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

James H. Laros III[*], Lee H. Ward[†]
Sandia National Labs[‡]

30th October 2003

Abstract

This paper describes a methodology for implementing disk-less[1] cluster systems using the Network File System (NFS) that scales to thousands of nodes. This method has been successfully deployed and is currently in use on several production systems at Sandia National Labs. This paper will outline our methodology and implementation, discuss hardware and software considerations in detail and present cluster configurations with performance numbers for various management operations like booting.

Keywords: Clusters, NFS, Disk-less, CIT

1 Introduction

Disk-less clusters are well suited for a large range of cluster applications. Clusters built for scientific applications, particularly parallel applications, have little need for local disks. Once the nodes are booted an application can be distributed to the targeted nodes in a variety of different ways, none of which necessitate a local disk. These types of applications use available system resources like CPU and resident memory, swapping is avoided. File input and output is typically accomplished to some sort of high-speed file-system with a global namespace, preferably a parallel file-system. For a cluster built to support these types of applications, a disk with the root file-system local to the node would be used only for booting the node, after which it would become a liability for reasons we will discuss later in this section.

Some cluster systems are constructed for more general-purpose use. These clusters can also benefit from a disk-less approach at least in the area of system infrastructure.

[*]James H. Laros III, Sandia National Labs - Corresponding author, jhlaros@sandia.gov

[†]Lee H. Ward, Sandia National Labs, lee@sandia.gov

[‡]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000. Sandia National Laboratories, PO Box 5800 Albuquerque, NM, 87185

[1]In this paper the term disk-less cluster is used to refer to a cluster comprised of nodes without disks, primarily referring to disks used for systems initialization and/or a root file-system.

Access to the root file-system for many general-purpose clusters is largely limited to reads. If this is the case, our approach can still provide much of the same benefit. Even if a local disk is required by the application for file output, as is customary in clusters that support database applications, taking the disk-less approach can still mitigate the difficulties of software distribution of the root file-system.

In a disk-full[2] system, the root file-system must be distributed to each node to support system initialization. In the best-case scenario, each node will have identical software. In practice this is usually not the case. While there have been many efforts to develop efficient software distribution mechanisms for this purpose, we contend that it is more efficient to avoid this issue entirely. Disk-less clusters have no disks, therefore, there are no software distribution issues for individual nodes.

The additional costs associated with disk-full clusters can also be a major consideration especially for cluster systems with node counts in the thousands. Consider a modest disk price of $200 per disk. For a 1000 node system the additional cost added to the cluster is $200,000. When dealing with large cluster systems, disk failure rates are also magnified by node count. A Mean Time Between Failure (MTBF) of 1,000,000 hours for a single disk sounds encouraging. Unfortunately, for our 1000 node cluster example this would translate to replacing a disk every 1000 hours or approximately 41 days[3]. Additional costs are also incurred in disk-full systems to power the disks and cool the additional heat generated by the disk. In short, it can be safely stated that disk-less clusters are less expensive to purchase and operate considering these hardware and environmental costs.

We will present our approach to implementing a disk-less cluster system using NFS, which can accommodate a wide range of clusters built for different applications. We will outline the configuration of the underlying hardware that accommodates this approach, the disk-less hierarchy that supports the software requirements on a node-by-node basis, and configuration aspects of using NFS and the Linux™ [1] operating system for this implementation. We will also present some timing results for common activities like cluster initialization.

2 Related Work

This paper will frequently refer to the Cluster Integration Toolkit (CIToolkit or CIT)[2]. The CIToolkit was developed at Sandia Labs to support disk-less clusters that could scale to thousands of nodes. Many other requirements influenced the design of the CIToolkit. Details regarding the software architecture and design of the CIToolkit can be found in "*An Extensible, Portable, Scalable Cluster Management Software Architecture*"[3]. Details on the implementation of the CIToolkit can be found in "*The Cluster Integration Toolkit*"[4]. In short, the CIToolkit provides the foundation and utilities necessary to accomplish most of what will later be discussed in the consistent and repeatable fashion essential for a production environment.

[2] In this paper the term disk-full cluster describes a cluster that has a disk per node which is, at the writing of this paper, the most common cluster configuration.

[3] (1000000MTBF per node/1000nodes)/24hours = 41.66666667

It is important to note that in 1997 when Sandia Labs implemented their first disk-less clusters there were no efforts underway for utilities to support even small disk-less clusters let alone clusters that would scale to thousands of nodes. Booting disk-less nodes using NFS is certainly not a new concept. However, using NFS to initialize and subsequently support clusters in excess of a thousand nodes had never been done. Sandia's efforts which included the CIT project were started to accomplish this and other goals. The result has been proven scalability to 1873 nodes[4], with no indication that continued scalability to many thousands of nodes is not possible.

Recently, other projects seem to be considering the advantages of disk-less clusters. Based on our current evaluation of industry progress in this area, Linux Networx[5] in cooperation with Los Alamos National Labs[6] has been the most aggressive with a 1024 node system[7]. While this system does not use NFS, it still represents a much welcome advance in the disk-less cluster field. The OSCAR[8] project has begun addressing this issue, and Dell[9] has published a paper[10] recognizing many of the benefits of this approach.

3 Hardware

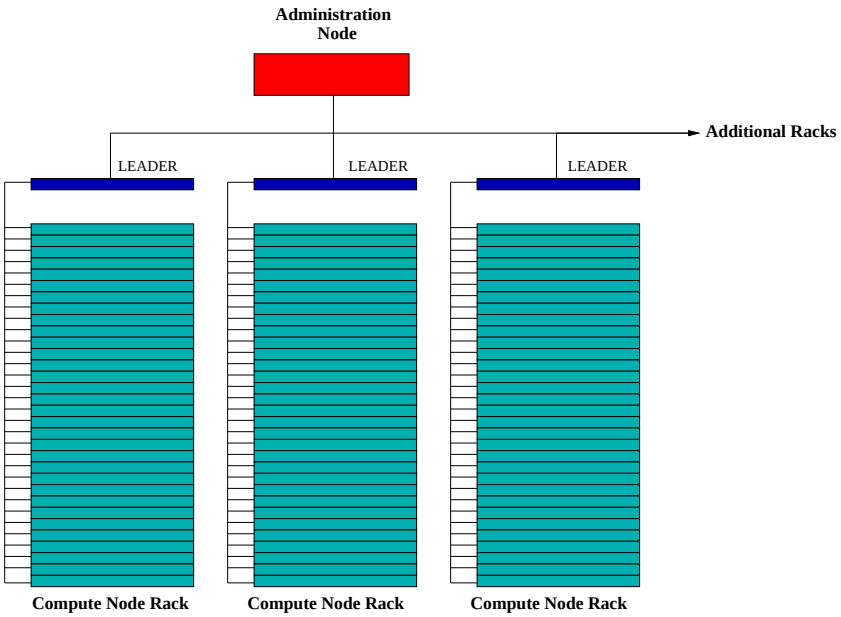

Figure 1: Hardware Hierarchy

[4] 1 administration node supporting 57 leaders, 1792 compute nodes and 24 service and I/O nodes.

To describe our approach to this problem, it is beneficial to understand the underlying hardware configuration. Figure 1 provides a logical depiction of the hardware components and network topology that will be important for this discussion.

The hardware configuration used in our approach is hierarchical in nature. For the purposes of this paper the management network of the cluster will be described. The term management network describes the network, in this case Ethernet, used to support cluster infrastructure operations like booting the nodes of the cluster. At the top of the hierarchy is the administration node. This node contains the sole disk in the cluster system used for purposes of node initialization, export of the nfs-root filesystem, and subsequent cluster management. As shown in Figure 1, below the administration node are some number of secondary nodes we refer to as "leaders".

In practice, the administration node is itself a leader to the leader nodes. Think of a leader as a node that has some responsibility for other nodes, mainly in a support sense. Below the leader nodes are the nodes that make up the bulk of the cluster. These nodes satisfy various roles (the purpose the node serves in the overall system) predominantly serving as compute nodes (nodes where applications are executed). The ratio of leaders to administration node, and nodes to leaders is dependent on the type of system used to support these roles. For the purpose of this paper we will use a ratio of 32 nodes per leader. This ratio is used on both of the example configurations detailed in this paper. The nodes on each level of the hierarchy share a network with the node above it. For example, the administrative node shares a common network with all of the leader nodes. A leader node and the nodes under it also share a common network. While our methodology does not require a specific hardware configuration, this hardware hierarchy nicely accommodates the distribution of work necessary to facilitate our methodology on large scale systems.

4 Methodology

4.1 Requirements and Issues

Let us consider the requirements as they occur as a disk-less node boots, in a generic sense[5]. First, the node requests information via a protocol like Bootstrap Protocol (bootp)[11]. The Dynamic Host Configuration Protocol (dhcp)[12] daemon is commonly present on a server to respond to the bootp requests. After obtaining information like Internet Protocol (IP)[13] address, and file name of the kernel to load, the node will make a file transfer request. This request is commonly serviced by the Trivial File Transfer Protocol (tftp)[14] daemon on the server. Eventually the node will want to mount its NFS root, which will be serviced by the mount daemon on the server node.

So in a generic sense, supporting a single node or a few nodes is not hard for a single server node. Supporting 1000 nodes becomes a problem. Using our example above, a server would first have to service 1000 bootp requests. In the next phase, the server node would have to deliver kernels via tftp to the same 1000 nodes. Finally, each of the 1000 nodes would request its root file-system via NFS. Even if this could be accomplished with a single server node, the initialization of the nodes would have

[5]This method was described for simplicity. Other initialization methods are easily supported.

to be staggered such that the time to boot all 1000 nodes from start to finish would be prohibitive.

Another consideration is the amount of disk space required to support an NFS root file-system image on the server. For a single node this is again a straightforward requirement. For two or more nodes the disk requirement increases as a multiple of the number of nodes to be supported. Consider the disk space requirements for 1000 separate root file-system images!

Taking a closer look at the NFS root requirements imposed by clusters with large node counts reveals more opportunities and issues. For example, based on the role the node will serve, much of the root file-system can potentially be shared. A command like "ls" may very well be the same for every node, independent of role, in a cluster. In contrast, a configuration file like ifcfg-eth0, which on many Linux systems is used to configure the IP address of an interface on a node, may need to be different on a per-node basis. The actual requirements of a cluster can range anywhere between these two extremes.

In short, the challenges mentioned can be grouped and addressed as follows:

- How do we overcome the load issues imposed by the various disk-less node requirements on the server node for large node counts?
- How do we satisfy the root file-system requirements of the cluster, on a node-by-node basis, in hopefully a dynamic fashion, without providing a copy of the root file-system for each node in the cluster?

Our methodology answers these questions in a way that makes it appropriate for clusters that can satisfy a wide range of applications.

4.2 Efficiency

To achieve the efficiency necessary to satisfy the requirements for large disk-less cluster systems, an appropriate hardware infrastructure as described in Section 3 is essential. Many of the previously described issues involve large amounts of requests made over a range of protocols to a single server. By distributing the server responsibilities, overloading the server node can be avoided.

The hardware configuration describes the administration node at the top of the hierarchy (Figure 1). If this administration node were the sole server node responsible for responding to the massive number of requests imposed by a large disk-less cluster, it would quickly become overloaded. By utilizing the underlying hardware configuration to distribute the responsibility of responding to these requests to the leaders, we greatly increase efficiency allowing a disk-less cluster to scale to much greater numbers than otherwise possible.

The leaders are disk-less clients of the administration node. A 1024 node cluster system requires 32 leaders based on the aforementioned 32:1 ratio. The administration node can easily handle the requests involved in a disk-less boot of the 32 leaders. Once booted, the leader nodes will be responsible for responding to the requests of the compute nodes below them. It is important to note that since the leader nodes are disk-less

themselves, the information required to respond to the requests of the compute nodes originates from the administration node.

Consider the bootp requests that will be made by the compute nodes during initialization. These requests will be serviced by the dhcp daemon on the leader. When the leader initialized its dhcp daemon, it read in the dhcpd.conf file from the NFS mounted filesystem exported from the administration node. Once initialized, this information is resident on the leader node and no further requests are made to the administration node to service the bootp compute node requests, effectively offloading this responsibility from administration node.

One of the pieces of information obtained from the leader by the compute nodes in the bootp exchange is typically the filename of the kernel to load. When the compute nodes request this file (kernel) the leader will service these requests through tftp daemons spawned in response to each request. When the first request for a kernel file is made the leader will not have this file cached and require the administration node to serve this file via NFS (along with the tftp daemon executable). Assuming that the compute nodes all require the same kernel file, which is often the case, this first request will be the only request that imposes on the administration node. All other requests by the initializing compute nodes will be serviced through the NFS cache resident in the leaders memory. These requests have to be made in a timely manner before the cache becomes stale, but normally we are trying to initialize the nodes as fast as possible.

The final series of requests made will be mount requests by the compute node, which again are largely serviced out of the leaders cache, depending on uniqueness of the request. By offloading these requests from the administration node to the leader node, we allow for the efficient initialization of large disk-less clusters since these operations can be done in parallel; each leader servicing a group of compute nodes, largely out of their cache, simultaneously. We will refer to this as the caching effect throughout this paper.

4.3 Bootable Hierarchy

The term Bootable Hierarchy is used to describe the layout of the NFS image used to support the disk-less cluster. The process of creating the bootable hierarchy is automated using the CIToolkit. For the purposes of this paper we will focus on the structure of the bootable hierarchy rather than focusing on the particulars of CIT.

As mentioned, some files required by the disk-less cluster may be the same for every node in the cluster, which provides an opportunity for file sharing. Other files may be unique on a per-node basis. Our methodology allows a bootable hierarchy to be created that can exploit sharing at any level of commonality, and preserve a node level granularity when necessary, on a cluster-by-cluster basis.

The first step in establishing the bootable hierarchy is to install what we call the image. The image is the stock Linux distribution installed in an alternate root path on the administration node. Rather than waste time selecting what files or packages will be installed and resolving conflicts based on this selection, we have taken the approach of installing all available software. This approach has no impact on the performance of the cluster (only files that are used are cached, see Section 4.2). While the image will be the same size regardless of the number of nodes in the cluster, the inode count increases as

the node count increases. The process of creating the bootable hierarchy, described in more detail below, heavily leverages linking. Since each link requires an inode, large clusters can impose a heavy inode requirement. This is an important consideration when preparing the filesystem that will hold the image and bootable hierarchy.

We currently use the RedHat[15] distribution and the Redhat Package Manager (RPM) to install the image (there is no dependency on this distribution). To reduce the inode requirement for large clusters we have paired down the bloated /dev directory in the standard release to remove thousands of unnecessary entries.

The image is the master copy of the root file-system for the cluster. As the hierarchy is built on top of the image, files that are desired to be specific to a group of nodes, or a particular node will override the image. This allows for the wide range of file sharing or granularity down to the node level mentioned. For example, commands that exist in directories like /bin are most likely common to all nodes in the cluster regardless of the role they serve in the cluster. Configuration files such as files in /etc, that may determine specific characteristics like the IP address of a specific node's interface, are unique to each node. This leaves a wide area available for medial levels of grouping. Our implementation provides for intermediate grouping based on the role of the node. Other groupings based on any scheme desired are easily implemented. On our production systems we have implemented three role abstractions, but for the purposes of this paper we will only describe leader and compute roles. These groupings or abstractions are instantiated in prototype directories described in the following paragraphs.

In our test systems the nodes will serve the role of leader or compute. As mentioned, the leader nodes are nodes that provide infrastructure services to other nodes. The compute nodes make up the bulk of the cluster. It should be noted that the reason these two roles were selected is the likely file based commonality of nodes that serve these roles. If there is utility in grouping nodes by other roles, those roles could be used as the basis for grouping in addition to, or instead of compute and leader. Since it is likely that leader nodes, as a group, will share some common file requirements that may be different from the image, we construct a leader.proto directory to house these common differences. For the same purpose, we construct a compute.proto directory. Files that are specific to an individual node regardless of its role in the system will be placed in the node specific directory at the bottom of the hierarchy. Figure 2 illustrates a logical layout of the bootable hierarchy and the three levels of abstraction we will discuss.

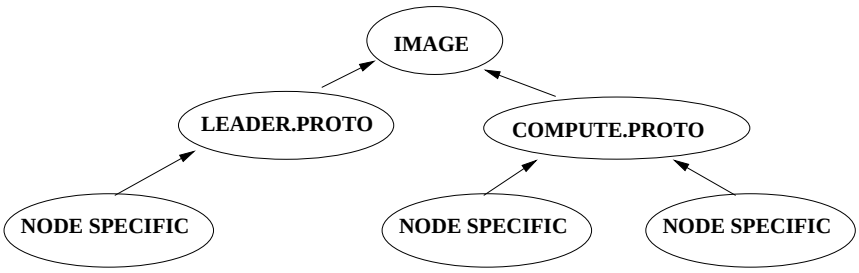

Figure 2: Bootable Hierarchy logical layout

As mentioned, the image contains the stock Linux distribution. The standard initialization files for this particular distribution reside in a series of numbered rc directories, which correspond to the init run-level. To simplify things for our implementation we add our own run-level to avoid what would be tedious modification of the stock image. All stock run-time directories are left as installed and just not used. In this newly created run-level directory we insert startup scripts that have the potential to be common for the specific cluster this image will support. For example, all nodes in the cluster will require some network interface upon initialization. Since this is the case, it is most efficient to put the actual network startup script in the image itself and link back to it from all nodes in the cluster. Remember, it is always possible to override this choice in either the proto or node specific directories. Figure 3 illustrates this concept. The node specific directory for a node called *node.n-1.t-0* has a startup file called *network* in the rc directory for run-level 7 (*etc/rc.d/rc7.d*). It first links back to the compute.proto level *network* startup file. Since the *network* startup file is a common requirement for the entire cluster, the file links all the way back to the image before it finds the file (the red color illustrates where the actual *network* startup file exists in the hierarchy). We currently incorporate this multi-level linking only in the /etc path since this is where the most potential exists for leveraging the concepts discussed. If the need arose to make other files that exist in other directories common to a group of nodes, or unique to a specific node, the bootable hierarchy could easily be modified to reflect this need. Note the *t-0* component in the node specific path, which designates that this node is in rack t-0. We have found this grouping useful especially with large cluster installations.

Figure 3: Network startup file links back to image

The first level of abstraction above the image is the group level which contains the proto directories for each group implemented. Files that are specific to a group

of nodes and absent for nodes belonging to other groups can be located in a group specific proto directory. The leaders, for example, require extra services to be started upon initialization like the dhcp daemon discussed previously. Since this startup script is unique to leaders it is located in the *leader.proto* directory. Figure 4 illustrates a specific example. Notice there is no *dhcpd* startup file in the *image* path. This is because *dhcpd* startup is not typically common for all nodes. Since *dhcpd* startup is typical for leaders we place a *dhcpd* startup script in the *leader.proto* path. Nodes with a role of leader, like *node.n-0.t-1*, link back to the *leader.proto* path, and by default start the dhcp service. Nodes with a role of compute would not find a *dhcpd* startup file either in the *compute.proto* path or the *image* path and therefore would not start this service.

Figure 4: dhcpd startup file links back to leader.proto level

The next level of abstraction in the bootable hierarchy and the final in our implementation is the node specific level. This level is provided to harbor node specific files. An area exists for each node in the cluster regardless of its role. This area may or may not contain anything specific to the node depending on the requirements imposed on that node in the cluster. In our implementation we find few files are required to be specific to an individual node, but this concept allows for a node to have a completely individualized root file-system if necessary. Figure 5 illustrates a file that is commonly unique to a specific node. The *ifcfg-eth0* file in our example configuration contains information to initialize the 0th interface for the node. This file has node specific information like the IP address for the node. Since this information is specific to a node, in this case *node.n-1.t-0*, it is placed in the node specific directory for that node. Notice that this file does not exist in the *image* or the *compute.proto* path. This is because when the hierarchy was created it was anticipated that this file would be specific to the node level in the hierarchy. Even if this file did exist in either the *image* or proto levels in the hierarchy the effective file would still be the node specific file, due to the precedence rules of the bootable hierarchy.

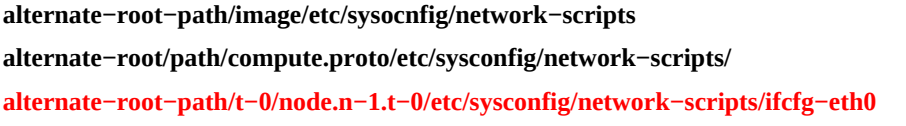

Figure 5: ifcfg-eth0 file specific to node

To avoid unnecessary linking which also reduces the heavy inode requirement, we

utilize this hierarchical linking only when it is beneficial. Entire directories like /bin, /lib and /usr, which so far have not required us to over-ride in any way, are mounted read-only and are shared by all the nodes in the cluster.

Other directories that are a challenge to share like /dev and /var are simply located in the individual node specific directories. Since avoiding writes is critical to the success of this approach, a sparse /var skeleton is all that is required. Individual /dev directories can either contain a full complement of device entries, or can be paired down to reflect the individual requirements of a node. Section 7 discusses options like the device file-system which can be easily incorporated into this methodology and would remove the necessity of individual /dev directories.

The hierarchy we have described is constructed automatically with tools provided by the CIToolkit based on the role concept and many other configurable attributes. These concepts allow us to do other creative things like supporting alternate images. If there is a requirement for nodes in the cluster to use a different Linux version or distribution, an image for both can be established on the administrative node. Additional group proto directories could then be established linking back to this new image, and finally specific nodes, existing in the same cluster, could utilize a completely different version or distribution.

Once the bootable hierarchy is created, it is ready to be used by the cluster. The following section will describe how this is accomplished.

4.4 NFS

The filesystem that holds the bootable hierarchy is exported via NFS to the cluster. Upon initialization, the leaders request the path of their nfs-root filesystem. This information is contained in the configuration file for dhcp daemon. This file is automatically generated with a utility in the CIToolkit on a per-node basis, populated with the path in the bootable hierarchy specific to this node, among other information. The leader mounts its nfs-root file-system. Other mounts are requested based on system, group or node specific initialization files. The leader also mounts the entire bootable hierarchy to a mount point, which will be made available to the nodes beneath it.

Since the kernel space NFS daemon does not currently allow re-exporting of filesystems, which is a key requirement of this methodology, a user space implementation of NFS [16] is used on the leader nodes to allow the bootable hierarchy to be re-exported to the nodes it supports. The leader supports all requests whether for bootp information, kernel image, or individual compute node NFS mount requests from its NFS mounted filesystems. As described in Section 4.2, many of these requests are handled from the leaders cache. This caching effect greatly enhances the scalability of the cluster.

A single administration node could easily be overcome by NFS traffic. Unfortunately, leveraging the caching effect only works with read requests. While read requests for the same file can be serviced from the leaders cache after the initial request is made, write requests made by the nodes must eventually be serviced by the administration node. Since this is the case, it is important to avoid unnecessary writes.

As it turns out, most writes are not necessary and can be avoided. Simple things like writing out a process id (PID) file when multiplied by a thousand nodes can contribute to overloading the administration node. Writes like this can easily be avoided with little

effect on the cluster. Another method of avoiding writes is using the mount command with the "-n" option to avoid writing to the mtab file, which is generally unnecessary for compute nodes. If needed, information about what is mounted on the node can be obtained from the /proc filesystem. Another technique we employ is mounting filesystems with the noattr option. This avoids attribute update actions, which also impose unnecessary load on the administration node. These are a few examples of ways to improve scalability. While these activities may appear trivial, they have a large impact on the efficiency of the cluster.

5 Results

The results we will present are from two separate clusters. Both clusters are very similar in architecture. Since it is not our intention to contrast the results but to report results on at least two different systems, these differences in architecture are not significant but are included here for completeness. While results are reported for only two systems we have successfully implemented all of our production and test clusters, numbering greater than a dozen, with this methodology.

Our 128 node test cluster[6] is comprised entirely of Alpha XP1000[17] nodes. The XP1000 nodes, also known as Monets, have a superior memory bandwidth when compared to the Alpha DS10[17] nodes, which make up the 1024[7] node cluster. Due to superior memory performance, the Monets generally make better performing leader nodes since leaders depend heavily on memory speed to perform their function. The 128 node test cluster also employs network switches instead of hubs, which are used on the 1024 node cluster. The last difference worth mentioning is the hardware used for the administration node. Our experience shows that even for our largest systems, special hardware is not required to support this role and remain within our production requirements. With that said, it is obvious that since the role of the administration node is so significant a better performing administration node can be beneficial. The administration node for the 128 node cluster is a single processor XP1000, while the results reported for our 1024 node cluster were obtained using a dual processor DS20 system.

We should additionally point out that the largest system that we have proven this methodology on, in production mode, is the 1873 node system mentioned in Section 2. In the following sections we will show timing results for system initialization and command execution. System initialization is shown due to the stressful nature of the operation. Our experience has shown that initialization of the cluster imposes, by far, the greatest load on the administration node and therefore is the best single test of our methodology. Command distribution is shown to illustrate the efficiency of general maintenance operations using this methodology.

5.1 128 node test system

The graph in Figure 6 illustrates initialization times obtained on our 128 node test system. Initialization is defined as the time from boot command execution until ev-

[6] 128 compute nodes + 4 leaders = 132 total nodes (32:1 compute:leader)

[7] 1024 compute nodes + 32 leader nodes = 1056 total nodes (32:1 leader:compute)

ery node reaches the login prompt. The times reported do not include initialization of leader nodes. The “init 1” times are the initialization times for each data point using freshly initialized leaders. The “init 2” times are a subsequent initialization of the same nodes, using the same leaders. Note that the “init 2” times are the most representative of normal initialization times experienced in a production mode. In production environments leaders are seldom re-booted. Figure 6 and Figure 7 illustrate the caching effect discussed in Section 4.2. As the number of nodes increase the caching effect becomes more prominent evident by a reduction in initialization times. Not shown in the provided graphs is the beneficial effect of caching on the administration node. Not only are initialization times decreased but the load on the administration node during initialization is decreased since most requests are intercepted at the leader level. In addition, the administration node also caches information and duplicate requests are more efficiently served directly from memory.

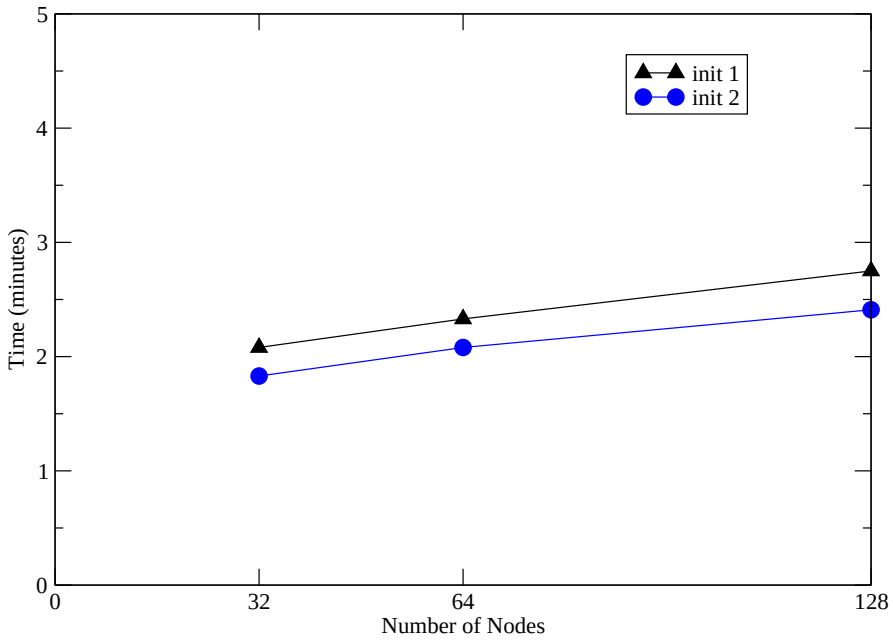

Figure 6: Initialization times for 128 node test system

5.2 1024 node test system

The graph in Figure 7 illustrates initialization times obtained on our 1024 node test system. Test parameters for this figure were identical to those stated for the 128 node test with the addition of delays inserted into the boot command execution. These delays are intentionally inserted to avoid potential thrashing on the administration node. A delay of 20 seconds after every 10 (32 node) racks was chosen when signs of thrashing

appeared while initializing 12 racks simultaneously. This delay adds 20 seconds of additional time for the 512 node, 40 seconds for the 768 node and 60 seconds for the 1024 node initialization. While this process slows the command distribution it speeds up the overall initialization by forcing requests to be more orderly, and at a rate more easily handled by the administration node.

As shown in Figure 7 our methodology provides for very efficient initializations even for the 1024 node configuration. Thoughts on how this methodology could possibly scale to many thousands of nodes will be discussed in Section 7.

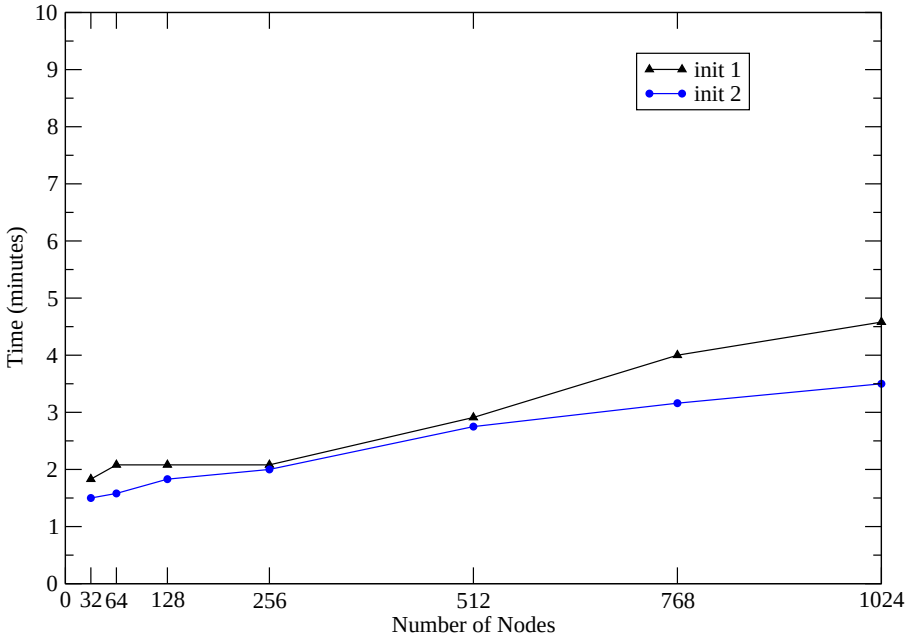

Figure 7: Initialization times for 1024 node test system

Figure 8 charts times for command distribution over the same 1024 node test system. This is provided to illustrate that the disk-less configuration efficiently supports execution of common utilities used to maintain the initialized cluster. While much of the efficiency of the command distribution can be attributed to the tool provided by the CIToolkit, this chart illustrates that like system initialization, command execution also benefits from the caching effects. The chart illustrates excellent scaling from 32 nodes at 8.4 seconds to 1024 nodes at 10.8 seconds. The command used for the recorded numbers is a simple ls command with its output redirected to /dev/null to remove possible delays incurred by buffering across the network to our terminal. In reality the numbers without redirecting to /dev/null only differed by fractions of a second and were not meaningful to include in our chart. The times shown include waiting for completion of the command on all nodes.

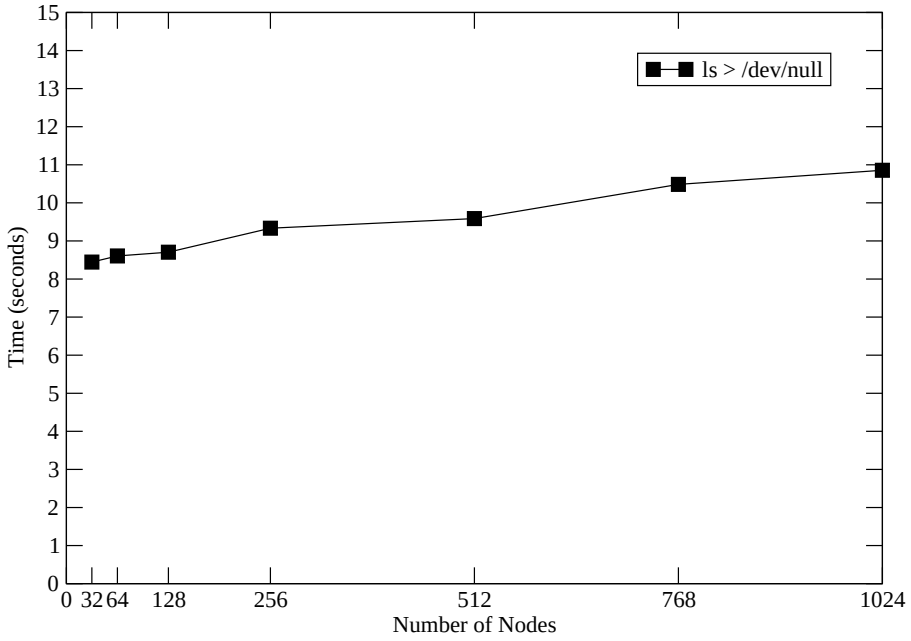

Figure 8: Command execution times

6 Conclusions

The most important goal of this work was to provide for a disk-less methodology appropriate for production use. While the results presented for initialization times and command execution are important and illustrate key aspects and capabilities necessary to accomplish this goal, they do not illustrate the most important aspect, the resulting stability of the system. Over the past three years this methodology has been used to support all of Sandia's production clusters. While individual nodes are lost due to various hardware and software failures, this technique has provided a stable production environment. Even our largest cluster systems consistently remain stable for the two week period between system preventative maintenance (PM) periods. Longer periods of time have routinely been experienced but it is typical to re-initialize the system during PM periods. Figure 9 illustrates our system utilization, by month, since the beginning of 2003 to the present. These percentages reflect system utilization of available node hours. The available node hours include our bi-weekly PM times which are used for hardware and system diagnostics, which do not reflect as systems usage.

It can be stated that our methodology shows no weakness in the area of stability when compared to a disk-full cluster used for similar purposes.

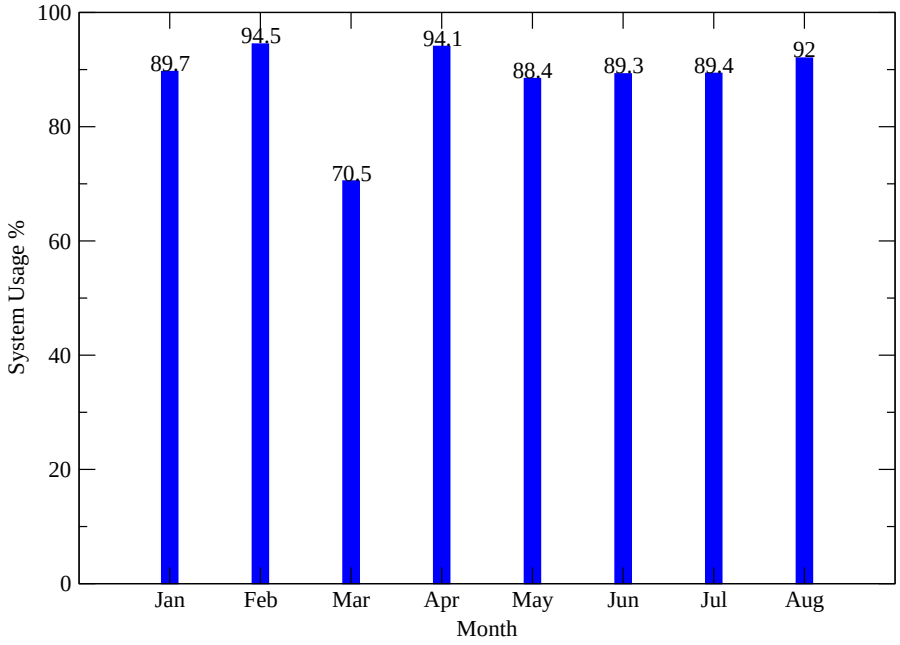

Figure 9: System Usage[18]

7 Future Work

Unfortunately, future work in this area is dependent on the availability of hardware resources. It has been our experience that concepts that scale to 128, 256 or even larger numbers of nodes won't necessarily continue to scale. It also happens that concepts scale beyond initial expectations. Based on our demonstrated success we feel confident that this concept could easily scale to between two and three thousand nodes using the same underlying hardware infrastructure. By deepening[8] the hardware hierarchy it is likely that this methodology could support node counts in excess of 5000. Other optimizations like increasing the performance of the administration node or the exported filesystem could also positively effect the scalability of the system. Leveraging new features available in the Linux kernel like the device filesystem can also enhance scalability and reduce the dependency on the administration node. Features like this are easy to add to our existing infrastructure.

While our methodology heavily leverages NFS, there are other ways to implement disk-less clusters. We would like to explore the applicability of Union File Systems (Union FS)[19] to this problem. Union FS could provide a private view of a read only shared file-system without the extensive linking our current implementation requires. It is also likely that a Union FS implementation would be much more tolerant to write

[8]Deepening or adding another level of leader nodes which will further distribute the server responsibility.

requests than our current methodology.

Light weight kernels have always been popular for high end systems at Sandia Labs. Our recent work has shown that a light weight approach using the standard Linux kernel source can be leveraged to eliminate the dependence on NFS entirely. The scalability of this approach should have no bounds.

Finally, the impact that future filesystem efforts like Lustre[20] may have on this or other methods of implementing disk-less clusters are yet to be demonstrated but have great promise.

8 Acknowledgments

We would like to thank the Scalable Systems Integration and Scalable Computing Systems departments at Sandia Labs for all of their contributions to this work. Nathan Dauchy and other members of the CIToolkit project have been instrumental in enhancing and stabilizing these concepts for our production platforms. The members of the Cplant^{TM}[21] project have also provided valuable feedback and technical contributions.

We would also like to thank the Scientific Computing department at Sandia Labs who has been the patient recipient of our product.

References

[1] Linux Trademark held by Linux Torvalds

[2] Cluster Integration Toolkit, developed at Sandia National Labs, http://www.cs.sandia.gov/cit

[3] An Extensible, Portable, Scalable Cluster Management Software Architecture, James H. Laros III, Lee Ward, Nathan W. Dauchy, Ron Brightwell, Trammell Hudson, Ruth Klundt, Proceedings of the 2002 IEEE International Conference on Cluster Computing, 23-26 Sept. 2002.

[4] The Cluster Integration Toolkit - An Extensible, Portable, Scalable Cluster Management Software Implementation. (PDF, PS), James H. Laros III, Lee Ward, Nathan W. Dauchy, James Vasak, Ruth Klundt, Glen Laguna, Marcus Epperson, Jon R. Stearley, Proceedings of the 1st Cluster World Conference and Expo, 23-26 June 2003.

[5] Linux Networx, http://www.linuxnetworx.com

[6] Los Alamos National Labs, http://www.lanl.gov

[7] http://www.lnxi.com/news/lanl_info.php

[8] OSCAR project, http://oscar.sourceforge.net

[9] Dell Computer, http://www.dell.com

[10] White paper regarding disk-less clusters
http://www.dell.com/us/en/biz/topics/power_ps4q02-guler.htm

[11] BOOTP - RFC 951 http://www.faqs.org/rfcs/rfc951.html

[12] DHCP - RFC 2131 http://www.faqs.org/rfcs/rfc2131.html

[13] Interoperation between bootp and dhcp - RFC 1534
http://www.faqs.org/rfcs/rfc1534.html

[14] TFTP - RFC 1350 http://www.faqs.org/rfcs/rfc1350.html

[15] http://www.redhat.com

[16] Originally developed by Mark Shand,
http://mops.uci.agh.edu.pl/doc/nfs-server-2.2beta31/

[17] RISC processor developed by Digital Equipment Corporation (DEC), DEC was purchased by Compaq in 1998, Compaq was subsequently purchased by Hewlett Packard, http://www.hp.com

[18] Production data provided by Sophia Corwell, Scientific Computing, Sandia Labs.

[19] http://www.usenix.org/publications/library/proceedings/neworl/full_papers/mckusick.a

[20] Lustre, http://www.clusterfs.com/lustre.html

[21] Cplant was developed at Sandia National Labs, http://www.cs.sandia.gov/cplant