Improving Cluster Management with Scalable Filesystems

Adam Boggs*, Jason Cope*, Sean McCreary*†, Michael Oberg*†, Henry M. Tufo*†, Theron Voran*, and Matthew Woitaszek*†

* University of Colorado, Boulder
† National Center for Atmospheric Research
michael.oberg@colorado.edu

Abstract. Reducing the complexity of the hardware and software components of Linux cluster systems can significantly improve management infrastructure scalability. Moving parts, in particular hard drives, generate excess heat and have the highest failure rates among cluster node components. The use of diskless nodes simplifies deployment and management, improves overall system reliability, and reduces operational costs. Previous diskless node implementations have relied on a central server exporting node images using a high-level protocol such as NFS or have employed virtual disks and a block protocol such as iSCSI to remotely store the root filesystem. We present a mechanism to provide the root filesystems of diskless computation nodes using the Lustre high-performance cluster file system. In addition to eliminating the downtime caused by disk failures, this architecture allows for highly scalable I/O performance that can be free from the single point of failure of a central fileserver. We evaluate our management architecture using a small cluster of diskless computation nodes and extrapolate from our results the ability to provide the manageability, scalability, performance and reliability required by current and future cluster designs.

1 Introduction

When designing a high-performance computational cluster, many factors limit system scalability. Shared networks, centralized servers, computational node design, and other hardware constraints place limits on potential system performance. Cluster scalability is also limited by management infrastructure decisions affecting node installation, maintenance, synchronization, monitoring, logging, and security. These infrastructure decisions unnecessarily constrain the number of manageable computational nodes.

Using diskless nodes enhances scalability in several ways. Removing the local disk improves node reliability and reduces heat output and electrical consumption. Instead of relying on disks placed in each node, the system relies on a root filesystem provided by a smaller number of more reliable servers. However, diskless nodes also provide greater scalability for node installations and software maintenance. By using a common root filesystem, diskless nodes simplify node administration and enhance security by keeping portions of the filesystem immutable. By safely storing the compute node root filesystem on centralized servers, backup and restore operations become much simpler and multiple versions of the filesystem can be easily maintained. This also supports more efficient cluster upgrades and configuration rollbacks, and allows quicker testing and rollout of new configurations.

Previous diskless node implementations have used techniques such as block-based iSCSI storage targets and read-only NFS root filesystems. While these solutions provide similar benefits, they are limited by the scalability of the underlying storage system and its servers. Our work leverages the high-performance, reliable, and scalable storage that is already present on many clusters. In this paper, we describe the creation and management of a cluster of diskless computation nodes using a root filesystem provided by a Lustre storage cluster. To do this, we made minimal modifications to the SuSE SLES 9 distribution so that it can boot from a Lustre root filesystem in a manner similar to the existing iSCSI or NFS root filesystem support. Our performance measurements on a small cluster using a Lustre-based root filesystem show that while an individual node’s interactive response time appears to increase, the performance of scheduled computational tasks is not adversely affected. Most applications using only one of two system CPUs experience a slight performance improvement when running on nodes with a Lustre-based root filesystem, but applications using both CPUs experience a marginal, but tolerable, performance decrease over disk-based systems.

2 Background and Motivation

Previous work with diskless nodes has concentrated on techniques ranging from low-level remote block devices to network-provided operating system kernels with NFS root filesystems. At the lowest level, the iSCSI protocol can be used to provide a diskless node access to a physical volume through a network. iSCSI has been shown to perform competitively with fibre channel (FC) in local area network configurations [1]. The Peabody project at the University of Colorado used iSCSI to provide remote root filesystems for a student computer laboratory with diskless clients [16]. However, iSCSI utilizes block-based disk access, so an iSCSI target cannot be easily shared among multiple nodes. Instead, a separate iSCSI target must be used to provide each system’s remote root filesystem, requiring multiple complete copies of the image. While remote boot using iSCSI moves the root filesystems from the nodes to a central server, it does not reduce management complexity. A more desirable approach would be a single read-only filesystem image shared among many nodes that supports exceptions for files that differ between nodes.

The pre-boot execution environment (PXE) standard provides a useful method for booting diskless nodes. At boot time, pxeboot retrieves IP and boot configuration information from a DHCP server, retrieves the kernel and initial RAM disk (initrd) via the Trivial File Transfer Protocol (TFTP), and boots the system. PXE is widely supported on Intel-based Ethernet hardware, but new implementations of Infiniband such as Voltaire IBNetBoot provide pxeboot capabilities from FC, IP attached storage, or IP attached servers [21] over Infiniband. After the kernel has been loaded, the system initialization script (initrd init) mounts the root filesystem. More advanced approaches, such as the LinuxBIOS project [14], eliminate the PXE boot loader and place an entire Linux kernel in BIOS. Once this kernel has been loaded, the node may utilize a root filesystem on the local disk or any networked root filesystem.

In traditional configurations, a commodity Linux cluster boots off a single filesystem image provided by a single NFS server [10]. As clusters increase in size, the load on this single server can become excessive and effectively limits the number of nodes which can be supported. Larger installations may use a cluster of NFS servers to extend the scalability of this design. One of the best examples is the Mare Nostrum supercomputer in Barcelona that uses 41 NFS servers for remote filesystem images [15]. Each NFS server supports 56 hosts located in the same cabinet. This configuration introduces additional management overhead, and Mare Nostrum uses IBM's prototype Diskless Image Management (DIM) product to configure the boot environment.

Modern Linux cluster designs often include a high-performance parallel filesystem, such as PVFS2 [17], GPFS [6], TerraGrid [18], or Lustre [13], to provide a scalable I/O subsystem. These filesystems provide tremendous performance gains over NFS when run on commodity Ethernet hardware [8], but they are typically only used for job data and temporary storage space. Parallel filesystems have not been used to provide root filesystems in the past due to perceived reliability problems and low community acceptance. Parallel filesystems have generally been used for volatile temporary storage, and they typically rely on a complex stack of kernel modules and software that is not part of the operating system distribution itself. For example, Linux supports booting from iSCSI and NFS remote filesystems, but the out-of-box network boot feature does not support third-party proprietary parallel filesystems. Most parallel filesystems are now sufficiently mature as to provide reliable and highly available storage without requiring customized kernel installations. This makes their use for root filesystems feasible. Our work extends the typical diskless node cluster design by exploiting these already-present highly scalable parallel I/O subsystems to provide computational node root filesystems.

3 Design

The objective of our design is to provide the root filesystem for a cluster of computational nodes from a Lustre parallel filesystem in a manner that reduces administrative complexity. Specifically, we aim to:

1. Use Lustre to provide a read-only copy of the entire root filesystem.
2. Provide a flexible mechanism that allows read-write access to a minimum set of files and directories to be maintained individually for each node (files in /etc, /var, etc.).
3. Constrain operating system changes to the smallest set possible to facilitate ongoing upgrades and possible future community acceptance.

Reducing the amount of node-specific files greatly reduces the cluster's administrative complexity. For example, all software installations performed on the global image are immediately available on all nodes in the cluster, and changes to /etc configuration files immediately take effect. With a properly implemented global read-only root filesystem, the configuration management components of IBM CSM [7] and xCAT [3] are no longer necessary. To minimize the management complexity, as much of the filesystem as possible must be shared. Only files that must be unique to each node should be stored individually.

There are several Linux techniques that can override portions of a shared read-only filesystem to provide node-specific files: standard directory mounting, an overlay filesystem, and symbolic linking. These are the same methods used by other systems built upon read-only root filesystems, such as NFS remote root and Live Linux CD-ROMs such as Knoppix [9]. Using standard mounts, certain node-specific directories can simply be mounted from a read-write volume provided by a ramdisk or a network filesystem [2]. This provides a very coarse-grained method of providing machine-specific files but increases the amount of management overhead required to keep node images consistent. Alternatively, overlay filesystems can be used to mount a read-write filesystem on top of a read-only filesystem. The files on the read-only filesystem remain visible, but the modifications are redirected to the read-write overlay filesystem. While this provides a generic solution for node specific files, it relies on third-party overlay filesystem software. In our design, we chose to use symbolic linking because it supports our design without modifying the boot environment and provides the most flexibility for a wide user community.

In our solution, symbolic links are employed to redirect certain files to node-specific locations. During the boot sequence, init first mounts the root filesystem read-only from a shared subdirectory of the Lustre cluster, and then mounts a directory named /local read-write from a node-specific subdirectory of the Lustre cluster. Portions of the read-only filesystem have been previously symbolically linked to /local to provide node-specific files. For example, the symbolic linking approach can be used to make every node have completely different /etc, /var, and /tmp directories:

```
/etc                -> /local/etc
/tmp                -> /local/tmp
/var                -> /local/var
```

This coarse approach actually complicates maintenance because the RPM package database is maintained under /var. In this configuration, installing a RPM on the global read-only image installs the software on all of the nodes but leaves the RPM databases inconsistent. Thus, to actually reduce complexity, the granularity of node-specific files must be reduced. We have determined that full functionality for each computational node can be provided by making the following files and directories node-specific:

```
/etc/ld.so.cache    -> /local/etc/ld.so.cache
/etc/localtime      -> /local/etc/localtime
/etc/mtab           -> /proc/mounts
/etc/resolv.conf    -> /local/etc/resolv.conf
/tmp/               -> /local/tmp
```

```
/var/adm/SRC          -> /local/var/adm/SRC
/var/lib/dhcp         -> /local/var/lib/dhcp
/var/lib/dhcpcd       -> /local/var/lib/dhcpcd
/var/lib/nfs          -> /local/var/lib/nfs
/var/lib/ntp          -> /local/var/ntp
/var/lock             -> /local/var/lock
/var/log              -> /local/var/log
/var/run              -> /local/var/run
/var/spool            -> /local/var/spool
/var/tmp              -> /local/var/tmp
```

Maintaining the node-specific configuration is straightforward because the shared read-only filesystem is stored on a Lustre filesystem and changes can be made by an administrator from any Lustre client. Files and directories may be made node-specific by changing a single directory entry in the read-only root filesystem hierarchy, and the changes are immediately visible to all nodes. This approach to node-specific files reduces the complexity of software installation and maintenance. The implementation, presented in the next section, describes how we modified the Linux boot procedure to support remote Lustre root filesystems.

4 Implementation

Booting a diskless node from a parallel filesystem presents several difficulties not present in an NFS solution. While NFS only requires a simple association of IP addresses and server mount points, a parallel filesystem like GPFS or Lustre requires significantly more local client configuration. In addition, parallel filesystems rely on binaries and kernel modules that are typically not part of the standard Linux distribution or available early in the boot process.

We address these challenges with several techniques. First, we modified the existing initrd init boot process to support a Lustre root filesystem through the “root” kernel command line option. Second, we use the Trivial File Transfer Protocol (TFTP) to provide initial boot images, a system configuration file, and supplemental kernel modules. Finally, we made a small number of changes to other operating system files, such as the daemon control scripts in /etc/init.d, to reduce the number of errors reported during system startup.

We were particularly careful to implement our Lustre root filesystem solution in a manner consistent with the existing scheme for remote root filesystems in SuSE Linux. By limiting the scope of the modifications, we reduce the work required to deploy new operating system updates, facilitate support in other Linux distributions, and increase the possibility of future community acceptance. *We did not directly change the initial ramdisk or the init script; we changed the tools that generate them instead.* To integrate Lustre root filesystem support into SuSE Linux, we only changed 20 lines in mkinird, 10 lines in kinit.sh, and added a new 120-line script to process the Lustre-specific boot operation. The entire system is integrated using higher-level software and can produce an initrd capable of booting from a Lustre root filesystem generated from a currently running system.

4.1 Boot Configuration

The SuSE SLES9 distribution provides NFS root filesystem functionality using command-line options to the init script contained in the initrd image. For example, to boot using a remote root NFS filesystem, an option of the format "root=nfs:/*server*/*mount*" may be used. For our implementation, we preserved the existing interface and added supplemental functionality to the init script itself to support booting from a Lustre filesystem.

Starting Lustre requires that a series of kernel modules be loaded in the correct order. However, we experienced an 8MB limit on CHRP (kernel and initrd package) file size for our PPC970 systems that prevented all of the Lustre modules from being packaged in a single initrd. To work around this limitation, we introduced a feature to download and install kernel modules using TFTP from a remote server. Thus, filesystem kernel modules need not be included in initrd proper, but may be downloaded and installed separately during the boot process. The module source and node-specific configuration information are encapsulated in a single configuration file that is passed to init using the "root" option using the following format:

```
root=lustre:tftp-server-address:configuration-filename
```

On PXE-based systems, node-specific configurations can be provided using init options passed by a DHCP server. However, because the PPC970 systems use CHRP images with static init command lines, TFTP server keyword substitution can be used to provide equivalent functionality. For example, every node can request a specific configuration file based on its IP address using a boot parameter similar to the following:

```
root=lustre:172.16.100.1:config-CLIENTIP.sh
```

The configuration file contains three parameters: a TFTP server source for supplemental kernel modules, a Lustre path to the read-only filesystem root, and a Lustre path to the read-write node-specific filesystem. For example:

```
MODULE_SOURCE=tftp://172.16.100.1/lustre-modules
ROOT_DIR=store01://lustre-mds/client/rootboot/rootfs-ppc970
NODE_DIR=store01:/lustre-mds/client/rootboot/node001
```

In this example, supplemental modules will be downloaded from the lustre-modules subdirectory on the TFTP 172.16.100.1. The Lustre path /rootboot/rootfs-ppc970 will become the read-only root directory for the node, and the path /rootboot/node001 will become the read-write node-specific directory /local. After the kernel modules have been downloaded and installed, the root filesystem can be mounted.

4.2 Filesystem Mounting

The Linux startup sequence mounts the root filesystem using a two stage process. Immediately after the kernel loads, the initrd is mounted as the root directory and /init is executed. This init script parses the command line options and chooses the correct device to mount as /root, which later becomes the root filesystem for the running operating system. For example, a hard drive, an iSCSI device, or a NFS volume may be mounted as /root. We added a small segment of code that can mount a Lustre volume as /root. After /root has been prepared, the initrd script executes /run_init, which pivots the root directory so that /root becomes the real root and then executes the SysV init (/etc/init) contained on that filesystem.

One obstacle to mounting a Lustre volume as the root filesystem is that, at the present time, the Lustre zero-conf mount implementation only supports mounting the full Lustre filesystem. It does not support mounting subdirectories such as:

```
mount –t lustre mds-server://mds-name/share-name/path /mount
```

We overcome this limitation by first mounting the full Lustre filesystem and then using a bind mount to access the correct subdirectory as suggested by CFS [12]. We actually mount the Lustre filesystem twice: once read-only for the root filesystem, and once read-write for the /local node-specific directory using a command sequence similar to the following but using variables from the configuration file:

```
mount –o ro –t lustre store01://lustre-mds/client /lustre-ro
mount –o bind /lustre-ro/rootboot/rootfs-ppc970 /root

mount –o rw –t lustre store01://lustre-mds/client /lustre-rw
mount –o bind /lustre-rw/rootboot/blade001 /root/local
```

Thus, when the initrd init completes, the /root filesystem has been mounted and completely configured. The subsequent call to run_init executes the chroot operation to make /root become /, and then the SysV init continues to load the operating system from the Lustre root filesystem. When combined with the node-specific filesystem configuration as described in the design section, nodes successfully boot from the Lustre filesystem. However, several additional changes to auxiliary files are required to boot without warnings.

4.3 Auxiliary File Changes

In order to make the nodes using the Lustre root filesystem boot without warnings and to preserve the read-only nature of the root filesystem, several changes to additional operating system files are required. First, to prevent the filesystem from being mounted read-write during boot, the /etc/init.d/boot.rootfsck script must be changed so that the filesystem is not remounted read-write. Second, to prevent the NFS service from failing, the /etc/init.d/nfs script must also be edited. After mounting the NFS volumes, the script runs ldconfig to update the linking environment:

```
ldconfig -X 2>/dev/null
```

This fails because ldconfig attempts to create a temporary cache file, ld.so.cache~, in the read-only /etc directory. The real ld.so.cache file is symbolically linked to /local/etc/ld.so.cache, so updating the ldconfig line to use that directory directly corrects the problem:

```
ldconfig -C /local/etc/ld.so.cache -X 2>/dev/null
```

The process of finding and eliminating startup errors and warnings such as these has been an iterative process. At this point, our implementation successfully boots nodes with no errors or warnings, and the nodes provide the same full functionality as the nodes booting from local hard disks.

4.4 Automated initrd Creation

To reduce the complexity of creating Lustre-compatible root filesystem initrd images, we modified the SuSE mkinitrd script. The modified script produces an initrd image containing the customized files and modules required to use Lustre as a root filesystem. This includes several of the Lustre modules, a TFTP client for downloading supplemental modules, modprobe, and additional utilities for debugging.

We also automated the creation of PPC970 CHRP boot packages which contain the kernel and initial ramdisk. Because the changes are localized, the software may be run on a clean system installed from vendor media. The underlying operating system kernel and modules are included in the Lustre-boot ramdisk. This functionality can be used to quickly update the boot images based on a single running node, allowing a single configured node to be used to create an entire cluster with minimum effort.

5 Results

Our system boots a cluster of computational nodes using Lustre as a root filesystem. While the nodes provide full functionality, it is important to examine the performance implications of using a shared storage cluster to host essential operating system files. We ran a series of tests, ranging from single-node interactive tests to MPI parallel applications, to determine the effect of using the Lustre root filesystem on computational job performance.

Our test environment is composed of a computational cluster and a Lustre storage cluster. The computational cluster consists of 28 IBM JS20 blades with each blade a dual-processor 1.6 GHz PowerPC 970 based system containing 2.5GB of RAM. The storage cluster consists of two identical IBM x345s, each with dual 3.06 GHz Intel Xeon processors and 2.5GB of RAM. Both servers function as Lustre object storage servers (OSS), and one also provides the Lustre metadata service (MDS). These servers use SuSE SLES9 with kernel 2.6.5-7.201-smp and version 1.4.5 of Lustre. In this environment, Lustre has been used for several months to provide high-performance scratch space for parallel jobs and as storage capacity for system and user backups. Our research leverages our existing Lustre infrastructure to provide the root filesystems for the computational clusters.

We dedicated 8 nodes of the JS20 cluster to testing the Lustre root filesystem, used 8 nodes for baseline performance testing, and allowed our users to continue running jobs on the remainder of the cluster. The /home and /opt partitions were mounted via NFS to our existing cluster management infrastructure. We intend to mount /home using NFS for the foreseeable future, but will mount /opt using Lustre when adopting the Lustre root filesystem throughout our datacenter.

We ran a series of tests to determine the effect of the Lustre root filesystem on single-node interactive operations and parallel computational jobs. We examined the amount of time required to boot a single node and groups of nodes. For single-system performance, we examined the time to perform many common system calls. To examine parallel application performance, we benchmarked Biome-BGC [19], High-Performance LINPACK (HPL) [5], and the Parallel Ocean Program (POP) [11]. All of the applications were compiled using the IBM XL compiler suite using MPICH.

5.1 Node Boot Time

We measured the amount of time required for a node to boot starting from the CSM "rpower on" command (see Fig. 1). By examining the timing of the TFTP requests for the network boot blades, we could also differentiate the power-on self-test (POST), the kernel load time, and the time required to run the init sequence. Nodes were considered fully booted when they responded to an interactive SSH request. The SSH connections were attempted every 5 seconds. The blades with hard drives were timed manually, but the manual timing does not distinguish the initrd init process from the kernel boot times because of its short duration.

Using a local hard drive, a single JS20 blade required an average of 3.04 minutes (SD=0.06 min) to boot. Using the network boot with the Lustre root filesystem, a single JS20 blade booted in an average of 6.25 minutes (SD=0.30 min). Booting up to 16 systems simultaneously had no noticeable effect on the boot time.

The network boot using the Lustre root filesystem requires more time for several reasons. When booting the blades over the network, the CHRP boot loader must wait to receive a DHCP addresses on both interfaces. The boot loader must then retrieve the kernel and initrd using TFTP. This process requires approximately 3 minutes, which exceeds the entire boot process on a node with a hard drive. After retrieving the kernel and initrd, the blade must retrieve additional kernel modules using TFTP and mount the Lustre filesystem. In addition, all of the nodes access the same files when booted simultaneously, so network, TFTP server, and storage server contention could increase the boot time. In a larger deployment, we anticipate that TFTP servers will be hosted on all of the Lustre OSTs.

Our test cluster included several blades with failed hard drives. During our testing we noticed that the blades with failed drives required an additional minute to boot. The kernel appears to spend this time probing the failed disk. This fails with an IDE error and delays but otherwise does not affect the boot process.

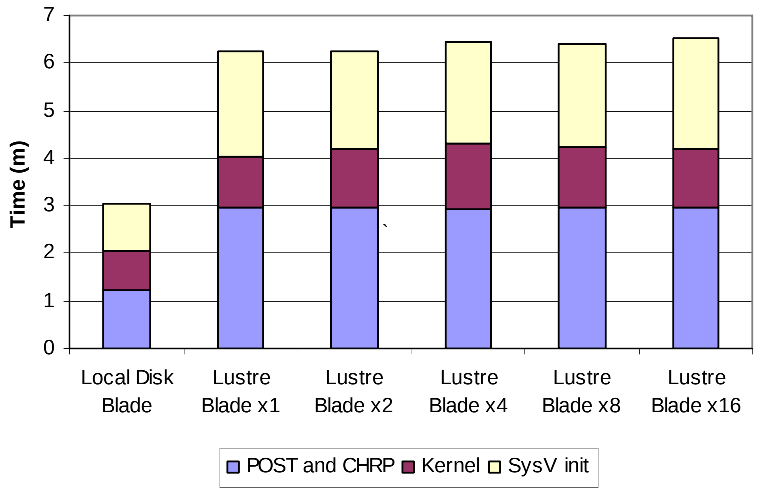

Fig. 1. System boot times for IBM JS20 blades using a local hard drive and a Lustre root filesystem

5.2 Micro-Benchmarks

We used a custom benchmark to test the performance of several common library and system calls when run on nodes using disk-based or Lustre-based root filesystems. These tests iterated over a library call many times in order to achieve a coarse grained measurement of the library call cost. For this battery of tests, we analyzed the time to create and join pthreads using pthread_create() and pthread_join(), the time to create a TCP socket using socket(), the time to copy memory between pre-allocated buffers using memcpy(), and the time to create and free a dynamically allocated buffer using malloc() and free(). Some of the tests produced small differences, but the results did not demonstrate any obvious distinction between nodes using Lustre root file systems and disk-based root file systems.

5.3 Serial and Multithreaded Execution of Biome-BGC

The Biome-BGC climate model [19] was also used to measure application performance for both boot configurations. Biome-BGC simulates the carbon cycle and plant growth rates based on meteorological data. The model is embarrassingly parallel and computes the carbon cycle parameters iteratively over a given number of grid cells in a geographic domain with no boundary conditions. This application is multithreaded using the Linux pthread library. Biome-BGC is invoked by a script that performs some model initialization and cleanup. We reported the user, system, and real (wallclock) execution time as measured by the Linux time command (see Table

1). The real time is relevant for our analysis because it shows the whole system performance as measured by the turnaround time for the entire application. As a system metric, it includes the effects of all running processes.

The Biome-BGC timing results show performance similar to HPL for disk-based and Lustre root filesystem configurations. When running one thread per node, the Lustre root blades outperform the disk-based blades by 0.7%, but when running two threads per node the disk-based systems outperform the Lustre-based nodes by 2.8%. We believe that the real time is adversely affected by other processes running on the system, such as Lustre and kernel helper processes that contribute to the node's load. While the amount of user time consumed by the application remains similar between the configurations, we observed small increases in the system time when using the Lustre root file system as well as small increases in the real execution time due to additional background load.

Table 1. Biome-BGC execution time for HDD and Lustre boot configurations on IBM JS20 blades in single threaded and dual threaded configurations.

Biome-BGC Time: 1 thread/node

	HDD Root (s)	Lustre Root (s)	Overhead (s)
User Time	10826.4	10723.0	-103.37
System Time	6.1	16.2	10.19
Real Time	10837.4	10756.4	-81.04

Biome-BGC Time: 2 threads/node

	HDD Root (s)	Lustre Root (s)	Overhead (s)
User Time	10824.9	10737.3	-87.56
System Time	6.5	20.8	14.29
Real Time	5445.38	5602.5	157.12

5.4 Parallel Execution of High-Performance LINPACK

We used the High-Performance LINPACK (HPL) benchmark [5] to measure performance for both the disk-based Lustre root filesystem configurations. HPL was compiled using GotoBLAS [4] and executed using N=10,000 and NB=256. PFACT and RFACT do not significantly affect LINPACK on PPC970 blades [20], so these parameters were arbitrarily set to "right." We ignored initialization and cleanup times and reported the internal timings for the WR00R2R2 test case.

The performance of HPL on Lustre-based blades closely follows the performance of the disk-based blades. Using the Lustre root filesystem increased the runtime of the HPL benchmark by about 3% when run with two processes per node (see Fig. 2). When run with one process per node, the Lustre root blades outperformed the disk-based blades for tests using more than one node.

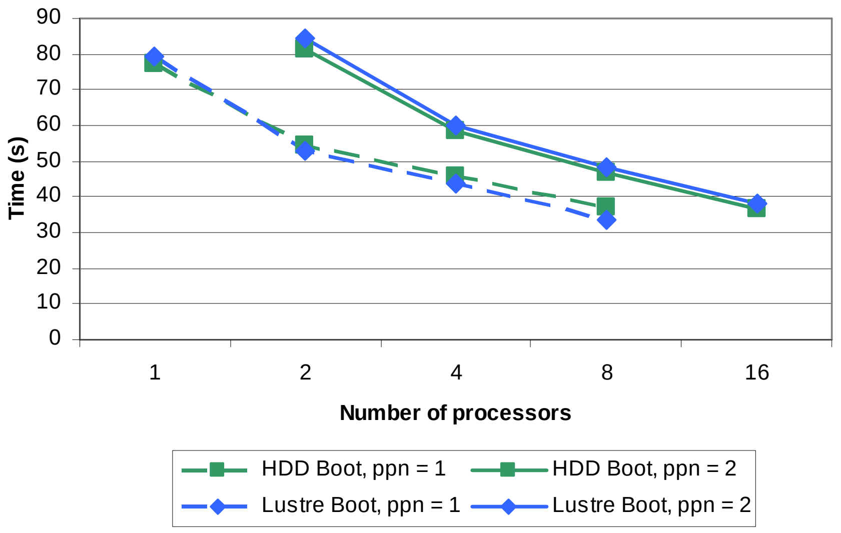

Fig. 2. High Performance LINPACK benchmark time on IBM JS20 blades with hard drives and a Lustre root filesystem

5.5 Parallel Execution of POP

The Parallel Ocean Program [11] is the ocean circulation component of the Community Climate System Model (CCSM). Prior work has shown that the performance of the memory subsystem can constrain the performance of POP [22]. For our test, we compiled POP with a 192 x 128 grid, 20 vertical levels, 16 x 16 blocks, and a maximum of 92 blocks per processor.

We executed POP using one and two processes per node for each boot configuration and reported the execution time as measured by the Linux time command (see Fig. 3 and Fig. 4). The results show that the wallclock time in both the disk-based and Lustre boot configurations are similar, but the disk-based nodes perform slightly better than the Lustre-based nodes. The user times for POP showed no difference due to the root filesystem implementation, but the system times increased when run on a node using the Lustre root filesystem (see Fig. 4). The additional system time consumed by POP using Lustre appears to be proportional to the number of processors. We therefore conclude that this performance penalty is constant per processor and that using the Lustre-based root filesystem imposes a small yet consistent overhead. We confirmed this time increase by examining the model's internal timer. For dual-processor runs, for 2 processors the overhead of using the remote filesystem was 1.05%, but for 16 processors the overhead increased to 18.8%.

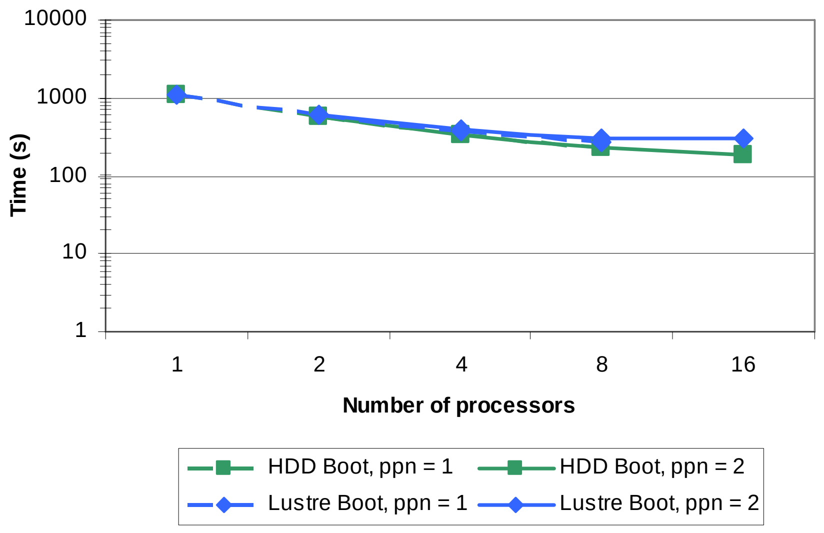

Fig. 3. POP192 real time on IBM JS20 blades with hard drive and Lustre root filesystems

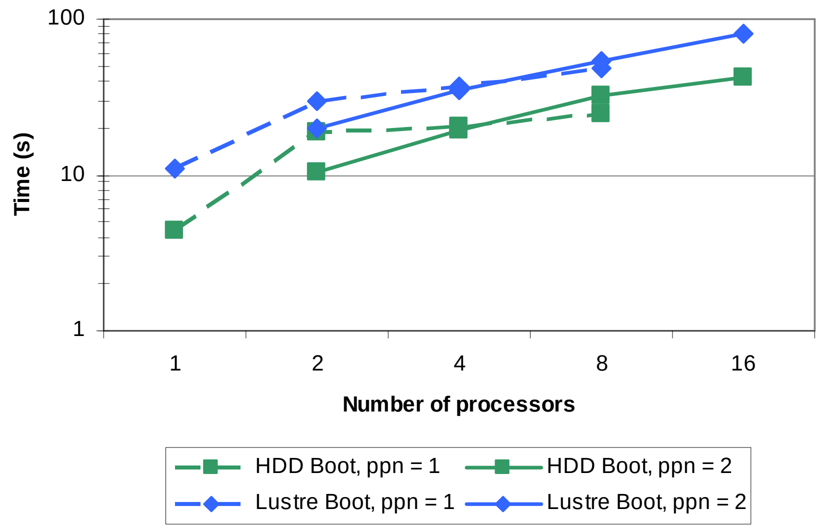

Fig. 4. POP192 system time on IBM JS20 blades with hard drive and Lustre root filesystems

Applications running on nodes using a Lustre-based root filesystem experience a small performance penalty in some cases. For Biome-BGC and HPL, tasks using only one CPU per node actually performed better when using Lustre-based root filesystems. For dual-CPU Biome-BGC, dual-CPU HPL, and POP, the application walltime was slightly greater when running on a Lustre-based root filesystem. The time increase is small, such as a 2.16m walltime increase for a 90.75m Biome-BGC job, or a 1.3s internal timer increase for a 36.5s 16-processor HPL run. The increase in run time appears to be due to two factors: increased background load on the node and additional system time accumulated by the task. Unfortunately, we did not have the opportunity to compare Lustre-based root filesystems to other network-based root filesystems, such as NFS remote root. Therefore, we are unable to compare the overhead imposed by the Lustre root filesystem with alternate diskless node solutions.

6 Discussion

Simplifying cluster maintenance is one of the primary objectives for centralizing computational node root filesystems. Software installation for our cluster was traditionally done locally via SuSE's YaST2, across the cluster with CSM's smsupdatenode command, or by using RPM directly. For a moderate sized cluster these mechanisms are likely sufficient but they also have several drawbacks. Each of these processes relies on a chain of dependencies to operate correctly, and updates are not atomic across the cluster. In the case of a single or multiple failed nodes, maintaining consistency becomes an issue, requiring the administrator or the management software to repeat the work at a later time. Many sites choose to deal with this problem by requiring partial or complete cluster downtime during the installation, particularly if a reboot is required or critical processes need to be restarted. Extensive up-front testing must also be performed to ensure a smooth transition due to the difficulty of backing out changes.

With a centralized root filesystem upgrading a single node, a set of nodes, or all of the nodes simultaneously becomes nearly transparent. Common system maintenance, such as file distribution, log rotation, and other routine tasks can similarly be performed from any management system that can access the Lustre filesystem. Programs on each node must still be restarted or notified when their data or log files have changed. Testing can be done with an alternate copy of the common root filesystem that can be mounted by a set of test nodes prior to production use on the full cluster. Install-time errors caused by full disks, slight differences in node disk size or layout, and other problems that occur with disk subsystems when installing new software at scale are virtually eliminated. Newly installed or modified files on the shared portion of the filesystem are immediately updated on the remaining nodes, and the coherency of the filesystem is maintained by Lustre. As an example, a cluster-wide upgrade of PBSPro was accomplished by remounting a single node's root filesystem read-write and performing the normal upgrade procedure, then returning the filesystem to read-only. The per-node files and directories created in /var/spool were easily distributed to the other nodes via a copy command on one of the Lustre storage servers.

A possible alternative to modifying the node root filesystem directory on a live compute node is to mount the root and local hierarchies read-write on a currently running system, then spawning a chrooted shell in that directory to emulate the compute node configuration. Prototype and backup copies of a system can be mounted in this manner too, providing a mechanism for testing a new installation by simply changing where the client mounts its filesystems from in the configuration file. Rolling back when problems occur is easy as the node's original filesystem remains unchanged. While we wrote a script to create this environment, we experienced difficulties issuing the chroot to a path in the Lustre filesystem. CFS has released a patch for this issue and further investigation is needed.

Centralized control often brings with it concerns about security. NFS provides root UID remapping, the nosuid flag to avoid vulnerabilities from set-UID root programs, and IP address-based client authentication. NFS is also vulnerable to IP spoofing, capture of unencrypted data streams, and denial-of-service attacks targeted at the single NFS server. Lustre is also vulnerable to many of these same attacks.

In light of these security concerns, centralizing the root filesystems offers benefits in the areas of system hardening and incident response. Each node is protected from many attacks due to the filesystem being mounted read-only, and a potential breach can be quickly resolved by restoring the directory from backups and rebooting the nodes. Moreover, common file integrity scanning techniques are run from a single management node with access to the Lustre filesystem, verifying the filesystems for the entire cluster with a single pass.

The scalability of diskless nodes is dependent on multiple factors and our design has not addressed all of them. The system boot process is still dependent on a single server to provide the BOOTP and TFTP services, and this can limit the number of nodes that can boot simultaneously. However, the TFTP service can be replicated on multiple servers, and BOOTP can be used to distribute the load among them. Another possible optimization for improved scalability is to use filesystem images instead of a directory hierarchy, and loopback mount these images in place of the bind mounts. This would reduce the metadata activity on the MDS that could improve the performance for tasks such as compiling and improve the local interactive experience.

Our work provides numerous benefits from the use of diskless nodes while exploiting the performance, scalability and reliability already in the I/O subsystem for the computational node root filesystem. While we developed our solution using Lustre, any filesystem that only requires kernel modules could be used as a root filesystem in this manner. Our solution should be sufficiently general to work with many of the common high-performance parallel filesystems being deployed for Linux cluster systems.

7 Future Work

The results of our work have shown that a Lustre root filesystem imposes a small but acceptable performance penalty on applications when compared to disk-based systems. We have not yet compared this to the overhead imposed by other remote root filesystem implementations. To provide a more appropriate comparison, we intend to examine the performance of applications on systems using a NFS root filesystem. We anticipate that Lustre's performance benefits over NFS will become apparent in this comparison.

To further explore application performance, we will expand our analysis suite to include additional benchmarks. Timing the execution of common system commands that result in filesystem access as well as measuring compilation time for popular packages, will quantify interactive and local system response times. For scalability performance, we will also benchmark the NCAR High Order Method Modeling Environment (HOMME).

While we have successfully booted the PowerPC 970 cluster with a Lustre root filesystem, we intend to extend the test environment to include our 64-node Intel-based cluster. This will verify that the implementation functions across heterogeneous hardware platforms using both the PXE and CHRP booting standards. More importantly, it will verify the robustness of the design, implementation, and automated build process. The larger cluster will also allow us to further examine the scalability of our approach. While our existing infrastructure supports booting 20 nodes simultaneously without problems, booting 84 nodes may expose limitations that require additional parallelism in the TFTP service.

In addition to fully supporting Lustre root filesystems on our production clusters, we intend to thoroughly document and distribute our work for public use. Our goal is to provide a set of patches to the core SuSE management scripts and utilities to allow a site with a pre-existing Lustre storage cluster to easily leverage diskless boot for their computation nodes. Should this implementation be accepted by the community, we hope that support for Lustre root filesystems will become a part of future Linux distributions.

8 Conclusion

Current Linux clusters typically include a high-performance parallel filesystem to satisfy application I/O requirements. Despite the presence of high-performance filesystems, computational nodes still rely on commodity hard drives or NFS-based root filesystem implementations to host an operating system. Through small changes to the Linux boot sequence, we have introduced a mechanism to provide the root filesystem on diskless computation nodes using Lustre. In addition to eliminating the downtime caused by disk failures, this solution leverages the high-performance Lustre filesystem to improve node stability and reduce administrative complexity.

The use of Lustre-based root filesystems simplifies cluster installation and administration. Node installation and maintenance is performed from a single system, eliminating the need to maintain node synchronization manually or through software deployment tools. Not only is node management simplified, but our results show that use of Lustre for computational node root filesystems imposes an acceptable overhead compared to disk-based root filesystems. We are currently finalizing plans to remove the hard drives from our IBM JS20 blade cluster and use the Lustre root filesystem in a production environment.

9 Acknowledgements

We would like to thank Peter Braam, Jeff Denworth, Oleg Drokin, and Peter Bojanic of Cluster File Systems for their assistance with this project. University of Colorado computer time was provided by equipment purchased under NSF ARI Grant #CDA-9601817, NSF sponsorship of the National Center for Atmospheric Research, and a grant from the IBM Shared University Research (SUR) program.

References

1. S. Aiken, D. Grunwald, A. Pleszkun, J. Willeke, “A Performance Analysis of the iSCSI Protocol”, proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), San Diego, CA, April 2003.
2. G. Beekmans, Linux From Scratch: Creating the mtab symlink, 2002. http://www.faqs.org/docs/linux_scratch/chapter06/mtablink.html
3. Extreme Cluster Administration Toolkit (xCAT), 2006. http://www.xcat.org/
4. K. Goto, GotoBLAS, 2006. http://www.tacc.utexas.edu/resources/software/
5. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://www.netlib.org/benchmark/hpl/
6. IBM, General Parallel File System, 2005. http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.html
7. IBM, IBM Cluster Systems Management (CSM), 2006. http://www-03.ibm.com/servers /eserver/clusters/software/csm.html
8. J. Cope, M. Oberg, H. M. Tufo, and M. Woitaszek, “Shared Parallel File Systems in Heterogeneous Linux Multi-Cluster Environments”, proceedings of the 6th LCI International Conference on Linux Clusters: The HPC Revolution, Chapel Hill, NC, 2005.
9. Knoppix, “What is KNOPPIX?”, 2006. http://www.knoppix.org/
10. Kostyrka, A. NFS-Root mini-HOWTO. 20 September 2002. http://www.tldp.org /HOWTO/NFS-Root.html
11. LANL, The Parallel Ocean Program 2.0, 25 March 2003. http://climate.lanl.gov/Models/POP/index.htm
12. Lustre, Lustre Bug 7013 Feature Request: Mount a subdirectory, 2005. https://bugzilla.clusterfs.com/show_bug.cgi?id=7013
13. Lustre, The Lustre Book, March 2004. http://www.lustre.org/docs/lustre.pdf
14. R. Minnich, J. Hendricks, and D. Webster, “The Linux BIOS”, The Fourth Annual Linux Showcase and Conference, Atlanta, GA, 4 October 2000. http://www.linuxbios.org /data/papers/als00/linuxbios.pdf
15. P. Morjan, “DIM: Diskless Image Management,” IBM presentation, September 2005. http://www.bsc.es/publications/documentation/pdf/dim_barcelona.pdf
16. C. B. Morrey III, and D. Grunwald, “Peabody: The Time Travelling Disk”, proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03), San Diego, CA, April 2003.
17. H. Ramachandran, Design and Implementation of the System Interface for PVFS2. M.S. Thesis, Clemson University, Clemson, South Carolina, December 2002.

18. Terrascale Technologies. Terragrid Overview, 2005. http://www.terrascale.com/prod_over_e.html
19. P.E. Thornton, B.E. Law, H. Gholz, K. Clark, E. Falge, D. Ellsworth, A. Goldstein, R. Monson, D. Hollinger, M. Falk, J. Chen, and J. Sparks, “Modeling and measuring the effects of disturbance history and climate on carbon and water budgets in evergreen needleleaf forests”, Agriculture and Forest Meteorology 113, 185-222, 2002.
20. V. Vdovikin, “Running High-Performance LINPACK on IBM pSeries JS20 Cluster with Myrinet Interconnect”, 2006. http://www.jscc.ru/informat/Linpack-report.pdf
21. Voltaire, “Voltaire Introduces New, Higher Performance I/O Virtualization Products for InfiniBand Fabrics”, 28 June 2004. http://www.voltaire.com/press_release_06_28_04.html
22. M. Woitaszek, M. Oberg, and H.M. Tufo, “Comparing Linux Clusters for the Community Climate Systems Model”, proceedings of the 5th LCI International Conference on Linux Clusters: The HPC Revolution, Austin, TX, May 2004.