Virtual Machine Checkpointing

Brendan Cully
University of British Columbia
with Andrew Warfield
University of Cambridge

Checkpointing in a nutshell

- Save the entire state of the system without shutting it down
- Similar to save followed by a restore, but…
- also need a checkpoint of *persistent* state (disk)
  - We would like to restore any checkpoint in history, and resume execution there
- it should be fast, so we can checkpoint regularly (more often than we'd save or migrate)

Why checkpoint?

- High availability
  - UBC *SecondSite* project
- Recoverable long-running processes
  - parallel computing applications
- Time-travelling debugging
- Forensics
- -1 day patching

Outline

- Background: save, restore, and migration
- Basic checkpointing (in Xen 3.0.5)
- Multi-host checkpointing
- What's cooking

Background: saving a VM

- Paravirtual approach: ask the guest for help by writing a suspend message to a location in xenstore on which the guest has a watch.
  - Guest disconnects itself from devices
  - Unplugs all but first CPU
  - Disables interrupts
  - Pins page tables of all processes into RAM
  - Prepares a *suspend record* - the *xen_start_info* structure, with the address of store and console pages converted to PFN, so that restore can rewrite them on restore
  - Makes a suspend hypercall that does not return

Saving a VM 2

- Meanwhile, back in domain 0…
  - Serialize guest configuration
  - Wait for Xen to announce that the domain has suspended
  - Map guest memory in batches and write it out with a header listing the PFNs in the batch
    - rewriting MFN in page tables to PFN as it goes (this is why all the page tables must be pinned)
  - Write out VCPU state for each VCPU, with MFN to PFN fix-ups (eg CR3)

What about disks?

- Nothing special is done to them. It is assumed the only user is the guest, and that it can recover when it is restored.
- The guest maintains a shadow copy of the request ring for recovering from device disconnects and restarts (including back-end crashes)

Restoring a VM

- Create a new domain from the serialized domain description
- Allocate and load in memory by PFN, rewriting pagetable entries from PFN to their new MFNs
- Reload VCPUs, converting CR3 to MFN. VCPU0 gets a new *start_info* record passed in via EDX
- Clear guest event queue (guest must reconnect)
- Fix up *shared_info* page (new P2M among other things)

Live migration

- Migrating a VM is as simple as piping the save image to a restore process elsewhere. The save file format is constructed to be streamable.
- *Live* migration does not suspend the guest immediately. Instead, it walks guest memory as it runs, sending pages out and then resending them as they change. The guest is finally paused when no progress is made.
- To track progress, guest memory management is put into *shadow* mode…

Shadow pagetables

- Guest page tables copied into shadow tables. CR3 is translated to use these.
- Page table updates are propagated, and page faults are fixed up according to these tables
- In live migration, shadow pages are used to track which pages are written to:
  - each PTE is marked read-only
  - If the guest should have write access to the page, the fault is fixed up, the PTE is marked RW, and a bit is set to note that this page was dirtied.
  - Calls exist to read the dirty bitmap, clean it and mark all PTEs as read-only again

Checkpointing

- Memory is handled just as in save or migrate
- Disks need more care (unless the only writeable disks are RAM disks)
- Several parts of Xen do not expect a domain to return to life after it has been shut down
- The guest can make some optimizations if it is resuming in the same domain

Disk checkpointing

- There is a xend hook for external device migration, but it was only used for the (somewhat esoteric) vTPM device.
- Extended to call external scripts at specific points for all save operations
  - 0: migration possible? (script can abort save)
  - 1: about to migrate
  - 2: domain paused: start device migration
  - 3: domain ready to resume: finish migration

Checkpointable disks

- Need some form of copy-on-write. The initial implementation used LVM. Current work is using Parallax.
- When migration reaches stage 2, the guest is paused. Start checkpointing the disk in the background (eg lvm snapshot).
- When migration reaches stage 3, make sure your disk snapshot has finished. It probably has.

Resuming after checkpoint

- Some plumbing needed:
  - when the guest returns from the suspend hypercall, it expects to be in a new domain. Either the guest must be modified, or the host must simulate a new domain for it
  - Xen needs a new domctl to make the domain schedulable again (clear the shutdown flag)
  - Xenstore needs a similar operation, otherwise it will not fire certain watches on the domain (like the shutdown watch)

Hard resume

- For old guest images, resume can be achieved by simulating a new domain
  - Tear down back end event channels and devices, then reconnect them
  - Fix up *start_info* record (PFN to MFN) by mapping suspend record and editing it in place
- Resume also provides recovery if save or migration fails

Soft resume

- Hard resume does a lot of unnecessary work (unplugging all VCPUs, disconnecting devices). Soft resume skips this.
- To tell the guest it is doing fast resume, the host process writes into the guest VCPU before returning it, setting EAX to 1.
- The host should assume that the guest does not support fast resume unless it is told otherwise. The guest advertises its support via a new ELF note in the kernel image.

Detour: ELF notes

- ELF notes are simple annotations to the kernel image, used by the domain builder to determine which features the guest supports or requires (eg PAE mode).
- Checkpointing requires this information *after* the domain has been built. The builder API has been extended to return ELF notes to xend, which parses them and enters them into XenStore.
- On checkpoint, xend passes the ‘soft resume’ flag to the checkpoint process, which then takes the faster path.
- ELF notes are also serialized into the image file, and loaded from it into xenstore when the image is restored.

Restoring a checkpoint

- The backend devices are encoded in the image file. They need to be set to the checkpoint device name, either while writing the image or when restoring (probably easier to coordinate)
- One simple approach is to rewrite the S expression into which the domain configuration has been serialized. I also have unreleased patches to allow command-line configuration overrides for xm restore, as is possible with xm create.

Variation: rolling checkpoints

- If checkpoints are taken frequently, it is a good idea not to start over at each epoch – the majority of RAM is unlikely to have changed.
- So leave shadow mode enabled after the first checkpoint, and begin with the pages that have been dirtied, using the previous checkpoint as a reference point.
- At intervals, the checkpoint collector applies the differentials to the previous base to create a new base checkpoint.

Multi-host checkpointing

- Goal: checkpoint multiple domains “at the same time”
- Nodes may run on very loosely synchronized clocks. So “the same time” is not easy.
- Solution: Self-synchronizing (Lamport) clocks.
- Nodes are checkpointed independently unless they communicate with each other

Lamport clocks

- A simple algorithm first described by Leslie Lamport about 30 years ago:
- Every node maintains its own timestamp (checkpoint epoch).
- Every message from a node embeds the checkpoint epoch of the sender.
- A message is not received until the recipient's timestamp is equal to or greater than the message timestamp

Checkpoint synchronization

- Hosts are networked via software bridge in domain 0 (the default arrangement)
- A standard ebtables rule (updated each epoch) translates the source MAC address of outbound packets to a nodeID/epoch tuple.
- A custom ebtables rule on inbound packets transforms back to source address. If the epoch of the packet is greater than the epoch of the rule, it queues the packet and initiates a checkpoint via a xenstore message.

Putting it together

- Daemons run in domain0, issuing independent checkpoint requests at a configured interval to each node
- If a node receives a message with a higher epoch (because another node finished its checkpoint earlier), it queues the message and forces an immediate checkpoint.
- Checkpoints are taken in the live migration style (minimal downtime), but a synchronization message causes an immediate jump to the final round (pause and copy).

What’s cooking

- Copy-on-write checkpointing
- Deadline scheduling
- Make it faster

COW Checkpointing

- Do not pause domain to create final consistent image. Instead, mark pages as COW and copy the originals into a send buffer provided by the migration task, if and when the guest touches them.
- Migration tool notifies Xen as it copies out pages, marking them RW again if they haven't been touched, or returning the copied page to the pool.
- Introduce checkpoint barrier into device request queue to ensure old requests are written to old device, new to new.

Deadline scheduling

- Estimate work to be done. Factors:
  - available bandwidth
  - uncopied pages
  - recent page dirty rate
- Start copy at deadline - estimate. If early, a little more copying is done. If time looks short, slow down or pause the guest
  - increases bandwidth
  - decreases page dirty rate

Making it faster

- Current checkpoint mechanism involves many synchronization points that could be folded together:
  - Many synchronous forks for external device migration (4 stages for each attached device)
  - Xenstore callbacks from host to guest and back to initiate final round, and from bridge to checkpoint daemon
- Dirty pages can be compressed and hashed to reduce IO load

Conclusion

- Basic checkpointing in Xen 3.0.5
  - xm save --checkpoint
- Multi-host checkpointing in the kick-the-tires stage
- COW/deadline checkpointing later this summer

fin