RDMA Migration and RDMA Fault Tolerance for QEMU

Michael R. Hines
IBM
mrhines@us.ibm.com

http://wiki.qemu.org/Features/RDMALiveMigration
http://wiki.qemu.org/Features/MicroCheckpointing
Migration design / policy Problems

• Admins want to evacuate any size VMs
  - Tens of GB / Hundreds of GB
  - Arbitrary storage configurations
  - Arbitrary processor configurations

• Management software is still pretty blind:
  - Is VM idle?
  - Busy?
  - Full of zeros?
  - Mission Critical?
  - System software wants our cake and eat it too
• Customers don’t (yet) want to re-architect
  - Just make my “really big JVM” or my “honking DBMS” run
  - Don’t ask me any questions about the workload
  - But willing to tell you “high-level” workload characteristics
  - Co-workers keep telling me security is important,
    probably don’t want any “extreme” visibility anyway

• Customers do *want* high availability
  - But they don’t really trust us (much)
    - They think we’re really good at running workloads
    - Not so sure we’re good at *moving* workloads
  - Also don’t want to re-architect
  - Don’t want to put all their eggs in one basket
    - Still very willing to spend money on mainframes

• “PaaS” not a panacea, but has the potential to be

High Availability design / policy problems:
State of the art

• **Migration**: 40 Gbps *ethernet* hardware already on the market
  - Faster ones not far away
  - Can we do TCP @ 40 Gbps?
    - Sure, at 100% CPU – that’s not good

• **Fault Tolerance:**
  - A.K.A: Micro-Checkpointing / Continuous Replication
  - Multiple hypervisors are introducing it:
    - Remus on Xen
    - VMWare lockstep
    - Marathon / Stratus EverRun
  - Where is KVM in all this “HA” activity?
Smaller active set of physical servers
  evacuate low utilization servers

Higher utilization for active servers
  lower headroom requires rapid rebalance when workload spikes

A decrease in energy use –
  rapidly adjust servers online to load variation to preserve SLA

Customers buy/install servers for peak loads
  fast VM migration allows dynamic size to actual

Unsolved Needs
35 IBM proprietary VMs sampled every 5 minutes (WebSphere, DB2) over 24 hours
• RDMA usage:
  - Memory must be registered on both sides
  - Small RDMA transfers are kind of useless
    - IB programming is like IPv6:
      - Necessary evil, but proven technology
    - RDMA over Ethernet products from multiple Vendors

• QEMU:
  - Avoid changing the QEMUFile abstraction – many users
  - Avoid re-architecting savevm.c/arch_init.c
  - Avoid “mixing” TCP and RDMA protocol code –
    - They are really mutually exclusive programming paradigms
  - One gives you a bytestream and one does not
Migration and VM stop times as a function of migration method: VM is running SPEC CPU2006 benchmark gobmk --rate 10; migrated @ 1.5 minutes; approx 700MB VM footprint; VM --smp 2 --mem 3072

Results
 RDMA Fault Tolerance

Status

- Slow going, but steady

1. Asynchronous Micro-checkpointing + I/O buffering + RDMA
2. Checkpointed TCP/IP traffic
3. Protected DB Server VM
4. Node 1
5. Node 2

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory / CPU</td>
<td>Done, ~10-20% penalty</td>
</tr>
<tr>
<td>Network Buffering</td>
<td>Done, ~50% penalty</td>
</tr>
<tr>
<td>Storage Buffering</td>
<td>In Progress</td>
</tr>
<tr>
<td>RDMA Acceleration for I/O</td>
<td>In Plan</td>
</tr>
<tr>
<td>and Memory</td>
<td></td>
</tr>
<tr>
<td>Application Awareness</td>
<td>Not yet in plan</td>
</tr>
</tbody>
</table>
• **Sub-millisecond-FT requires consistent I/O**
  - Outbound network must be buffered
  - Outbound storage must be mirrored

• **Expensive Identification of dirty memory**
  - Multiple KVM log_dirty calls => QEMU bitmap
  - Multiple QEMU bitmaps => QEMU bytemap
  - Bytemap used by VGA? PCI? Others?
  - Bytemap => single bitmap
  - Stream through bitmap

• **60 milliseconds to identify dirty memory on a 16GB vm!** Ahhhhh!
Network Barriers - how it works with Micro-Checkpointing (MC)?
Micro-Checkpointing “barometer”: where do the costs come from? Average checkpoint size < 5MB

- Transmit (VM Running)
- Local Copy to Staging (VM Stopped)
- Prepare Bytemap => Bitmap (VM Stopped)
- GET_LOG_DIRTY + bytemap (VM stopped)
• Very “sensitive” topic =)
  • Myth: Sub-second, sub-working-set over-commitment
  • Reality: Sub-day, sub-hour, working set growth and contraction
• RDMA Migration:
  • Overcommit on both sides of connection?
  • Again: Does your management software know anything?
    • Lots of zero pages? VM is idle? Critical?
    • Why can’t our management software have APIs that libvirt can share?
      Or maybe OpenStack can share?
    • Does policy engine know when to use RDMA?
• RDMA Fault Tolerance:
  • Checkpoints arbitrary in size – double the VM in the worst case
  • Checkpoints also need to be compatible w/ RDMA

Support for Over-commitment
You (or your policy management) is dealing with a *known* memory bound workload that doesn’t converge with TCP

- **Migration Time = O(dirty rate)**
  - Where Dirty rate > TCP bandwidth
- You have sufficient available free memory on the destination host
- Your source host cannot tolerate stealing CPU cycles from VMs for TCP traffic

When *should* you use RDMA?
1. Is your VM idle?
   - Migration Time = $O(bulk\ transfer)$
   - Is your VM “young”? (Full of zeros?)
2. - Migration Time = $O(almost\ nothing)$
3. Your known-memory-bound application doesn’t converge AT ALL
   - Migration Time = $O(infinity)$
   - It is so heavy, you need “auto-converge” capability (new in 1.6)
1. You have no idea what the workload is
   But “someone” told you (i.e. paid) that it was important
2. Or you know what the workload is, but its memory content is mission-critical
3. I/O slow down is not a “big deal”
4. You must have high-availability
5. Without re-architecting your workload

When should you use FT?
1. Your workload can already survive restart
2. Your crashes are caused by *guest* bugs, not host bugs or hardware failures
3. Your workload is stateless
4. Your workload cannot tolerate poor I/O
5. You’re customer is willing to tightly integrate with the “family” of HA tools: HACMP / Pacemaker / LinuxHA / Corosync / Zookeeper ….. The list goes on

When you should *not* use FT?
• RDMA knows what a “RAMBlock” is
• New QEMUFileOps pointers:
  - `save_page()`
    Allows for **source** ‘override’ of the transmission of a page
  - `before_ram_iterate()`
    Allows for **source** to perform additional initialization steps *before* each migration iteration
  - `after_ram_iterate()`
    Similar **source** override, but at the end of each iteration
  - `hook_ram_load()`
    Allows the **destination** override the *receipt* of memory during each iteration
• QEMUFile operations:
  - Translate into IB Send/Recv messages
• qemu_savevm_state_(begin|iterate|complete)
  - Invoke previous hooks (before_ram/after_ram)
  - Hooks instruct destination to handle memory registration on-demand
• ram_save_block()
  - Invokes save_page()
  - If RDMA not compiled or enabled, then we fallback to TCP
• Checkpoint size = O(2 * RAM), worst case
  Cannot allow QEMU to hog that much RAM
  Cannot register that much with RDMA anyway

• Solution:
  Splitup checkpoint into ‘slabs’ (~5MB each)
  Grow and evict slabs as needed
  Register slabs as they are needed

(Implemented) FT Technical Summary (2)
• Network ‘output commit’ problem:
  - Xen community has released their network buffering solution into netlink community:
    - Output Packets leave VM
    - End up in tap device
    - Tap device dumps packets into IFB device
    - Turns ‘inbound’ packets into ‘outbound’ packets
    - Qdisc ‘plug’ inserted into IFB device
    - Checkpoints are coordinated with ‘plug’ and ‘unplug’ operations
• Plan is to active QEMU ‘drive-mirror’
  ‘drive-mirror’ would also need to be RDMA-capable
  Local I/O blocks go to disk unfettered

• Mirrored blocks:
  • Held until checkpoint complete, then released
  This was chosen over shared storage
    For performance reasons
    And ordering constraints

Not implemented: FT storage mirroring
Drive-mirror + MC

Protected VM
virtio-frontend

Virtio-backend

MC-thread

Drive-mirror
(Direct I/Os)

Mc Buffer
Control Signals

Disk writes

Linux / KVM

Local Storage

Backup VM
virtio-frontend

Virtio-backend

Drive-mirror
(Buffered I/Os)

Checkpoint
(TCP or RDMA)

Mirrored
writes

Linux / KVM

Local Storage
Thanks!
Backup Slides
Challenges:
- REST command does not return after FT is initiated => permanent thread must get kicked off
- Openstack has no concept (yet) of VM Lifecycle management or Failure
- Openstack would also need storage-level compatibility with our FT storage approach
- Who owns storage after failure?
- How to recalculate cluster resources?
- How to keep mysql consistent?

Controller Node

$ > ./cb vmprotect
$ > nova migrate

Compute Node 1

---

Compute Node 2
Sequence of Events:

1-b) Insert I/O barrier A
2) Stop virtual machine
3) Capture checkpoint
4) Insert I/O Barrier B
5) Resume virtual machine
6) Transmit Checkpoint

1) VM running

1-a) ACK / Ready for next checkpoint

7) Receive checkpoint

8) Transmit ACK of checkpoint

 VM not running

 6)

9-a) Apply checkpoint

 VM running

11) Release I/O Barrier A

10-b) Receive ACK
Failure modes: I/O

Barrier A lost, revert checkpoint

Sender

1-b) Insert I/O barrier A
2) Stop virtual machine
3) Capture checkpoint
4) Insert I/O Barrier B
5) Resume virtual machine
6) Transmit Checkpoint

Receive

1-a) ACK / Ready for next checkpoint
7) Receive checkpoint
8) Transmit ACK of checkpoint
9-a) Apply checkpoint
10-b) Receive ACK
11) Release I/O Barrier A
Failure modes: I/O

Barriers A & B lost, revert checkpoint

1) Stop virtual machine
2) Capture checkpoint
3) Insert I/O Barrier B
4) Resume virtual machine
5) Transmit Checkpoint
6) VM running

1-b) Insert I/O barrier A
2) Stop virtual machine
3) Capture checkpoint
4) Insert I/O Barrier B
5) Resume virtual machine
6) Transmit Checkpoint
7) Receive checkpoint
8) Transmit ACK of checkpoint
9-a) Apply checkpoint
10-b) Receive ACK
11) Release I/O Barrier A
Failure modes: I/O
Barriers A & B lost, revert checkpoint

<table>
<thead>
<tr>
<th>Time</th>
<th>Sender</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-b) Insert I/O barrier A</td>
</tr>
<tr>
<td>2</td>
<td>2) Stop virtual machine</td>
</tr>
<tr>
<td>3</td>
<td>3) Capture checkpoint</td>
</tr>
<tr>
<td>4</td>
<td>4) Insert I/O Barrier B</td>
</tr>
<tr>
<td>5</td>
<td>5) Resume virtual machine</td>
</tr>
<tr>
<td>6</td>
<td>6) Transmit Checkpoint</td>
</tr>
<tr>
<td>7</td>
<td>VM running</td>
</tr>
<tr>
<td>8</td>
<td>VM running</td>
</tr>
<tr>
<td>9-a)</td>
<td>9-a) Apply checkpoint</td>
</tr>
<tr>
<td>10-b)</td>
<td>10-b) Receive ACK</td>
</tr>
<tr>
<td>11</td>
<td>11) Release I/O Barrier A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Receive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-a)</td>
<td>1-a) ACK / Ready for next checkpoint</td>
</tr>
<tr>
<td>7</td>
<td>7) Receive checkpoint</td>
</tr>
<tr>
<td>8</td>
<td>8) Transmit ACK of checkpoint</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>
I/O barrier B lost, revert checkpoint.

- 1-b) Insert I/O barrier A
- 2) Stop virtual machine
- 3) Capture checkpoint
- 4) Insert I/O Barrier B
- 5) Resume virtual machine
- 6) Transmit Checkpoint

VM not running

- 7) Receive checkpoint
- 8) Transmit ACK of checkpoint
- 9-a) Apply checkpoint
- 10-b) Receive ACK
- 11) Release I/O Barrier A

VM running

- 1-a) ACK / Ready for next checkpoint

Failure modes: I/O barrier B lost, revert checkpoint.

VM not running

- 6

VM running

- 8

Time

31