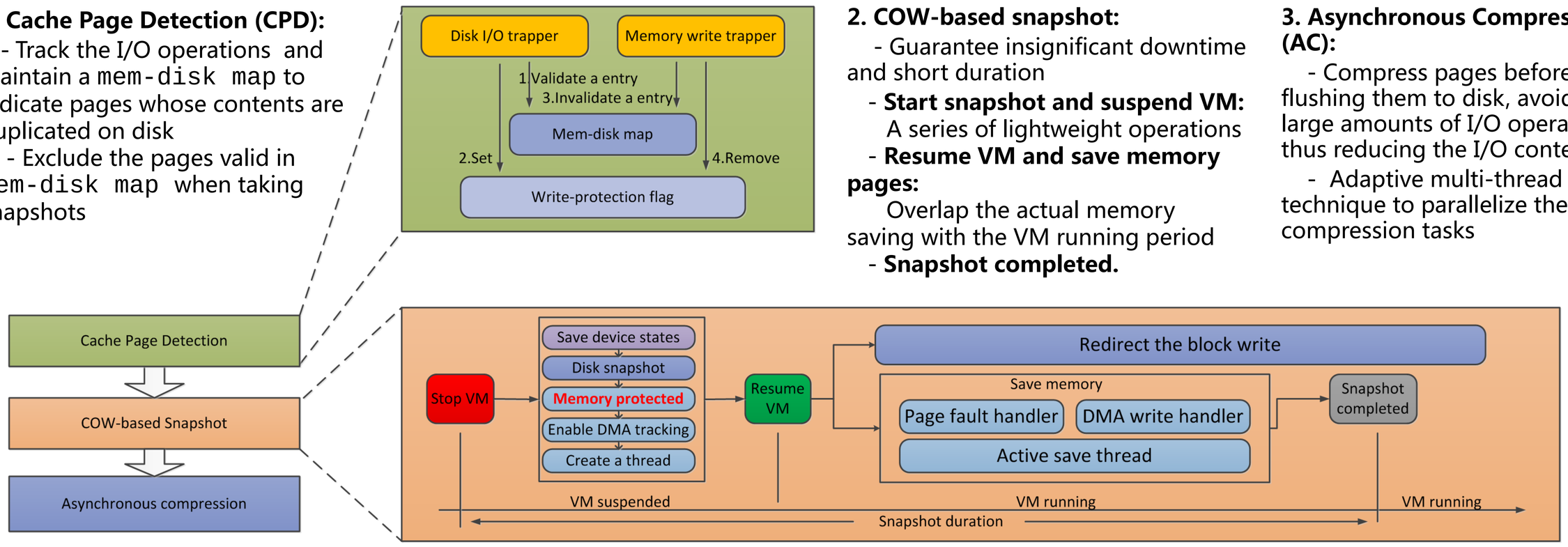


System Architecture and Design Details

1. Cache Page Detection (CPD):
- Track the I/O operations and maintain a `mem-disk map` to indicate pages whose contents are duplicated on disk
- Exclude the pages valid in `mem-disk map` when taking snapshots

2. COW-based snapshot:
- Guarantee insignificant downtime and short duration
- **Start snapshot and suspend VM:** A series of lightweight operations
- **Resume VM and save memory pages:** Overlap the actual memory saving with the VM running period
- **Snapshot completed.**

3. Asynchronous Compression (AC):
- Compress pages before flushing them to disk, avoiding large amounts of I/O operations, thus reducing the I/O contention
- Adaptive multi-thread technique to parallelize the compression tasks



Evaluation

Result of Cache Page Detection

1. Experiment Environment:
- Host Machine: 16 CPU 48GB memory Intel Xeon 2.4GHz
- Virtual Machines: 2GB memory
- Workload: Kernel Compilation

2. Performance
- Run the Kernel Compilation workload in the VM until all memory is used, and then take several snapshots at 80-second intervals
- We measure the VM performance in terms of the Compilation time.
- **Conclusion:** COW-based method can balance the VM downtime and snapshot duration; Asynchronous Compression can further decrease the performance overhead; Cache Page Detection contributes to the reduction of both duration and performance overhead significantly.

This work is supported by the 973 Program (No.2011CB302602), NSFCPrograms (Nos. 91118008, 61202424), China MOST grant (No. 2012BAH46B04), 863 project (No.2013AA01A213), NewCentury Excellent Talents in University 2010 and SKLSDE-2012ZX-21.

Efficient Live Snapshot for Virtual Machines: A Multi-layer Coordination Approach

Jingsheng Zheng[1], Jianxin Li, Lei Cui[1], Yangyang Zhang[1]
School of Computer Sci&Eng, Beihang University, {*zhengjs, lijx, cuilei, zhangyy*}@act.buaa.edu.cn

Motivations: As virtualization technology has become one of the cores of cloud computing systems, the reliability of the data and services hosted in a virtual machine (VM) becomes a top concern. Live Snapshot is a prevalent technique to provide protection for running systems by saving the running state of the VM to persistent storage. The mainstream virtualization solutions, e.g. KVM, Xen, VMware, provide snapshot technique in a *stop-and-copy* manner, which results in long and unacceptable VM downtime. Another approach is based on the *pre-copy* [1] mechanism. Although VM downtime is reduced, this method may incur longer duration particularly for write-intensive workloads. There are also some works to achieve fast snapshot [2]. However, these methods only consider one aspect of the snapshot process, and the applications inside the VM suffer from performance degradation significantly.

Design: Our key idea is to divide the whole snapshot process into three layers: *Detection Layer*, *Snapshot Layer* and *Flushing Layer*. Each layer concentrates on one specific purpose, and then they coordinate together to achieve efficient live snapshot without compromising VM performance.

Since the memory size of VM is always up to several GBs, saving the entire memory pages results in long duration and significant performance degradation. Therefore, in the *Detection Layer*, we deal with the entire memory pages to obtain a subset containing the pages which must be saved, so that the other pages can be eliminated when taking snapshots. Generally, the pages which can be discarded mainly involve the cache pages, which contain data that have been recently read from block devices. Namely, the cache pages are already present on external storage. By this pre-processing, the amount of the memory pages to be saved is reduced, which decreases the snapshot duration as well as the VM performance degradation. In the *Snapshot Layer*, we take snapshot based on the memory subset, and save the pages to a temporary buffer. Since the snapshot process mainly occupies I/O resources, the applications inside the VM can be affected significantly during the snapshot process, especially for the I/O-intensive applications. We solve this problem in the *Flushing Layer*, where a daemon running in the background to process the pages in the buffer to further reduce the data size to be flushed to the disk. Data compression is a good choice because D. Gupta et al. indicates that even if several VMs reside on a same multi-core machine, CPU resource is still rich because physical CPUs are frequently amenable to multiplexing [3]. We can exploit the abundant CPU resources to compress the memory pages before flushing them to the disk, thus avoiding the I/O contention.

In our system, we track the I/O operations of the guest to external storage and maintain a `mem-disk map` to indicate pages whose contents are duplicated on disk in the *Detection Layer*. The `mem-disk map` is indexed by the memory pages' PFN (page frame number), and each entry consists of the corresponding disk block number and a valid flag. We validate (or invalidate) an entry when trapping an I/O request (or a write-protection fault), so the valid entry in the `mem-disk map` corresponds to the redundant data.

In addition, we adopt the Copy-on-Write (COW) based snapshot mechanism in *Snapshot Layer* to guarantee insignificant downtime and short duration. We suspend the executing VM immediately when taking snapshot, followed by a series of operations, mainly involve setting all memory pages write-protected. All these operations are lightweight enough so that the VM downtime can be insignificant. Then we resume the VM, and save memory pages passively when intercepting write-protection faults or other memory writes such as DMA operations, while a background thread saves memory pages actively at the same time. We copy all the memory pages only once, resulting in a short duration.

In the *Flushing Layer*, once the buffer is not empty, the daemon fetches the memory pages from the *Shared Buffer Pool* and compresses them, followed by flushing the compressed data to the disk. In addition, we adopt multi-thread technique to parallelize the compression tasks to reduce the duration.

Evaluation: We measure the snapshot duration and performance overhead under the Kernel Compilation workload. Compared to the implementation in qemu-kvm (*stop-and-copy* mechanism), our system can reduce the duration about 69.5% while the downtime is within milliseconds, and the VM performance overhead is reduced up to 93.6%. Compared to the *pre-copy* method, the values are 88.4% and 80.2% respectively.

References:

[1] Clark, C., et al. Live migration of virtual machines. NSDI'05.

[2] Park, E., et al. Fast and space-efficient virtual machine checkpointing. ACM SIGPLAN Notices. 2011.

[3] Gupta, D., et al. Difference engine: Harnessing memory redundancy in virtual machines. Communications of the ACM. 2010.

[1] Students; Jingsheng Zheng will present. We can demo our system.