Xen VMs, Virtual Clusters and Programmatic Partitioning
Xen Overview
What is Xen

- Xen is “virtual machine monitor” (VMM) used to control VMs

- Xen VMM is called the “hypervisor”

- Xen VMs are called “guests”
What is Xen

- Guest traps and exceptions are passed to and handled by hypervisor
But Xen in HPC?

- The performance issues are with I/O
- Interconnects
  - With Myrinet, one can assign the card to one domain at a time
  - With IB, ‘VMM-bypass’ for RDMA support
    - Mellanox has alpha version software that attains “near-native I/O performance”
- Using disk partitions or logical block devices can increase disk I/O
Xen in Rocks 5.0
Step 0

- You must install a Rocks 5.0 frontend with the Xen Roll
Step 0.5

- You must install at least one cluster node as a “VM Container”
Supported Configuration

◆ Frontend
  ✐ Normal “physical” frontend
    • No xen kernel or xen tools are installed

◆ “VM Container” appliance houses Xen VMs
  ✐ VM Container is dom0
Supported Configuration

[Diagram showing a supported configuration with Frontend, vm-container-0-2, vm-container-0-1, vm-container-0-0, compute-0-0-0, compute-0-0-1, compute-1-2, compute-1-1, compute-1-0]
Key VM Functions

◆ “add”
  ➔ Add a new VM to the cluster

◆ “create”
  ➔ Install a VM

◆ “start”
  ➔ Boot a VM
Adding a VM

- "rocks add host vm" command
  ```
  # rocks add host vm {physical machine} \ 
  membership={rocks membership}
  ```

- "rocks membership" is one of:
  ```
  # rocks list membership
  MEMBERSHIP          APPLIANCE
  Frontend:           frontend
  Ethernet Switches:  network
  Power Units:        power
  Remote Management:  manager
  NAS Appliance:      nas
  Compute:            compute
  VM Container:       vm-container
  ```
Adding a VM

Example

```
# rocks add host vm vm-container-0-0 \ 
membership="Compute"
```

Output:

```
added VM on node "vm-container-0-0" slice "0" with vm_name "compute-0-0-0"
```
Adding a VM

- “rocks add host vm” command adds entries to the nodes and networks tables
- Allocates a unique MAC address for the VM
  - Using the Xen reserved prefix: 00:16:3e
- Adds an entry to the vm_nodes table
  - Keep track of which physical host houses the VM
- Adds an entry to the vm_disks tables
  - Allocates disk space for the VM
    - Uses the Xen “file” virtual block device
    - Puts file on the largest partition of the physical host
Install a VM

- “rocks create host vm” command
  
  # rocks create host vm compute-0-0-0

- This starts a standard Rocks installation on the VM
Install a VM

- After the networking stack is initialized and anaconda is up, you can monitor the install with “rocks-console”

  # rocks-console compute-0-0-0

- Just like a physical compute node!
Boot a VM

- After the VM is installed, boot it:
  
  ```
  # rocks start host vm compute-0-0-0
  ```

- About 30 seconds later, login to it with “ssh”.
  
  - Just like a physical compute node!
Other Rocks Xen Commands
list

◆ List info about all configured VMs

```
# rocks list host vm
VM-HOST     SLICE MEM  CPUS  MAC                  HOST                STATUS
compute-1-4-0: 0 900 1  00:16:3e:00:00:08  vm-container-1-4 active
compute-1-3-1: 1 900 1  00:16:3e:00:00:07  vm-container-1-3 active
```
set

◆ Change VM parameters

# rocks set host vm {host} [disk=string] [disksize=string] \ 
  [mem=string] [physnode=string] [slice=string] \ 
  [virt-type=string]

◆ Example, allocate 4 GB of memory to a VM:

  # rocks set host vm compute-0-0-0 mem=4096
pause/resume

* Execute the “pause” and “resume” Xen commands on a VM

  # rocks pause host vm compute-0-0-0
  # rocks resume host vm compute-0-0-0
save/restore

- Execute the “save” and “restore” Xen commands on a VM
  
  ```
  # rocks save host vm compute-0-0-0
  # rocks restore host vm compute-0-0-0
  ```

- What’s the difference between “pause” and “save”?  
  - “pause” keeps the VM in memory  
  - “save” writes VM state to a file and releases memory and CPU
stop

- Destroy a VM

  `# rocks stop host vm compute-0-0-0`

- This is equivalent to pulling the power cord on a physical machine
move

- Move a VM from one physical node to another
  
  # rocks move host vm compute-0-0-0 vm-container-1-0

- This operation will take some time
  
  - It “saves” the current VM
  - Copies the VMs disk file to the new VM container
    - If your diskfile is 36 GB, it will move 36 GB across the network
  - Then “restores” the VM
Other “Internal” Commands

◆ “dump”
  ➤ Used on the restore roll to capture VM configuration

◆ “report”
  ➤ Called by “rocks create host vm” and “rocks start host vm” to create Xen VM configuration files

◆ “remove”
  ➤ Called by “rocks remove host” to remove the VM specific info for a host
Xen Debugging Tool

- Use “virt-manager”
- Login to VM container and execute:

  # virt-manager
Virt-manager

- Double click on ‘compute-0-0-0’ to bring up console
Virt-manager
Xen in Rocks Futures
Futures

- Run Xen on the frontend
  - The frontend physical machine is a VM Container
  - Frontend functionality runs within domU

- Support multiple virtual clusters on one physical cluster
  - Need to add VLAN support to Rocks Xen support

- Hardware virtualization support
  - Can run any OS within a domU
Programmatic Partitioning
How I Feel About Partitioning
The Problem With Rocks
Partitioning In The Past

- Too hard to enforce user-specified partitioning onto nodes
  - Rocks defined ‘<part>’ XML tags
- Too hard to define different partitioning schemes for different nodes or appliance types
  - Had to build new appliance types, had to build new distributions, etc.
Goals of Rocks Partitioning

- Don’t lose user data
  - Save partitions in database
  - Mark ‘seen’ disks (.rocks-release)

- Write partitioning specification once
  - In the past, you’d:
    - Write an XML file with <part> tags
    - Rebuild distro
    - Reinstall nodes
    - Remove XML file
    - Rebuild distro
Goals of Rocks Partitioning

- Make it easy for the user to reason about the partitioning scheme
- Flexible framework that allows fine-grained control
How It Works

- In a <pre> section, the user populates RedHat-specific partitioning directives into a file named:

  /tmp/user_partition_info
Example 1

◆ Create an XML file:

```bash
# cd /home/install/site-profiles/5.0/nodes/
# cp skeleton.xml replace-partition.xml
```
Example 1

Create a `<pre>` section:

```<pre>
  echo "clearpart --all --initlabel --drives=hda
  part / --size 8000 --ondisk hda
  part swap --size 1000 --ondisk hda
  part /mydata --size 1 --grow --ondisk hda" > /
  /tmp/user_partition_info
</pre>
Example 1

- **Rebuild the distribution:**
  
  ```bash
  # cd /home/install
  # rocks-dist dist
  ```

- **Remove old partitioning from database:**
  
  ```bash
  # rocks remove host partition compute-0-0
  ```

- **Remove ‘seen’ marker on compute node**
  
  Rocks will not reformat a disk that has .rocks-release on any partition in that disk
  
  ```bash
  # ssh compute-0-0
  # rm /.rocks-release
  ```

- **Reinstall:**
  
  ```bash
  # shoot-node compute-0-0
  ```
Software Raid Example

Create a `<pre>` section:

```bash
<pre>
echo "clearpart --all --initlabel --drives=hda,hdb
part / --size 8000 --ondisk hda
part swap --size 1000 --ondisk hda

part raid.00 --size=10000 --ondisk hda
part raid.01 --size=10000 --ondisk hdb

raid /mydata --level=1 --device=md0 raid.00 raid.01" \>
  > /tmp/user_partition_info
</pre>
```
Yeah, But …

- What if I don’t know the name of the disk devices?
  - Or, what if I have a mix of disk devices in my cluster (e.g., hda, sda, cciss, etc.)?

- What if I want to apply different partitioning schemes to different nodes?
Let’s Write a Program!

◆ We’ll write a program to populate:
  
  /tmp/user_partition_info

◆ The program will have access to:
  
  ➔ The node’s name
  ➔ The node’s membership
  ➔ The names of the discovered disks
import rocks_partition

membership = '<var name="Node_Membership'/>'
nodename = '<var name="Node_Hostname"/>'

def doDisk(file, disk):
    file.write('clearpart --all --initlabel --drives=%s
' % disk)
    file.write('part / --size=6000 --fstype=ext3 --ondisk=%s
' % disk)
    file.write('part /var --size=2000 --fstype=ext3 --ondisk=%s
' % disk)
    file.write('part swap --size=2000 --ondisk=%s
' % disk)
    file.write('part /mydata --size=1 --grow --fstype=ext3 --ondisk=%s
' % disk)

# main
p = rocks_partition.RocksPartition()
disks = p.getDisks()

if len(disks) == 1:
    file = open('/tmp/user_partition_info', 'w')
    doDisk(file, disks[0])
    file.close()
One or Two Disk Partitioning

```python
# main
#
# p = rocks_partition.RocksPartition()
disks = p.getDisks()

file = open('/tmp/user_partition_info', 'w')

if len(disks) == 2:
doRaid(file, disks)
elif len(disks) == 1:
doDisk(file, disks[0])

file.close()
```
def doRaid(file, disks):
    file.write('clearpart --all --initlabel --drives=%s

    % ', ','.join(disks))

    raidparts = []

    for disk in disks:
        if disk == disks[0]:
            part = 'part / --size=6000 --fstype=ext3 ' + \
              ' --ondisk=%s
' % disk
            file.write(part)

            part = 'part /var --size=2000 --fstype=ext3 ' + \
              ' --ondisk=%s
' % disk
            file.write(part)

            part = 'part raid.%s --size=5000 --ondisk=%s
' % (disk, disk)
            file.write(part)

            raidparts.append('raid.%s' % disk)

    raid = 'raid /bigdisk --fstype=ext3 --device=md0 --level=1 %s

    % ', '.join(raidparts)
    file.write(raid)
<pre>
import rocks_partition

membership = '<var name="Node_Membership"/>

nodename = '<var name="Node_Hostname"/>

def doRaid(file, disks):
    file.write('clearpart --all --initlabel --drives=%s\n
    % ', ','.join(disks))

    raidparts = []

    for disk in disks:
        if disk == disks[0]:
            part = 'part / --size=6000 --fstype=ext3 ' + \
                '--ondisk=%s\n' % disk
            file.write(part)

            part = 'part /var --size=2000 --fstype=ext3 ' + \
                '--ondisk=%s\n' % disk
            file.write(part)

            part = 'part raid.%s --size=5000 --ondisk=%s\n' % (disk, disk)
            file.write(part)

            raidparts.append('raid.%s' % disk)

    raid = 'raid /bigdisk --fstype=ext3 --device=md0 --level=1 %s\n
    % ', ','.join(raidparts)
    file.write(raid)

def doDisk(file, disk):
    file.write('clearpart --all --initlabel --drives=%s\n
    % disk)

    file.write('part / --size=6000 --fstype=ext3 --ondisk=%s\n
    % disk)

    file.write('part /var --size=2000 --fstype=ext3 --ondisk=%s\n
    % disk)

    file.write('part swap --size=2000 --ondisk=%s\n
    % disk)

    file.write('part /mydata --size=1 --grow --fstype=ext3 --ondisk=%s\n
    % disk)

    # main
    #
    p = rocks_partition.RocksPartition()
    disks = p.getDisks()

    file = open('/tmp/user_partition_info', 'w')
    if len(disks) == 2:
        doRaid(file, disks)
    elif len(disks) == 1:
        doDisk(file, disks[0])

    file.close()
</pre>
Partitioning Based on Node Name and Disk Count

```python
# main
#
p = rocks_partition.RocksPartition()
disks = p.getDisks()

if nodename in ['compute-0-0']:
    file = open('/tmp/user_partition_info', 'w')
    if len(disks) == 2:
        doRaid(file, disks)
    elif len(disks) == 1:
        doDisk(file, disks[0])

file.close()
```
Force Rocks Default Partitioning

<pre>
 echo "rocks force-default" > /tmp/user_partition_info
</pre>
Force Manual Partitioning

<pre>
echo "rocks manual" > /tmp/user_partition_info
</pre>

◆ This will cause the RedHat partitioning screen to appear on an installing node
You can interact with this screen by executing on the frontend:

```
# rocks-console compute-0-0
```