Implementing Scalable Disk-less Clusters using the Network File System (NFS)

James H. Laros III
Lee H. Ward
Sandia National Labs
LACSI Symposium, 28 Oct 2003
http://www.cs.sandia.gov/cit
jhlaros@sandia.gov
Outline

• Introduction
• Related Work
• Hardware
• Methodology
  – Requirements and Issues
  – Efficiency
  – Bootable Hierarchy
  – NFS
• Results
  – 128 Node Test System
  – 1024 Node Test System
• Conclusions
Introduction

• Some “general” context
  – Target Large Clusters (up to 10,000 nodes)
  – Applications requiring High Performance Computers (HPC’s)
• Why go Disk-less??
  – Cost of Device (200*1000=200,000)
  – Cost of Power
  – Cost of Cooling, energy in energy out
  – Cost of Replacement/Downtime –
    (1,000,000 MTBF/1000 Nodes)/24 hours = 41.67 hours
  – Cost associated with elevated temp on other components?
  – Software distribution!!!!!
    • Even for disk-full systems can avoid root fs distribution
    • No matter how good software distribution methods get it is still better to just say no!!
  – Red/Black switching
• NFS is one way to implement even large disk-less clusters
Related Work

• Didn’t find any efforts using NFS for Disk-less Clusters
  – At least of any size…
• Linux Networx/Los Alamos Labs
• OSCAR
• Dell
• Booting A disk-less node, or a FEW, using NFS is not a new concept!
Hardware

• When this work began, hardware limitations were a much more significant consideration.
• How many nodes could one “server” support?
  – For older systems 32:1 worked nicely
    • System capability
    • How many nodes fit in a rack!
      – Probably could have done 64:1 but wouldn’t fit in rack
  – Recent tests prove 256:1 not a problem
    • Still hard to fit 256 in a rack!
    • Maybe in a blade rack
    • Other issues remain, like command distribution
Methodology
(Requirements and Issues)

• Some example requirements
  – BOOTP/DHCP request processing
  – TFTP of kernel image
  – Mount requests
  – Storage of NFS-root image
• Been done for years for small numbers of clients
• Will it work for 1000’s of nodes?
  – Thousands of BOOTP, TFTP and mount requests?
  – Thousands of copies of the NFS-root filesystem?
• How can we approach these issues in an efficient manner?
Methodology (Efficiency)

• Hardware infrastructure increasingly important as cluster size grows
  – Offload single Administration node responsibilities

• How?
  – Leaders initialize first
    • Disk-less nodes themselves!!
    • all requests serviced by administration node directly.
  – Compute nodes initialize
    • BOOTP requests serviced at the leader level
      – Leader running DHCP daemon
    • TFTP request for kernel also serviced at the leader level
      – Kernel cached after first request and serviced from memory
    • Mount requests again serviced at the leader level
      – NFS file-system re-exported by leader
      – More benefits from caching
Methodology
(Bootable Hierarchy)

- Describes the layout of the NFS image used to support the cluster.
- Some files common for every node in a cluster
- Some files common for groups of nodes in a cluster
- Some files unique to a single node
- Bootable Hierarchy leverages sharing whenever possible while preserving node level granularity when needed.
Methodology
(Bootable Hierarchy)
cont.

• First install stock Linux™ distribution in an alternate root path (image)
  – Easiest to just install everything
    • No impact on performance of cluster, only what gets used gets cached
    • Image the same regardless of cluster size
• Bootable Hierarchy built on top of image
  – Single command
  – Multiple images supported
• Files specific to a group of nodes or a single node that differ from image will override the image
• Our intermediate grouping based on the “role” that the node serves in the cluster
• These “roles” are instantiated in .proto directories
  – compute.proto and leader.proto
  – These roles selected for the potential of file based commonality
Bootable Hierarchy Logical Layout
Methodology
(Bootable Hierarchy)
cont.

• Some specific examples…

```
alternate-root-path/image/etc/rc.d/rc7.d/network ↘
alternate-root-path/compute.proto/etc/rc.d/rc7.d/network ↘
alternate-root-path/t-0/node.n-1.t-0/etc/rc.d/rc7.d/network ↘

alternate-root-path/image/etc/rc.d/rc7.d
alternate-root/path/leader.proto/etc/rc.d/rc7.d/dhcpd ↘
alternate-root/path/t-0/node.n-0.t-0/etc/rc.d/rc7.d/dhcpd ↘

alternate-root-path/image/etc/sysconfig/network-scripts
alternate-root-path/compute.proto/etc/sysconfig/network-scripts
alternate-root-path/t-0/node.n-1.t-0/etc/sysconfig/network-scripts/ifcfg-eth0
```
Methodology
(Bootable Hierarchy)
cont.

• This process heavily leverages linking
  – Each link is a file
  – Every file requires an inode
  – Lots of inodes!!
  – Can minimize by getting rid of things you don’t need
    • Like 6000 entries in /dev!!!
• Not limited to this example, a hierarchy for any specific need can be implemented.
  – We have found only /etc, /var, and /dev need to be dealt with
Methodology (NFS)

- Kernel space NFS daemon doesn’t currently allow re-exporting
  - Some silly reason like security
- User space implementation used on leader nodes to allow re-exporting
  - Provides the caching effect that we leverage
- Important to note that caching is only beneficial for read requests!
  - Luckily most of what is done is reads
  - Most writes that do take place are unnecessary
    - Distributions are brain dead for this purpose
  - Most can be avoided
    - Not writing PID files
    - mount --n
- Using kernel space daemon on admin node seems to perform better
Results

• 128 Node system
  – Alpha XP1000’s
    • Better memory bandwidth than DS10’s
    • Better leaders
  – Single processor XP1000 Admin node
  – Switches

• 1024 Node system
  – Alpha DS10’s
  – Dual processor DS20 Admin node
  – Hubs
  – Part of larger production system
    • Limited time to perform these tests
Results
cont.

- Initialization defined as time from executing boot command until all nodes are at the login prompt
- Init1 times represent initialization with “freshly” booted leaders
- Init2 times are 2nd initializations of same nodes using same leaders
  - Most common case in production
  - Leaders seldom rebooted
- Init1 vs Init2 evidence of caching effect
- Initialization chosen since it is the most stressful test
- Other observations?
  - I was left with many questions
- Command graph demonstrates efficiency of common operations
Initialization times 128 Nodes

![Graph showing initialization times for 128 nodes.](image-url)
Initialization times 1024 Nodes

![Graph showing time vs. number of nodes]

- init 1
- init 2
Command Execution Times

![Graph showing the execution time of ls command over different numbers of nodes.

- The x-axis represents the number of nodes (0, 32, 64, 128, 256, 512, 768, 1024).
- The y-axis represents the time in seconds (0 to 15).
- The graph shows a linear increase in execution time as the number of nodes increases.
- The line with solid squares indicates the execution time for ls command.

Source: Sandia National Laboratories]
Conclusions

• Based on these results this methodology provides a sound foundation.
• More importantly based on our production experience of the past 3 years!!
• In practice even larger systems efficient and stable (1873 nodes)
• No problems with stability vs. disk-full systems.
• The problems we have encountered have were not related to the methodology
  – Network driver issues biggest headache
System Usage

<table>
<thead>
<tr>
<th>Month</th>
<th>System Usage %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>89.7</td>
</tr>
<tr>
<td>Feb</td>
<td>94.5</td>
</tr>
<tr>
<td>Mar</td>
<td>70.5</td>
</tr>
<tr>
<td>Apr</td>
<td>94.1</td>
</tr>
<tr>
<td>May</td>
<td>88.4</td>
</tr>
<tr>
<td>Jun</td>
<td>89.3</td>
</tr>
<tr>
<td>Jul</td>
<td>89.4</td>
</tr>
<tr>
<td>Aug</td>
<td>92.1</td>
</tr>
</tbody>
</table>
Future Work

• Future work somewhat dependent on hardware.
• We tend to feel that you can scale to the level that you have proven you can scale to
• That said based on what we have seen this methodology is not yet approaching its limits
  – Faster more powerful processors
    • Have demonstrated 256:1 ratio
  – Deepen hardware hierarchy
  – Leverage new features in Linux™
    • Dev filesystem
    • Tmpfs
• Other ways to approach disk-less clusters
  – Union FS
  – Light-weight approach to standard Linux™ kernel
  – Lustre?
• Possible Correction, first efforts were in 1997 not 1987
http://www.cs.sandia.gov/cit

jhlaros@sandia.gov