KVM Live Migration

Uri Lublin, Qumranet
Anthony Liguori, IBM
Agenda

- Introduction
- Algorithm
- Migration Protocols
- How to add migration support for new devices
- Using it
- Summary/Merits compared to other hypervisors
- Future Work
Introduction

- Live Migration, A valuable feature of any hypervisor
  - Almost unnoticeable guest downtime
  - Load Balancing, Maintenance, Hardware Upgrades
    Software Upgrades
- Guest is not involved
- Capable of tunneling VM State through an external program
- Short and Simple
- Easy to Enhance
- Hardware (almost) independence

KVM Forum 2007
Algorithm

1. Setup
2. Transfer Memory

KVM Forum 2007
Algorithm

1. Setup
2. Transfer Memory

Host A

Host B

Whole Memory

1 -- 133000

KVM Forum 2007
Algorithm

1. Setup
2. Transfer Memory

Host A

Dirty Pages

Host B

2 -- 5800
Algorithm

1. Setup
2. Transfer Memory
3. Stop the VM

KVM Forum 2007
Algorithm

1. Setup
2. Transfer Memory
3. Stop the VM
4. Transfer State

KVM Forum 2007
Algorithm

1. Setup
2. Transfer Memory
3. Stop the VM
4. Transfer State
5. Continue VM
Algorithm

1. Migration Request arrives (migrate VM from A to B)
   - Spawn external command (If applicable)
   - Connect (if applicable) and send header
   - Allocate resources + Setup
Algorithm

1. Migration Request arrives
2. Transfer Memory
   - First transfer all memory pages (first iteration)
   - For every next iteration I transfer all dirty pages of iteration i-1
   - Until convergence
Algorithm

1. Migration Request arrives
2. Transfer Memory
3. Stop the VM
Algorithm

1. Migration Request arrives
2. Transfer Memory
3. Stop the VM
4. Transfer VM State
   - Each device “transfer” its own state
   - Dirty pages (from the last iteration) included
Algorithm

1. Migration Request arrives
2. Transfer Memory
3. Stop the VM
4. Transfer VM State
5. Continue the VM
   • On remote (B) if migration was successful
     • Send (broadcast) an Ethernet packet to announce the new location
   • On local host (A) if migration failed
Memory Transfer

- Requires support for dirty memory page logging
- Homogeneous page optimization
- Rapidly written pages / Writeable Working Set
- Dynamic Bandwidth Limitation
Dirty Page Logging

- **Qemu**
  - One byte per page – supports up to 8 different dirty types
  - Devices that write directly into guest memory must update the dirty-byte-map

- **KVM**
  - One bit per page
  - Pages are mapped RO to intercept first-write
  - Enabled/Disabled when migration begins/ends
  - Merged with Qemu’s dirty log before every memory transfer iteration
State Save/Load

- Qemu devices register save/load functions
- Functions are called upon VM save/load/migrate
- Versioning/Backwards-Compatibility
- KVM State (register values) is synchronized with qemu as part of cpu and other devices’ state-functions
Memory Transfer Convergence Rules

- Transitions the algorithm from live phase to offline phase, according to the following rules:

- Convergence: $N_1 = 50$ dirty pages (or less) left

- No Progress: $N_2 = 2$ iterations where the number of transferred memory pages is smaller than the number of pages that got dirty.

- Hard limit: $N_3 = 30$ iterations passed
Convergence example: fc6 running httpd

Number of pages sent per iteration during fc6-httpd live migration

KVM Forum 2007
WWS example: fc6 running httpd

Histogram: Number of times a pages was sent during fc6-httpd live migration

KVM Forum 2007
End Of Migration Protocol (for tcp://)

- **Goal:** prevent a case where a guest continues to run on both hosts

- **Algorithm:**
  - A transfers state to B and waits for ACK
  - B receives state, sends ACK and waits for GO
  - A receives ACK, and sends GO
  - B receives GO and continues
  - Upon any timeout (lost messages), migration fails

- **Worst Case: Go was lost**
  - VM does not run (on any host)
  - A stops, B exits
  - Third party (management) intervention required
Migration Support for new devices

- If the new device writes directly to guest memory – update byte-map-log

- If some syncing needs to be done before/after state transfer, register to get VM stop/cont events

- Register save/load state function

- Don’t forget versioning (support for backwards compatibility)
Migration Support for PV drivers

- PV host-side must
  - Save/Load its state
  - Make sure guest state is valid on remote host

- Use Non-Locking guest-host synchronization mechanism.
  - Such as rings.
  - Guest must never be stopped while holding a guest-host shared lock.

- Keep hypercall-calling a single command
Qemu monitor commands and cmd line

- (qemu) migrate [–d] <migration_protocol:params>
- On remote <kvm-cmd-line> –incoming <protocol:params>
- /usr/bin/kvm –m 512 –hda /images/a.img –incoming stdio
- (qemu) migrate_set_speed <bytes_per_second>

- (qemu) migrate_cancel

- (qemu) info migration
Migration Protocols – Use Cases

- Using TCP sockets
  - Migrate tcp://remote:port
  - -incoming tcp://0:port

- Built In ssh support
  - Migrate ssh://remote

- Save image to file
  - Migrate “exec: dd of=STATEFILE”
  - -incoming file://STATEFILE

- Compress using gzip (bzip2)
  - Migrate “exec: gzip –c > FILE.gz”

- Encrypt using gpg into a file or to remote
  - Migrate “exec: gpg –q –e –r KEY –o FILE.gpg”
  - Migrate “exec: gpg –q –e –r KEY | nc remote port”
    - Nc –l port | gpg –q –d –r KEY | <kvm-cmd>
Merits compared to other hypervisors

- Short and Simple
- Built in security using ssh
- Guest is not involved
- Hardware independence
- Migration of stopped guests
- Tunneling/Flexibility/Extensibility
- Compression/Encryption
- Backwards Compatibility
- Upon Failure, guest continues to run on source host.
- Open

KVM Forum 2007
Future Work

- Support for more migration protocols
  - Live Checkpoints, compression, encryption, file
  - Partial support for unknown migration protocols
- WWS optimization
- Fine tune parameters
- Deal with device assignment/direct access.
- Migrate to a remote location
- Migrate disk as together with state.
- Support for new features of kvm (e.g. pass through)
Thank You 😊