NFS, Linux, and clusters: Network storage and its future

Brian Pawlowski
beepy@netapp.com
Trond Myklebust
trond.myklebust@fys.uio.no

This paper is a draft position paper and and work in progress for an invited talk to the SANE 2004 Conference in Amsterdam.

This paper explores requirements and approaches to storage in a Linux cluster environment, and discusses specific features NFS Version 4 that make it well suited for use in a cluster computing network. While this paper primarily concerns itself with NFS, alternatives will be briefly described. The paper ends by touching on some future directions for NFS in cluster applications.

Introduction

The Network File System (NFS) has been in existence for 20 years and provides the underlying distributed file system architecture for most networked Unix inspired operating systems today. NFS was first deployed in SunOS and later ported to or re-implemented in other operating systems to provide remote file access. NFS has always been easy to implement and easy to deploy due in part to its lean design and simplified error recovery mechanisms. [Sandberg85]

In the 1990's, as networking speeds increased to match traditional storage interconnect speeds, the concept of Network Attached Storage (NAS) as an alternative to Direct Attach Storage (DAS) or SCSI-based Storage Area Networks (SAN) took hold. A NAS storage architecture was typified by the use of standard TCP/IP and Ethernet as the data transport. In contrast to Fibre Channel-based SANs, NAS deployments were (and are) perceived to provide more cost-effective solutions to networking storage. Regardless of how storage is networked, releasing captured DAS storage allows interesting solutions to problems of backup and disaster recovery leveraging the network. Typical applications included shared home directories, software development and engineering applications. In roads were made in hosting more traditional enterprise applications, like database, over NFS. NFS evolved slowly to meet increasing requirements for performance and data access during this time. [Pawlowski94]

The emergence of large scale compute clusters, first using BSD Unix and later Linux on top of commodity Intel x86 compatible hardware, put NFS into an increasingly important role in large, cost-effective, scalable storage deployments for clustered applications. Scalable e-mail architectures and web service farms led the charge. Inherently partitionable applications that arise in CAD design and simulation, animation and special effects rendering were also well suited for a scalable compute farm attack. Reducing the cost of application deployment using commodity compute hardware horizontally scaled was the driver.

We now live in interesting times, and much of the interest lies in using things like Linux and clustering to support cost-effective high-performance application deployment. A new version of NFS, Version 4, is in the early stages of adoption and deployment. Some of its design features are intended to provide better behaviour and performance for clustered applications. [Pawlowski00] [Adamson01]

File systems for a Linux cluster

It is useful to consider the requirements for and the issues facing a storage architecture and file system in a Linux cluster environment. A brief discussion of the goals and motivations of cluster computing will shed some light on the requirements on storage to support it.

The Linux compute cluster

Cluster computing arose originally from a need to provide performance scalability beyond the capabilities of any single large compute node on problems involving massively parallel tasks. Today, as cluster computing has become more and more mainstream it has been seen increasingly as a way to aggregate computing resources of commodity computers to achieve better price/performance compared to large monolithic SMP application servers.

A *Linux compute cluster* is first and foremost cost driven. Although compute clusters are certainly not a new concept, the cost-driven basis for Linux compute clusters have implications for subsequent storage decisions in support of the compute farm. Cost reductions in deploying Linux compute clusters arise perhaps less from the "free" software aspect of the solution and more from the ability to exploit cheap commodity hardware that provides good performance.

Realistically, Linux provides additional cost savings over other x86 platforms enablers like FreeBSD because of the growing infrastructure, expertise and support options available. But FreeBSD remains an alternative, and Sun is investing anew in Solaris x86. Microsoft Windows server is also seeing use in scalable compute clusters – emphasizing the underlying commonality of the x86 as the commodity platform of choice. As the compute elements become commoditized, so in turn does the operating system. Applications drive the compute cluster architecture and have few if any dependencies on a particular operating system running on a node. (It is the case that multiple operating systems may co-exist in a compute cluster – this can easily arise in a rolling migration to a new operating system platform for an application. A Unix system is a Unix system is a Unix system.)

The key feature of the compute nodes in the cluster is that they are interchangeable so that any node may serve a portion of the clustered application and be replaced or redeployed in event of node or network errors. A compute node holds no significant state, and the cluster needs nothing other than cursory recovery (primarily through job reissue to another node in the cluster after some job queue management cleanup). The symmetry is relatively weak. Linux compute cluster architectures can absorb new hardware platforms sporting enhanced capabilities (threaded or multi-core CPUs, for example). Differences in performance of the individual nodes can be managed by defining small discrete units of work and issuing additional units (jobs) to more capable nodes in the cluster. But from the application viewpoint, during execution, the compute nodes are the same.

Compute clusters are horizontally scalable – additional processing capacity is achieved by adding additional similar nodes to the network.

In a cost-driven model for a compute cluster that leverages commodity components, the cluster interconnect or fabric itself is a commodity. TCP/IP and Ethernet are the cluster interconnect of choice for Linux compute clusters.

In large Linux compute clusters, failure is not only an option – it is a way of life. 10 – 20 compute node failures a *day* per 2,500 nodes is probably not uncommon. Of those, perhaps 2 or 3 per week represent a hard failure not cured by a reboot. (Interestingly, hard failure rates following initial installation of new compute nodes due to early mortality/DOA/marginal components/shipping damage are thought to run higher – as weak nodes are culled from the herd). The obvious candidates for failure arise (in the like order of): disk drives, memory, and power supplies. [Private communication] Using "commodity" hardware results in this trade-off of individual node reliability and overall cluster availability and performance. Diagnosing and replacing disk drives and power supplies is fairly straightforward. In driving down the cost of the compute nodes it can be the case that once a node suffers a "system" hard failure (not easily diagnosed to one of the above cases) it is simply cheaper to replace the compute node with a spare rather than attempt further diagnosis and repair.

Linux compute clusters consisting of 1,000's of nodes and their associated storage present site issues first in compute density (CPUs per m^2 of floor space) and then in providing sufficient power and cooling for the cluster. Blade architectures attempt to optimize power and cooling requirements for a given amount of CPU through innovative packaging.

Linux compute clusters probably exist in fundamentally two different forms reflecting different applications. The following taxonomy is probably a gross over-simplification of application specific cluster designs – but illustrates that there exist wide variations in cluster size.

In the first application type, large clusters of 2,000, 5,000 and going to 10,000 nodes within the next two years are used for highly partitionable applications that can be executed in parallel threads. [Private communication] While read sharing occurs, writing is strictly partitioned. Such applications include search engines, e-mail, animation and rendering, financial modeling, scientific simulations and engineering applications.

Interestingly, large partitionable applications deployed via a job queue management tool such as the Load Sharing Facility (LSF) do not depend on distributed locking in the underlying file system. In a completely partitionable application, the unique output of each job in a cluster is implicitly locked through job queue scheduling. Otherwise, application specific locking is often deployed.

The second application type of interest deployed on Linux compute clusters is the scalable database. These deployments typically involve 128 nodes or less and have a higher degree of write sharing and locking than the large scale, highly partitionable applications mentioned above. Locking is controlled by the database application itself. Database integrity is supported by fencing operations and recovery techniques (that may require feature support in the underlying storage architecture).

Possibly a third cluster type exists in extreme high performance computing with I/O bandwidth requirements per node that are not met with commodity TCP/IP and Ethernet networking. In such clusters, the I/O bandwidth available to any one storage node is insufficient and so a requirement exists for aggregating bandwidth over several storage devices and networks. This paper will not go into

these issues (except to touch on some future work in NFS).

Clusters vs. Grids

The term "Grid" has become overused to define any cluster of computers. Distinguishing cluster computing from Grid computing is useful in gaining insights into the requirements for scaling compute clusters.

Classically, Grid computing takes the form of sets of toolkits and technologies to construct pools of shared computing resources. The Grid technique arose to attempt to tie together geographically dispersed computing centres to satisfy the exponentially increasing demands of scientific research. It is the geographically dispersed aspect that most distinguishes a classic Grid computing architecture from simple cluster computing. Analogies are made to "the power grid" utility approach in defining a similar model for computing.

Grid computing places additional constraints on accessing shared storage. Once outside the confines of a data center, uniform global naming policies for data access to allow sharing become confounded. A fine problem arises because what is really needed is support for local autonomy in managing well-defined portions (sub-trees) of a global namespace with changes appearing to other (geographically remote) parts of the Grid.

There is usually a strong relationship between well-defined "local" sub-trees in the global name space and security administration domains. Outside a single data center providing secure access to shared data suggests the use public key authentication. The problem of remote private key distribution is impractical in general.

Grid computing can arise in the small even within an organization's internal network. While a Linux cluster may exist in a well-managed data center it may well be that Linux computers exist on user desktops, particularly in technical enterprises. Grid computing techniques can be used to harness unused desktop compute cycles off hours to apply to parallel application problems using the same job queue management as in the Linux cluster proper.

Fine points aside on the differences between clusters and Grids, you will often find the terms used interchangeably.

A scalable storage architecture

The goal for storage in a Linux compute cluster can be succinctly framed as the answer to the question "Why can't my storage scale as easily as my compute nodes?" Answering that seemingly simple question defines the issues and requirements in providing a scalable storage solution for Linux compute clusters.

The storage network of choice for Linux compute clusters is TCP/IP and Ethernet. Ubiquitous, high-speed, existing infrastructures can form the basis for data sharing in the compute cluster. NFS is well suited for Linux compute clusters since it enables the use of commodity networking.

From a feature standpoint there are several key factors to be addressed by a storage solution.

First is the requirement for **data sharing**. To enable cluster computing data must be accessible by all compute clients (application servers). Clearly this can be achieved with a distributed file system such as NFS. Hidden in the data sharing requirement is a requirement for a

uniform global (to the extent of the cluster) name space.

Second is the requirement for **data availability**. While compute nodes are essentially "stateless" and individual node failures are transparent to the overall cluster availability and performance, data access failures are not. While relatively inexpensive (and unreliable) x86 compute nodes are used in a Linux cluster, highly available storage nodes are often deployed. Disks fail, power supplies fail. RAID and redundant power supplies are just two of the features required for storage in a compute cluster. The storage nodes have no single point of failure and are redundantly connected to the cluster fabric to provide continuous access even in the face of partial network failures.

A major contributor to storage and data availability is support for non-disruptive upgrades, expansion and additional storage provisioning. Additional storage capacity is configurable on more capable storage arrays without affecting client (compute node) operation.

Third is the requirement for **reliable data handling**. Often compute nodes are essentially stateless. Storage nodes are always inherently stateful, with each node holding unique instances of persistent state – the application's data. (Replication for improved availability of shared libraries and techniques such as remote mirroring for disaster recovery are refinements on the basic model.) Reliable data handling is different than making data available. It is the property that the data stored to a storage device is what is later delivered back to a client. More capable storage arrays provide redundancy checks on the data to verify and repair data in the face of possible corruption.

A fourth requirement is **security**. While physical security of a storage network may be sufficient for many installations (as is the case with most fibre channel SANs), strong security based on encryption technology is usually a requirement of a TCP/IP-based architecture today (regardless of whether it is actually enabled or not). The IETF mandates strong security for all new protocols defined for the Internet (including NFS Version 4).

The fifth requirement bears reiterating because it is perhaps too obvious, but for large commodity Linux clusters, in particular, the storage **solution must support commodity TCP/IP and Ethernet networking**.

The sixth requirement is **performance**. While the compute nodes are chosen based on cost, and don't necessarily have to be the highest performing single node performance available (as overall cluster performance is achieved through sheer numbers), storage nodes have stricter requirements. With a 100:1 fan in or greater of compute nodes to storage nodes, storage nodes may be specified at a higher performance to serve expected overall cluster I/O bandwidth requirements to a given storage node. Overall storage bandwidth is scaled horizontally as is done with the compute nodes.

While it is obviously useful to view scaling of the compute nodes separately from the scaling of the storage nodes in a cluster deployment, a refinement on the organizational model of a compute cluster and its associated storage is to define (and over time refine) provisioning additional capacity in terms of a discrete group of compute nodes and storage nodes. The combined set of nodes is the additional incremental capacity added in any expansion

operation. The node counts and ratio of compute nodes to storage nodes in an expansion group is application dependent. Such a technique is useful in managing, for example, a scalable e-mail architecture, where adding (many) additional users requires both additional storage and additional compute capabilities. Defining a group of nodes as the basis for management and expansion provides an opportunity to define partial failure domains within an otherwise uniform cluster (useful on the theory that partial failures are more acceptable than complete cluster outages). It is certainly the case that care must be taken when designing a Linux compute cluster to avoid any single point of failure in those components that provide "persistent critical state" (which can arise in job queue scheduling or the storage nodes).

Completely ignored in this paper are the rather complex issues of data management in the storage nodes. Investment in the storage nodes is usually made not only to provide highly available data to the compute nodes, but also to gain access to data management technologies for backup and disaster recovery.

Also ignored here is the trend towards asymmetric or hierarchical approaches to storage that is based on the premise that not all data is of equal value. Within a Linux compute cluster it is certainly the case that all storage nodes need not necessarily be equivalent in performance (or availability) but must transparent to the compute nodes in terms of functionality (data access, sharing and name space).

NFS Version 4: Features and issues

NFS is the ubiquitous distributed file system in all Unix environments today. It is bundled by default with all variants of "Unix", including Linux. Both NFS Versions 2 and 3 are in use today. NFS Version 4 is the emerging variant that is beginning to appear on more and more platforms.

NFS is designed to be upwardly compatible as new versions are deployed. Interoperability is achieved by having clients and servers supporting multiple versions of NFS simultaneously. Clients (in the case of a Linux cluster, the compute nodes) negotiate with an NFS file server (the storage node) the highest version of NFS mutually supported. Both clients and servers can communicate with any version of NFS the other end supports, and a server can be serving the same data via Versions 2, 3 and 4 simultaneously, This is potentially useful in support of rolling upgrades within a cluster. Storage nodes can be upgraded to support NFS Version 4 independently of the compute nodes. The compute nodes can be upgraded in a roll to enable NFS Version 4 on a compute node by compute node basis.

For existing deployments of NFS, NFS Version 4 provides additional capabilities (performance and reliability) transparently. Other features, such as enhanced security are enabled via administrator control or simply through use (as in defining Access Control Lists on a file).

NFS supports heterogeneous file sharing. This property of being able to access data uniformly regardless of the client operating system (data is usually in a specific format which can be understood by the application regardless of what operating system platform it is running on) has proven useful in implementing application migration strategies to new environments such as Linux. While an application binary executing on a

particular operating system platform may differ from another platform, its invocation and management is usually sufficiently similar such that a common application data format and NFS enables mixed operating system clusters that may arise during a long-term migration to a new platform. Local file systems, or vendor specific cluster file systems, over SAN architectures typically inhibit migration.

An important point in the design of NFS Version 4 is that while significant state has been introduced to enable new capabilities (such as delegations), the recovery model is primarily driven by the client and resembles the simplified error recovery model that proved successful in NFS Versions 2 and 3. That is, when a server reboots clients will retry their operations until the server comes back on-line. For large Linux compute clusters the "retry forever" client behaviour is critical in support of unexpected (recoverable) outages (the fabled unplugging of the file server by the cleaning crew) and in support of rolling upgrades of file servers (new file server software or firmware upgrades with reboot). Such behaviour is critical in the face of applications in which a compute node runs a long job and the cost of restarting that job on affected compute nodes due to a storage failure or upgrade begins to affect the overall cost savings obtained from a cluster approach in the first place.

Leases were introduced in NFS Version 4 to solve the problem of "lost locks" in NFS Versions 2 and 3 due to client failures while holding a byte-range lock. A lease bounds the lifetime of a lock held by a client, and it is the responsibility of a client to renew the lease periodically for the length of the lock. (Most lease renewals are implicit in NFS Version 4 as a client accesses a file while reading and writing.) Server recovery is simple. If a server reboots, lock recovery occurs during which time clients holding locks reestablish them. During this "grace period" no new lock requests are allowed. If a client crashes, all client locking state is lost (potentially with unflushed data) and the server will expire the lease on a lock allowing access to the data by other clients. This is obviously useful for a compute cluster with redundant nodes. However, practically speaking, lost data due to client crashes probably requires some application (or job queue scheduling) driven recovery – though it may be as simple as restarting a job (one compute node's worth of work) from scratch.

Of some concern in NFS Version 4 is the behaviour of compute nodes when a network partition occurs. The problem is that from the server's viewpoint a client crash cannot be distinguished from a network partition. But automatic lock recovery on lease expiry may or may not be the policy a particular site wants to implement (though the alternative of never releasing locks in the event of the crash of one or more compute nodes is certainly no better). The handling of network partitions (that is, server behaviour to clients being inaccessible for any reason) is under discussion at this time in the NFS community. In a Linux compute cluster where application or job queue management initiated recovery simply consists of restarting a job on available nodes, lease expiry by the server in the face of partial network partitions is probably appropriate.

Delegations were introduced in NFS Version 4. A delegation is a capability granted by the server to a client that allows a client to aggressively cache data and locking state in the absence of conflicting data access. Read sharing enables caching of shared data

libraries in a Linux cluster. Write delegations allow a client (compute node) to cache modified data reducing latencies for the application and perhaps network traffic overall. Delegations are currently bound to a session and are granted on OPEN of a file. A server will recall a delegation if a conflicting request comes in from another client. Once recalled, the delegation cannot be reinstated short of reopening a file. The latter is considered a limitation to be fixed in a minor revision on NFS Version 4. NFS Versions 2 and 3 can co-exist with NFS Version 4 even with delegations enabled. Access to files by an NFS Version 2 or 3 client may result in a recall of a delegation. The server resolves the conflict. The net effect of multiple version access is no disruption to transparent data access. Delegations are a performance optimization, not a cache coherency mechanism. In the face of write sharing, for example, delegations are disabled for the file in conflict and traditional NFS behaviour ensues (that is, undefined in the absence of explicit locking). Delegations depend on the ability of the server to callback to a client, over a different port. It is not firewall friendly and its use in Grid computing is problematic and this is another area of discussion in the community.

NFS does not (yet) define a global name space as part of the protocol, and tools and policies need to be introduced to allow uniform data access. Various client-driven technologies, such as amd [Pendry91] or Autofs [Labiaga99], provide the uniform global name space for NFS required in a Linux compute cluster. The name space is defined by *maps,* accessed via a name service, that allow clients to construct uniform views of the data. The maps allow specification of alternate server locations for shared read-only libraries – these can be used to implement read-only replicated libraries distributed across several file servers to eliminate hot spots. While it is true that each node added to a cluster can be configured with a set of static mount points, adding new storage or reconfiguring storage becomes intractable without the ability to effect global changes to 1,000's of compute nodes via automounter-like tools.

NFS Version 4 simplifies the definition of the automount maps by providing a single uniform name space per server rather than a set of distinct export points that need to be mounted individually by the client. This is a small but useful simplification particularly in the face of large primary file server deployments as it allows for dynamic reconfiguration of the namespace by the server without disturbing compute node operation.

Large Linux compute clusters have special issues in scalability unrelated to NFS Version 4 specifically, but bear mentioning all the same. The total storage for a 2,000 node Linux compute cluster can reach 500 Terabytes, The data will be distributed over 20 or more storage nodes, with a compute node to storage node ratio of 100:1. Since all compute nodes require uniform access to all storage in the cluster (as each compute node is the same as any other), start-up events like initial cluster power-on or automatic reboot due to a site wide power outage result in extreme transients of activity along secondary data paths like initial mounting of file systems. Work has occurred in both the Linux NFS client and NFS file servers to gracefully handle and scale transient conditions in large cluster deployments.

One of the primary design points of NFS Version 4 was enabling its use over the Internet – thereby making NFS Version 4 a candidate for a Grid file system. While

the NFS protocol itself has always been defined to reside at a well-known port (2049), versions prior to NFS Version 4 required adjunct protocols such as mount, locking and status monitoring which were assigned dynamic port numbers. This made NFS difficult to deploy in the face of firewalls. NFS Version 4 collapses the protocols into one protocol residing at the well-known port.

A general-purpose security framework based on GSSAPI was introduced with NFS Version 4. Primary interest in the framework is the deployment of Kerberos as a strong authenticator for file access. For certain sensitive data deployments, NFS Version 4 allows an administrator to enable privacy (data encryption) to prevent snooping on the wire – at the cost of performance. Security is negotiated between the client and server. A server defines the set of acceptable security flavors required to access a given portion of its exported file systems. The security framework allowed the specification of a public key authentication flavor to support NFS in a wide area network, which is useful in enabling secure Grid computing. Unfortunately, implementation practice has fallen behind in delivering the public key authentication support (in favor of private key-based Kerberos authentication which has immediate applicability for current NFS installations). NFS Version 4 also introduced a standard ACL model to be used in conjunction with the strong authentication capabilities. Specification of user identities with security realm attribution in NFS Version 4 is also useful in Grid deployments.

Alternatives to NFS

NFS is a proven storage architecture that is in use both for high performance Linux database clusters and for large Linux clusters used to deploy highly partitionable applications. There are alternative approaches to storage architectures besides NFS that bear mentioning.

Traditional clustered file systems are typically used for highly I/O intensive applications such as clustered databases or file serving. By sharing block storage in a fully symmetric architecture, they tend to work well for medium-sized clusters of a few tens of nodes, but tend to fall prone to contention problems when scaling beyond a hundred nodes. Red Hat's GFS is a renewed attack on the clustered file system that is attempting to address the scalability issues. [Redhat]

It bears reiterating that the primary motivation for Linux-based compute cluster is cost. While clustered file systems can enable Fibre Channel-based SAN architectures in a Linux cluster, such deployments may only make economic sense if an existing Fibre Channel infrastructure with spare capacity exists. That said, it is the case that the cost of per port connectivity for Fibre Channel is dropping in price. As mentioned at the start, these are interesting times. But let's assume that Fibre Channel costs are an impediment.

Enter iSCSI. iSCSI is a specification for encapsulation of the SCSI protocol over TCP/IP (and Ethernet). [Meth03] iSCSI is all about reducing costs of deploying a SAN. It is essentially a replacement for Fibre Channel SAN block storage using commodity networking. Like NFS, iSCSI uses commodity networking as its underlying transport making it suitable for use in Linux compute clusters. However, iSCSI provides only block level semantics and requires the use of a clustered file system or similar

technology to provide the name space and data sharing semantics required.

Lustre is an advanced clustered file system for Linux which seeks to enhance scalability by means of a more asymmetric model in which the server tasks of handling file metadata and storing data are handled as two separate clustered processes. Its main focus is large compute clusters. [Cluster]

So, if all of the commodity Linux compute cluster nodes contain a disk used only for booting, and minimum disk sizes today are 60GB – what do you do with all that embedded unused disk space? Google answered this when they created the Google File System (Google FS). [Ghemawat03] Google FS uses the embedded disks available in the commodity compute nodes themselves to provide storage for a replicated and clustered file system. Some compute cycles of each node in the cluster are stolen to provide a distributed, replicated, resilient storage network. It is not entirely clear that the technique is generally applicable to further storage deployments, particularly in the smaller high performance database cluster architectures. Google FS clearly demonstrates a clever alternative approach to storage architectures for a scalable and highly available application.

CIFS, that is Windows network file sharing, is an alternative to NFS primarily in Windows networks. Efforts like Samba provide Windows file sharing capabilities in non-Windows environments.

No paper covering topics of data sharing, a global name space and networking can avoid paying homage to the Andrew File System. [Howard88] AFS defined the basic issues and explored solutions to secure, wide area file sharing. AFS still sees some use and is being actively worked on in the OpenAFS project.

Future directions and research

Unlike previous versions of NFS, Version 4 includes support for minor versioning to allow for small evolutionary changes of the protocol. NFS Version 4 is the basis for future innovation in NFS. The development of minor revision Version 4.1 is already well under way, notably with a proposal to extend the delegation model to include directories.

There is interest in providing more robust global name space support in future versions or implementations of NFS. While automounter-like techniques driven from the client are sufficient for local Linux compute clusters, widely distributed compute clusters present problems in new storage provisioning in a local part of the cluster and advertising the new storage to geographically remote nodes. A global name space driven by the servers themselves, with support from the protocol ala AFS is a potential area to explore. The model desired is simply “administer locally, access globally.”

Spinnaker Networks, later acquired by Network Appliance, created a set of technologies to cluster storage nodes and present a horizontally scalable cluster that provides a uniform clustered name space to all clients. The storage nodes act as general-purpose pools to hold data sets that can be migrated or replicated transparently to clients within the storage cluster to support storage load balancing. The techniques also support asymmetric or hierarchical storage architectures transparently. Technologies such as file switches provide similar virtualization and features.

NFS Version 4 has defined support for heterogeneous transparent migration and replication, but more work needs to be done in defining the server features required to support it. The feature is expected to be enabled in future implementations of NFS Version 4.

The current lack of a global Grid file system means that by far the most common method of accessing remote data on Grid projects today is to use FTP to download the files to local storage on the Grid compute node. The GridNFS project (an in-progress proposal at the CITI research group at the University of Michigan) aims to push those technologies needed to adapt NFS Version 4 to work with the Globus Grid toolkit, commonly used by the scientific community. By providing tools that adapt the existing NFS Version 4 public key authentication design to the Globus Security Infrastructure, and developing a global Grid namespace infrastructure, it will allow applications to securely access remote data without placing undue burdens on local storage or requiring significant rewrites of applications.

In the long term, it would be useful to adapt NFS to scale better for the case of large numbers of clients on remote networks. Ideas that are currently being explored include the on-demand creation of remote clusters of clients that can share both a delegation on a file and maintain a common cache to minimize the high latency traffic to the server.

In the mid-to-long term, local clusters will be required to scale beyond the current storage network bandwidth limits. Work suggested for Parallel NFS will go beyond current restrictions by allowing clients and compute clusters to perform NFS operations in parallel (striping across the network) and interact directly with a back-end SAN for the tasks of reading and writing file data. Seeking an open protocol solution for server-client communication, several vendors of such file systems have formed an informal working group dedicated to developing a minimal set of extensions to NFS Version 4 to support such models. Proposed extensions will allow clients to retrieve the block maps for a given file on the SAN. These maps may then be used to read or write to the file using a native block protocol such as iSCSI. In this mode, the client interactions with the server itself are limited to operations on the file metadata as per the Lustre model.

Another attack on high performance computing with NFS entails hardware assisted networking. Offloading the task of copying data between the operating system's networking buffers and the user's data buffers onto the networking hardware itself can result in performance gains for applications such as high performance clustered database. The technique of Remote DMA (RDMA) is gaining in acceptance with work within the IETF to produce an RDMA protocol that uses standard TCP/IP and Ethernet. [Mogul03] Other RDMA transports such as Infiniband also exist – NFS/RDMA efforts are RDMA interconnect agnostic. Work done with DAFS, a derivative of NFS Version 4, demonstrated the performance gains that could be achieved using such technology. [Talpey03] Revision 4.1 of NFS will include extensions that were derived from experience with DAFS. [Callaghan02] DAFS work suggests that the greatest gains in performance are to be had with application level RDMA access to data (and complete operating system bypass following initialization).

Acknowledgements

Approximate figures on Linux compute cluster sizes and management and issues

arose in several private conversations with several companies deploying such clusters for a variety of technical applications.

Thanks to Eric Melvin and Mike Eisler for valuable comments on a draft of the paper. Thanks to Skottie Miller for illuminating conversations describing practical experience in deploying large Linux clusters and storage.

Further information

A (currently) good web resource is the set of links found here:

http://www.citi.umich.edu/projects/asci/references.html

The web page for the NFS Version 4 working group within the IETF provides pointers to standard specifications (which are not mentioned individually in the bibliography) and work in progress focused on future extensions to NFS:

http://www.ietf.org/html.charters/nfsv4-charter.html

Information on parallel and high performance NFS can be found at this site:

http://www.citi.umich.edu/NEPS/agenda.html

Bibliography

The following bibliography is not meant to be exhaustive, but provides a starting point for additional reading and further investigation.

[Adamson01] Adamson, William A., Kendrick M. Smith. "Linux NFS Version 4: Implementation and Administration." OLS 2001 http://lwn.net/2001/features/OLS/pdf/pdf/nfsv4_ols.pdf

[Callaghan02] Callaghan, Brent. "NFS over RDMA." Usenix FAST '02. http://www.citi.umich.edu/projects/rdma/refs/callaghan-fast02.pdf

[Cluster] "Lustre: A Scalable, High-performance File System." http://www.lustre.org/docs/whitepaper.pdf

[Ghemawat03] Ghemawat,, Sanjay, Howard Gobioff, and Shun-Tak Leung, "The Google File System." SOSP 2003. http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf

[Howard88] Howard, J.H., M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West, "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems* 6(1). February, 1988. http://www-2.cs.cmu.edu/afs/cs/project/coda/Web/docdir/s11.pdf

[Labiaga99] Labiaga, Ricardo. "Enhancements to the Autofs Automounter." Lisa '99. http://www.usenix.org/publications/library/proceedings/lisa99/full_papers/labiaga/labiaga.pdf

[Meth03] Meth, Kalman Z., Julian Satran "Design of the iSCSI Protocol." Various including 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03). http://storageconference.org/2003/papers/19-Meth-Design.pdf

[Mogul99] Mogul, Jeffrey C. "TCP offload is a dumb idea whose time has come." Usenix HotOS IX Workshop 2003.

http://www.citi.umich.edu/projects/rdma/refs/mogul-hotosix.pdf

[Pawlowski94] Pawlowski, B. Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D., "NFS Version 3 Design and Implementation." Proceedings of the USENIX Summer 1994 Technical Conference. http://www.netapp.com/ftp/NFSv3_Rev_3.pdf

[Pawlowski00] Pawlowski, Brian, Spencer Shepler, Carl Beame, Brent Callaghan, Michael Eisler, David Noveck, David Robinson, Robert Thurlow. "The NFS Version 4 Protocol." SANE Conference, NL. 2000. http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf

[Pendry91] Pendry, Jan-Simon, Nick Williams. "Amd: The 4.4 BSD Automounter Reference Manual." http://docs.freebsd.org/info/amdref/amdref.pdf

[Redhat] Red Hat cluster file system technology. http://sources.redhat.com/cluster/

[Sandberg85] Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network Filesystem," *USENIX Conference Proceedings*, USENIX Association, Berkeley, CA, Summer 1985. http://citeseer.ist.psu.edu/rd/62292817%2C577908%2C1%2C0.25%2CDownload/http%3AqSqqSqwww.pdos.lcs.mit.eduqSq6.824qSqpapersqSqsandberg-nfs.pdf

[Soltis96] Soltis, Steven R., Thomas M. Ruwart, Matthew T. O'Keefe, "The Global File System." NASA Storage Conference 1996. http://citeseer.ist.psu.edu/soltis96global.html

[Talpey03] Talpey, Tom. "The Direct Access File System (DAFS)." Usenix FAST '03. http://www.citi.umich.edu/projects/rdma/refs/DA_presentation.pdf