Dirty Memory Tracking for Performant Checkpointing Solutions
Lei Cao
lei.cao@stratus.com

August 25, 2016
Software Fault Tolerance

- **Checkpointing** is a technique to create a fault tolerant virtual machine by connecting a pair of servers and periodically send VM state from a primary server to a standby server
  - Checkpointing supplies a greater level of availability relative to typical HA or cluster style solutions in that failures cause no downtime and no data transaction loss.

- **This presentation** overviews checkpointing and then describes a set of KVM changes to improve checkpointing performance
Agenda

- Fault tolerance via checkpointing
- Motivation
- Design goals
- Proposed KVM Changes
- Upstream status
Checkpointing Overview

- A protected guest (OS and applications) runs inside a virtual machine
- The hypervisor contains support to:
  - Pause the VM
  - Capture static and incremental IO state
  - Capture incremental memory state
    - Pages dirtied since last checkpoint
  - Resume the VM
- The above operations are called a checkpoint
- This captured state is sent to another physical (standby) server whose hypervisor runs a paused VM with the same configuration
- In case of a failure of the active server/VM, the standby has sufficient state to resume guest operation from the last checkpoint.
Guest Run Epoch N-1 → Checkpoint N-1 → Guest Run Epoch N → Checkpoint N → Guest Run Epoch N+1

Standby Host → Checkpoint N-1 → Checkpoint N → Guest Run Epoch N+1
Active Host

Standby Host

Guest Run Epoch N

Checkpoint N

Guest Run Epoch N+1

Checkpoint N+1

Guest Run Epoch N+2

Release I/O CP N

Transmit CP N

Apply CP N

Release I/O CP N+1

Transmit CP N+1

Apply CP N+1

ACK

ACK

Time

Active Host
Known Open-Source Checkpointing Solutions

Active

- **COLO**
  - A checkpointing enhancement, needs an underlying checkpointing mechanism.
  - Originally released for Xen in 2012 by Intel/Huawei leveraging Remus
  - KVM upstream effort started in 2014 leveraging MicroCheckpointing project
  - Patch submission started 2015, project is very active with widening participation.
Known Open-Source Checkpointing Solutions

Inactive or Less-active

- **Remus**
  - Created in 2007 at the University of British Columbia (and Citrix). Accepted upstream in Xen 4.0 in 2009, no KVM activity.

- **Kemari**
  - Created in 2008 at NTT Cyber Space Labs for Xen. KVM patches created in 2010 but never upstreamed.

- **MicroCheckpointing**
  - Created in 2013 at the IBM Watson Research Center. Upstreaming activity now dormant, possibly superseded by COLO.
Known Proprietary Checkpointing Solutions

- **Vmware FT**
  - 2015 (preceded by a non-checkpointing single core version)

- **Stratus everRun**
  - Build on former Marathon MX product (released in 2010, preceded by non-checkpointing single core version), portions GPL (e.g. KVM mods) and portions proprietary.

- **Avaya Machine Preserving High Availability option for Aura® Application Enablement Services**
  - 2012, available only for Avaya environment (not general purpose)
Motivation for Proposed KVM Changes

- Checkpointing performs anywhere from >90% of a non-checkpointing VM for CPU intensive loads to 25% for high-bandwidth low-latency network intensive loads.
- Realistic commercial workloads typically perform at around 50% of a non-checkpointing VM.
- Majority of a checkpoint is spent on capturing dirty pages
Current Memory Tracking Mechanism

- Use of VM-sized bitmap to track dirty memory
- The number of dirty pages is bounded in a checkpointing system
  - For commercial workloads:
    - Number of checkpoints per second: 150 to 1500
    - Number of dirty pages per checkpoint: 300 to 3000
  - Compare to 2300k total pages (8G VM)
- Traversing a large, sparsely populated bitmap every checkpoint is time-consuming
- Copying bitmap to user space every checkpoint is time-consuming
Design Goals

- **Easily portable to various kernel versions**
  - CentOS 6.4, CentOS 6.5, CentOS 6.6, CentOS 6.7, CentOS 7.2
  - Ubuntu 14.04
  - SLES12

- **No change of existing KVM functionality**
  - New ioctls

- **Co-exist with current dirty memory logging facilities**

- **Usable by live migration as well as checkpointing**

- **Avoid dynamic memory allocation and freeing during checkpointing cycle**
  - Done when VM enters/exists checkpointing mode
Proposed Changes (1 of 3)

- **Compact lists of dirty GFNs**
  - One list per online vCPU
    - Avoid locking when vCPUs dirty memory
  - One global list
    - Pages dirtied by KVM
    - Overflow dirty pages from per-vCPU lists
  - Avoid duplicates via bitmap
    - Duplicates undesirable due to fixed size list
    - Duplicates from guest time update by KVM, PV EOI set/clear by KVM
    - Can reuse current bitmap
Proposed Changes (2 of 3)

- **Dirty log full force VM exit**
  - Number of dirty pages is bounded per epoch due to limited buffering
  - Exceeding buffer size results in expensive resynchronization
  - Force VM exit to user space when number of dirty pages reaches the threshold
  - Threshold calculated by user space and passed to KVM during memory tracking initialization
Proposed Changes (3 of 3)

- **Initialization/cleanup (KVM_INIT_MT)**
  - During initialization
    - User space indicates initialization or cleanup
    - User space specifies max number of dirty pages per checkpoint cycle

- **Activate/deactivate (KVM_ENABLE_MT)**
  - Allocate/free dirty lists
  - Enable/disable dirty traps
Proposed Changes (3 of 3) continued

- **Prepare for new checkpoint cycle (KVM_PREPARE_MT_CP)**
  - Reset the indexes/counters for all dirty lists

- **Fetch dirty list (KVM_MT_SUBLIST_FETCH)**
  - Support fetch from multiple user space threads

- **Rearm the dirty traps (KVM_RESET_DIRTY_PAGES)**
Execution flow

- Init
- Enable
- Prepare
- Enter checkpointing mode
- Checkpoint cycles
- Reset
- Fetch
- Disable
- Cleanup
- Exit checkpointing mode
How about live migration?

- The proposed changes do not break live migration
- Checkpointing mode can be used for live migration
  - Need user space support
- Improve the predictability of live migrations of memory write intensive workloads
  - Autoconverge tries to address this problem via cpu throttling
  - Cpu throttling may not be effective for some workloads where memory write speed is not dependent on CPU execution speed
Upstream Status

- **Version 1 submitted to KVM mailing list**
  - [PATCH 0/6] KVM: Dirty memory tracking for performant checkpointing and improved live migration
  - http://www.spinics.net/lists/kvm/msg131356.html

- **Version 2 planned for September submission**