Workshop Goals and Rocks 5.0 Enhancements

Why are we Here?

ROCKS™

The Goals of the Workshop

- We're here for <u>you</u>
- Tell you where we've been and where we're going
- Have you tell us what's important for you to get out of the workshop
- Shape the Agenda to better meet the needs of those who are here
  - How much tutorial vs. discussion ?
  - How much detail on the internals ?
  - How much on Futures?

Rules

1. You must ask questions and participate
2. See Rule #1

Introductions

- Rocks Team from UCSD
- Around the Room
  1. Name
  2. Institution/Company, Dept. Role
  3. #1 thing you want to take away from the workshop

Agenda Overview

- There is a logical rationale for the way things are laid out
- 5 Rocks Workshop Sessions
  - +1 Rocks-focused general-interest session (Wed 10:30 – 12:00, does not collide with Workshop, does collide with IntroTutorial Session #4)
- An Intro/Tutorial thread runs throughout the conference to cover Rocks, Globus, SGE. 3 of theses sessions are Rocks

Agenda Specifics

- #1 (Tu 10:30 – 12:00)
- #2 (Tu 4:30 – 6:00)
  - Xen VMs, Virtual Clusters, Programmatic Partitioning
- #3 (We 8:30 – 10:00)
  - How to develop your own rolls. Some Solaris Updates
- (#3.5) (We 10:30 – 12:00)
  - Innovative uses of Rocks [Room 208]
- #4 (We 1:30 – 3:00)
  - Extending Functionality through the Rocks Command Line
- #5 (We 4:30 – 6:00)
  - Talk back to the Rocks Developers
- Thursday – Grill the Gurus – Bring us your problems. This is a free consulting session

Details #2

- "The internals of Xen support in Rocks will be presented and dissected in detail. A preliminary roadmap for enhanced support for completely virtualized clusters (frontends and slave nodes) will be given. New for Rocks 5.0 is the ability to fully program how a node partitions its local hard drives so that any partitioning policy can be implemented. Methods, techniques and examples of partitioning schemes will be presented."

- **What do people want to You want to see here ?**

Details #3

- "Rolls are the primary mechanism for customizing Rocks installations while enabling reproducibility to any number of clusters. Rolls can be commercial or open-source. ClusterCorp has produced several rolls and will describe their techniques and issues. Techniques for how Linux-based rolls are built and tested at UCSD. An introduction to the needed Rocks changes to support Solaris and Rocks-on-Solaris will be presented."

- **Others?**

Details #4

- The Rocks command line is the way rolls extend the command structure for Rocks. The Rocks Viz Roll will be used as key example of roll-based extension to support tiled-display clusters.   The Solaris command set (currently under development) will be illustrative of how Rocks commands can work across different architectures.
- ??

Straw mode of operation

- At the end of each workshop session, we’ll spend the last 5 minutes asking you for what else is still missing.

What I do when the Rocks Workshop isn't running ?

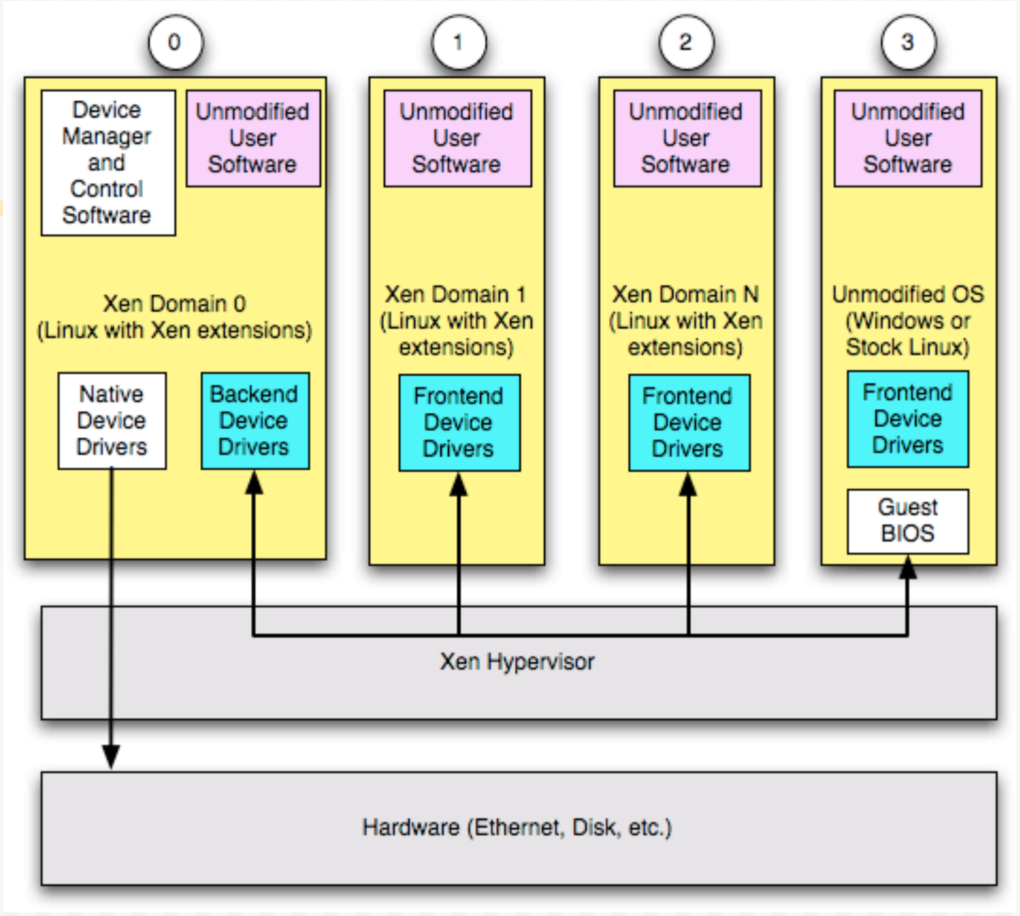
- Visit the other Workshops (Globus, SGE)
- Talk to us "offline"
- Work with each other
  - How about the "updates" group?
- Do e-mail

Rocks 5.0 Enhancements

Xen Support

- Xen Roll
  - Optional roll
    - Still can build good ol' physical clusters
- New appliance: “VM Container”
  - Physical machine that houses VMs
  - Not schedulable
    - That is, queueing systems won't see VM Containers

What is Xen

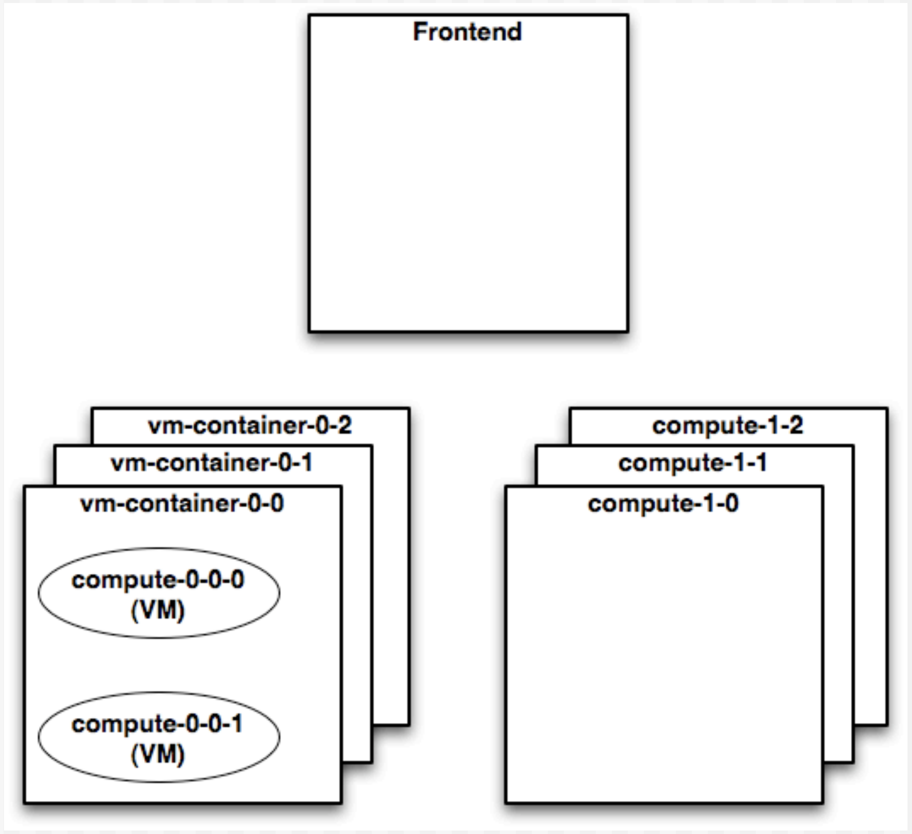


- Guest traps and exceptions are passed to and handled by hypervisor

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



Xen Support

- Rocks command line expanded to manage Xen VMs

```
rocks add host vm
rocks create host vm
rocks start host vm
```

Key VM Functions

- “add”
  - Add a new VM to the cluster
- “create”
  - Install a VM
- “start”
  - Boot a VM

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"
```

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

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 to 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
  - Then “restores” the VM

Other “Internal” Commands

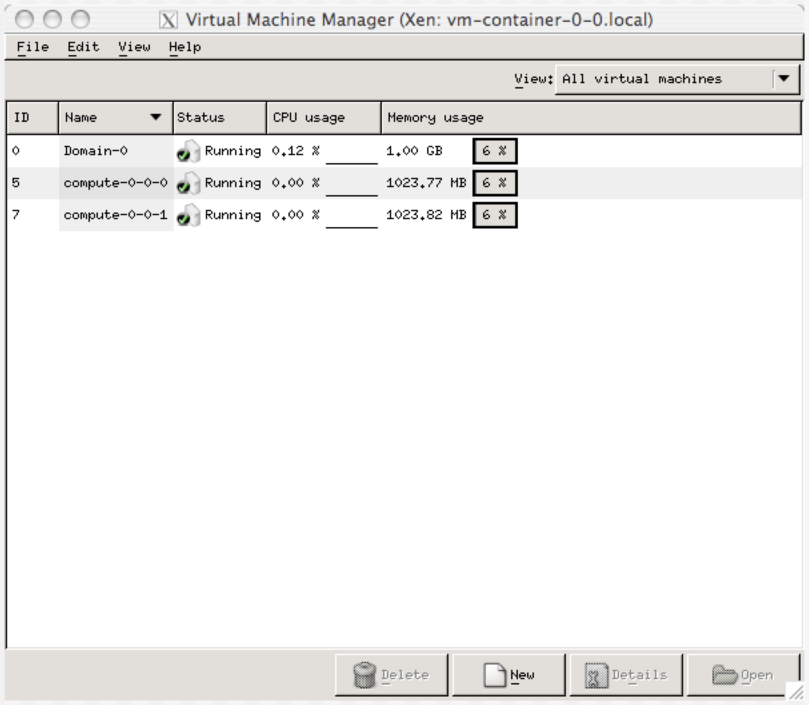
- “dump”
  - Used on the restore roll to capture VM configuration
- “report”
  - Called by “rocks create host vm” and “rocks start host vm” to create Xen VM configuration files
- “remove”
  - Called by “rocks remove host” to remove the VM specific info for a host

Xen Debugging Tool

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

```
# virt-manager
```

Virt-manager

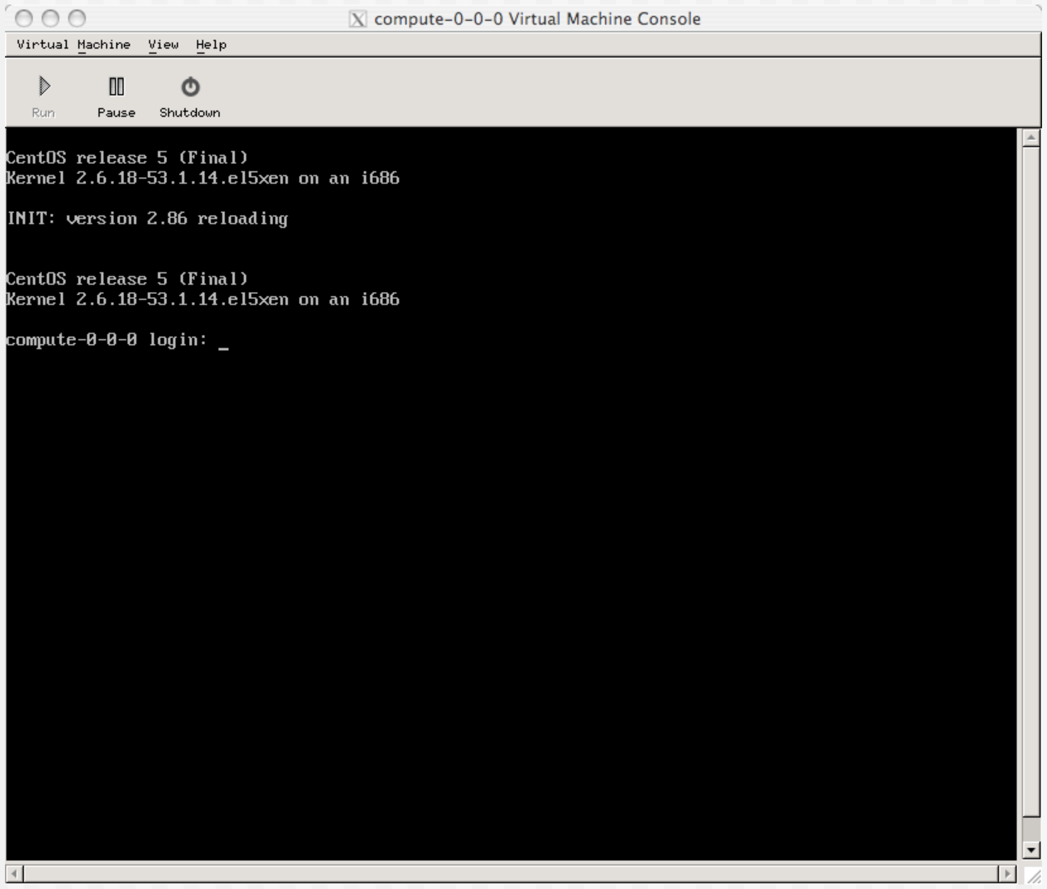


Virtual Machine Manager (Xen: vm-container-0-0.local)

ID	Name	Status	CPU usage	Memory usage
0	Domain-0	Running	0.12 %	1.00 GB 6 %
5	compute-0-0-0	Running	0.00 %	1023.77 MB 6 %
7	compute-0-0-1	Running	0.00 %	1023.82 MB 6 %

- Double click on ‘compute-0-0-0’ to bring up console

Virt-manager



```
CentOS release 5 (Final)
Kernel 2.6.18-53.1.14.el5xen on an i686

INIT: version 2.86 reloading


CentOS release 5 (Final)
Kernel 2.6.18-53.1.14.el5xen on an i686

compute-0-0-0 login: _
```

Programmatic 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

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

Programmatic Partitioning

- Can write a program to express the exact partitioning that you desire for your heterogeneous cluster
  - We provide helper functions to get names of all discovered disks

Programmatic Partitioning

- One XML node file can output partitioning info for node that have:
  - Different number of disks
  - Disks with different names (e.g., hda, sda)
  - Different appliance types (e.g., NAS, compute, tile)
  - Different names (e.g., compute-0-0, compute-0-1)

```
<pre arg="--interpreter /opt/rocks/bin/python">

import rocks_partition

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

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()

if len(disks) == 1:
        file = open('/tmp/user_partition_info', 'w')
        doDisk(file, disks[0])
        file.close()
</pre>
```

Flashing BIOS with PXE

Flash BIOS via PXE

1. Download BIOS flash program
2. “make build”
3. “make install”
4. Rocks command line to set PXE behavior to ‘pxeflash’
5. Boot compute node
6. Rocks command line to set PXE behavior to ‘os’
7. Flash compute node
8. Reboot compute node

Setting Kernel Boot Parameters

Installation Boot Parameters

- Example, we’ll add “ucsd=rocks” to compute-0-0 boot parameters
- The boot action of compute nodes is controlled by the Rocks command line:

```
# rocks list host pxeboot
HOST         ACTION
olympic:     -------
compute-0-0: os
```

  - “os” = boot the OS off local disk
  - “install” = on next boot, install

Installation Boot Parameters

- List all boot actions:

```
# rocks list host pxeaction compute-0-0
ACTION           COMMAND              ARGS
install          kernel vmlinuz       append ks initrd=initrd.img ramdisk_size=150000
                                             lang= devfs=nomount pxe kssendmac selinux=0 noipv6
install headless kernel vmlinuz       append ks initrd=initrd.img ramdisk_size=150000
                                             lang= devfs=nomount pxe kssendmac selinux=0 noipv6 headless vnc
memtest          kernel memtest       ----------------------------------------------------------------------
os               localboot 0          ----------------------------------------------------------------------
pxeflash         kernel memdisk bigraw append initrd=pxeflash.img keeppxe
rescue           kernel vmlinuz       append ks initrd=initrd.img ramdisk_size=150000
                                             lang= devfs=nomount pxe kssendmac selinux=0 noipv6 rescue
```

Installation Boot Parameters

- Change boot action:

```
# rocks set host pxeboot compute-0-0 action="install"
```

- Check our work

```
# rocks list host pxeboot
HOST          ACTION
olympic:      --------
compute-0-0: install
```

Add a New PXE Action

- Add global action:

```
# rocks add host pxeaction action="install ucsd" command="kernel vmlinuz” \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

- Check our work

```
# rocks list host pxeaction compute-0-0
ACTION           COMMAND               ARGS
install          kernel vmlinuz        append ks initrd=initrd.img ramdisk_size=150000
                                       lang= devfs=nomount pxe kssendmac selinux=0 noipv6
install headless kernel vmlinuz        append ks initrd=initrd.img ramdisk_size=150000
                                       lang= devfs=nomount pxe kssendmac selinux=0 noipv6 headless vnc
install ucsd     kernel vmlinuz        append ks initrd=initrd.img ramdisk_size=150000
                                       lang= devfs=nomount pxe kssendmac selinux=0 noipv6 ucsd=rocks
memtest          kernel memtest        ---------------------------------------------------------------
os               localboot 0           ---------------------------------------------------------------
pxeflash         kernel memdisk bigraw append initrd=pxeflash.img keeppxe
rescue           kernel vmlinuz        append ks initrd=initrd.img ramdisk_size=150000
                                       lang= devfs=nomount pxe kssendmac selinux=0 noipv6 rescue
```

Add a New PXE Action

- Add compute-0-0 only action:

```
# rocks add host pxeaction compute-0-0 action="install ucsd" \
  command="kernel vmlinuz” \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

- Override global action

```
# rocks add host pxeaction compute-0-0 action="install" \
  command="kernel vmlinuz” \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

Installation Boot Parameters

- Change boot action:

```
# rocks set host pxeboot compute-0-0 action="install ucsd"
```

- Check our work

```
# rocks list host pxeboot
HOST         ACTION
olympic:     ------------
compute-0-0: install ucsd
```

Running Boot Parameters

- Get the current boot flags

```
# rocks report host bootflags
rocks-168: dom0_mem=1024M
compute-0-0: dom0_mem=1024M
```

- Add a boot flag

```
# rocks set host bootflags compute-0-0 \
  flags="dom0_mem=1024M ucsd=rocks”
```

- Check

```
# rocks report host bootflags
rocks-168: dom0_mem=1024M
compute-0-0: dom0_mem=1024M ucsd=rocks
```

Running Boot Parameters

- Reinstall to apply boot flags
- After the node installs, check

```
# cat /proc/cmdline
ro root=LABEL=/ dom0_mem=1024M ucsd=rocks
```

Rocks Futures

Our Heads are in the Cloud(s)

Other Stuff

- Xen
  - Support for hardware virtualization
  - Frontend in domU
  - Support for multiple virtual clusters on one physical cluster
- Move /export/home/install to /export/rocks/install
  - Rocks distro will no longer be on an NFS share