File Virtualization with DirectNFS

Anupam Bhide, Anu Engineer, Anshuman Kanetkar, Aditya Kini
{anupam, anu, anshuman, aditya}@calsoftinc.com
CalSoft Private Limited, Pune 411 013, India
Tel: +91 20 567-4644
Fax: +91 20 567-7279
Christos Karamanolis, Dan Muntz, Zheng Zhang
{christos,dmuntz,zzhang,gary_thunquest}@hpl.hp.com
HP Research Labs
1501 Page Mill Road, Palo Alto CA 94304-1126
tel: +1 650 857-1501
Gary Thunquest
HP Colorado
{gary_thunquest}@hp.com

Abstract

There is a definite trend in the enterprise storage industry to move from Network Attached Storage (NAS) solutions to high performance Storage Area Networks (SAN). This transition is not easy because of the well-entrenched NAS infrastructure that has already been deployed. This paper attempts to define a file system that can leverage the existing NAS software infrastructure along with evolving SAN technology to provide the benefits of high performance storage access while reducing the cost of migrating to these networks.

In this paper, we propose a new network file system, DirectNFS, which allows NAS clients to take full advantage of the performance and scalability benefits of SANs. In order to achieve this goal, the system presents a NAS interface to existing NAS clients while allowing DirectNFS clients to access storage directly over shared SAN, i.e. clients bypass the server for data access. A server maintains the NAS interface for legacy clients and arbitrates access to metadata by DirectNFS (SAN aware) clients. This metadata server ensures that the system is operable for both legacy NAS clients as well as DirectNFS clients. The communication protocol of DirectNFS is designed as an extension of traditional network file systems protocols, such as NFS and CIFS.

A prototype of DirectNFS has been built for Linux, as an extension to the native NFSv2 implementation. Initial results demonstrate that the performance of data intensive operations such as read and write is comparable to that of local file systems, such as ext2.

1. Introduction

For the past few years, there has been an increasing trend to replace NAS storage systems by SAN. The primary reasons for this migration have been the increased data storage requirements that constantly plague the enterprise computing environment. SANs provide seamless expansion, combined with high throughput, and increased manageability. However, NAS architecture has been around for many years and has a well-entrenched installed base. The migration to SAN makes this NAS infrastructure obsolete and adds to the cost of already expensive SAN systems. One major drawback of the SAN systems that are deployed now is the lack of interoperability. However, this

situation will eventually be remedied as more users adopt SANs and as SAN standards evolve.
Today, with multiple operating systems and multiple vendor platforms present in most data centers, SAN inter-operability is highly valued. NAS technologies, on the other hand, are mature and interoperable. They use de-facto standards such as NFS[1] and CIFS[2] to provide data access. NFS clients are available for almost all platforms. Both NFS and CIFS have mechanisms to control and synchronize simultaneous access to shared data. These inherent features of NAS were taken advantage of in the design of DirectNFS.
A simple way of using the SAN, as shown in Figure 1, is to retain the familiar client/server model, with all the storage resources on the SAN appearing as local disks to the server. All the file accesses by clients in this scenario are forced to pass through the file server. This creates heavy loads on the file server.

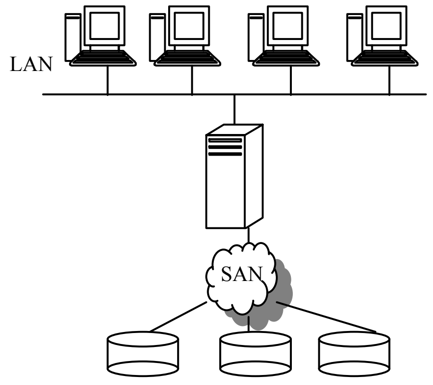

Figure 1: SAN with NAS Clients

In order to eliminate this overhead of data being copied through both SAN and LAN, the clients must be given the ability to access the data directly through SAN. To enable clients to access data directly, we have to provide them with a file location map that describes on which device and on which block the file data resides - information that is maintained as part of the metadata of the file system.

There have been different solutions to the distributed storage problem, ranging from "Shared Everything" to "Shared File Volume" architectures. In a "Shared Everything" filesystem, all clients maintain data as well as metadata portions of the file system. Most of the cluster file systems follow this approach (Petal /Frangipani[3], GFS[4]). In a "Shared File Volume" filesystem, one central entity is in charge of updating the data and metadata. Most client / server file systems follow this approach (NFS, CIFS). In a "Shared Everything" approach the implementation of the file system and its recovery on failure is complex. On the other hand, in a "Shared File Volume" approach, the scalability and performance of the file system are limited due to the existence of a single server. In the design of DirectNFS we have chosen to tread a middle ground between these two approaches. We have chosen to create a shared architecture for data, by making the clients aware of the physical layout of each file, which allows the clients to access data directly through the SAN. However, we do not allow clients to modify the metadata directly. Once we allow the clients to access data directly, the NAS-provided guarantees of single system semantics break down. This is unacceptable because a lack of single system semantics would lead to corruption of the file system. The solution is to create an entity that enforces these semantics, and this entity in DirectNFS is known as the metadata server. The metadata server is responsible for all metadata modifications in the file system. Since most filesystem metadata operations are atomic in nature, a single authority in charge of metadata modifications makes file system implementation and recovery easier. The metadata server also provides NAS interfaces to legacy clients for interoperability. This approach does have a drawback of introducing a single point of failure (metadata server) which makes the system less fault tolerant as compared to "shared everything" file systems. We believe that the potential gains from implementing a "shared everything" file system and making it compatible with legacy clients are not worth the complexity of the implementation.

DirectNFS clients are allowed to cache the block metadata, or the information pertaining to location of files. Coherency is enforced using a lease protocol. The metadata server acts as an arbitrator between the clients to make sure that the cached metadata is valid. The network architecture of DirectNFS is shown in Figure 2. By adding a SAN connection and DirectNFS software to each client, clients can utilize the file server for file system metadata access, locking, and coherency, but they read and write file data directly from the storage, bypassing the file server. The introduction of a simultaneous data access path can improve file serving performance through parallel and direct transfer of data between the data sources and the client systems. This also achieves better utilization of the file server by reducing the CPU and network load on the metadata server. Clients that either do not have a SAN connection or do not have the DirectNFS software can continue to access data through the server using the NFS or CIFS protocol clients, which they already have. This makes DirectNFS a powerful tool in migration of existing LAN/NAS combination to SAN.

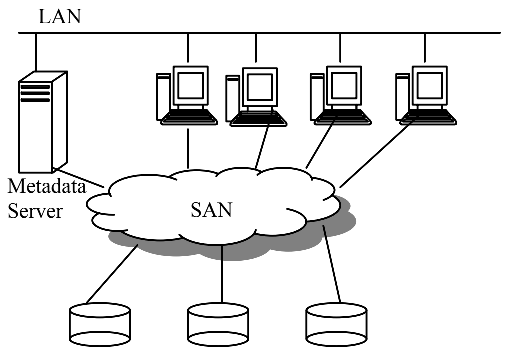

Figure 2: DirectNFS Network Architecture

We have implemented a GNU/Linux prototype of DirectNFS. Many platforms such as FreeBSD, Solaris and HP-UX were considered for reference implementation. GNU/Linux was chosen primarily because of the ease of source code availability, general acceptance in terms of usage and the support from the large community of hackers.

In our GNU/Linux prototype, we have demonstrated throughput comparable to that of a local (ext2) file system. Thus, we provide client applications the ability to have both shared file access and near local file system performance simultaneously. We have also observed lower server resource utilization in the metadata server compared to a NAS server, which implies that DirectNFS can support more clients than traditional NAS servers. DirectNFS implementation is transparent to applications running on the clients: no source code changes are necessary to client applications. During system operation, DirectNFS can be turned on or off without altering the file system semantics.
In this paper, in section two we talk about the goals associated with the DirectNFS design, section three talks about the design in detail. Section four of this paper deals with the Linux prototype. Section five discusses work done previously in this area. In section six, we highlight the performance achievements of DirectNFS. We present future directions for DirectNFS in section seven, and conclude in section eight.

2. DirectNFS Design Goals

In this section, we provide a list of design objectives of the DirectNFS architecture. In subsequent sections, we discuss the DirectNFS architecture in greater depth.

- *Storage Scalability* - Storage space must scale well with the continuous accumulation of data.
- *High Performance* - DirectNFS aims to provide a high performance remote file system, with orders of magnitude performance improvements over traditional NAS protocols.
- *File System Scalability and Recovery* - To create a simple distributed file system that can provide both scalability and recoverability.
- *Independence from Physical File Systems* - DirectNFS must be able to run irrespective of the underlying physical file system that is used for storage.
- *Portability* - DirectNFS should be portable to other Operating systems without much effort.
- *File Virtualization over SANs* - Enable the seamless integration of Storage Area Networks into NAS environments by adding a "File Virtualization" layer on top of the block-level interface that SANs provide.

3. Design

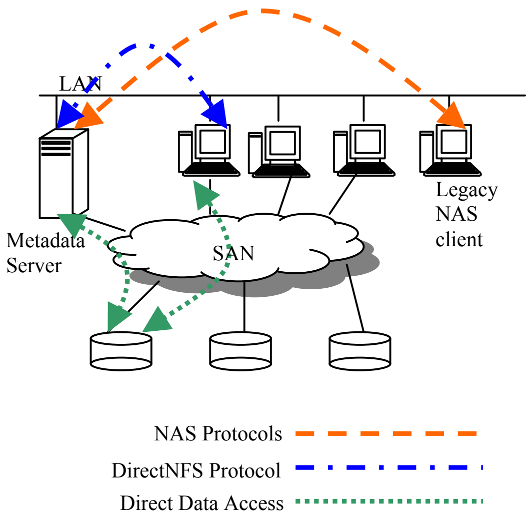

Figure 3: DirectNFS Architectural Overview

The basic philosophy behind the design of DirectNFS is the separation of data from metadata operations to increase parallelism in file system operations. Only read and write operations are taken over by DirectNFS client software, all the other file system operations are still performed through the NAS protocol. This makes DirectNFS design portable, thereby enabling us to use the same design on a host of other platforms including NT, BSD, Solaris and HP-UX.

The Figure 3 shows these operations more clearly, the communication between the DirectNFS client and metadata server. This communication includes lease protocol communication to maintain metadata coherency, the metadata information requests and NAS protocol functionality that is not intercepted by DirectNFS. The legacy NAS client communicates with the metadata server as if it were an ordinary NAS server.

3.1. Architecture Overview

This section provides an overview of DirectNFS architecture including DirectNFS extensions to the NFS protocol, cache coherency mechanisms, optimizations, and security.

3.1.1. Extensions to NFS

DirectNFS defines extensions to the NFS-RPC[5] protocol that implement the separation of the data/metadata path. This includes new RPCs used by the clients to retrieve the physical location of files on the storage (block lists) and additional RPCs to enforce cache coherency. The native RPC set of NFS is used to perform metadata operations on the server.

The new RPCs implemented by DirectNFS are,

- *GETBLKLIST* : This RPC allows the clients to get the block list of the files that are present in the system. The arguments to this RPC are the NFS file handle and the byte range for which the block list is requested.
- *GETLEASE* : This RPC is used by the DirectNFS client to acquire the lease for locally cached metadata. This RPC can be piggy backed on the GETBLKLIST RPC. The argument is the NFS file handle and duration. The reply sent by the server indicates whether the requested lease has been granted or denied.
- *VACATELEASE* : This RPC is used by the metadata server to ask a client to release the lease it has on certain file. The argument to this RPC is NFS file handle.
- *VACATEDLEASE* : This RPC is issued by the client, when it releases an lease due to the request from the metadata server.

Using these RPCs, clients are able to retrieve the physical locations of files and access them directly without conflict.

3.1.2. Metadata Caching and Cache Coherency

DirectNFS clients use extensions to the NAS RPC protocols to retrieve file metadata, i.e. physical block and device numbers. This file metadata is then cached locally on the client in a Block-Number Cache (BNC). This allows DirectNFS clients to cache the most frequently used physical block numbers for files that are most frequently used. However, introducing a distributed cache also introduces coherency issues, which we solve using a leases-based protocol.

A lease is a time-bound object granted by a lease server to a lease client. In DirectNFS, a lease is granted on a per-file basis to clients by the metadata server. The lease guarantees the client that as long as its lease is valid; it holds the most current copy of the data object (i.e. the cached list of blocks for the file). Multiple clients are allowed to share leases on the same data object for read-only access. However, any changes to this data by a third party can only be made when the server has revoked all other leases. This revocation is either done explicitly by notifying the client, or implicitly, if the leases time out. In either case, once the lease expires, the lease-holder has to discard the cached data protected by the lease.

The time-bound property of leases ensures simple recovery of clients/servers in case of a crash or network failure. Neither the client nor the server maintains any state. In case of a system crash, the leases that were issued before the system went down will expire, which brings the system to a known, stable state. This makes the recovery algorithm extremely simple to implement, especially when compared to the NLM protocol or other Distributed Lock Managers.

However, this coherency mechanism does not protect the system against SAN partitions, which may lead to data corruption – it is assumed that the SAN provides a reliable and available service for data delivery.

When the DirectNFS client needs to read/write a block of data, it first ensures that it has the right lease for the kind of access it needs to perform. The interaction between DirectNFS clients and metadata server for lease acquisition in write and read scenarios is illustrated in figures 4 and 5 respectively.

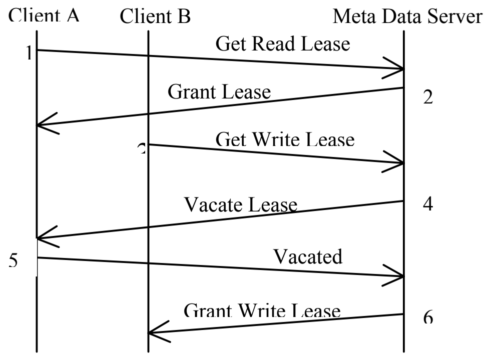

Figure 5: Sequence Diagram for Lease Protocol Interactions (Read-Write Conflict Case)

Once the lease has been validated, the client looks up the Block Number Cache for the physical location of the data. The metadata server is then queried for metadata information only in the event of a cache miss.

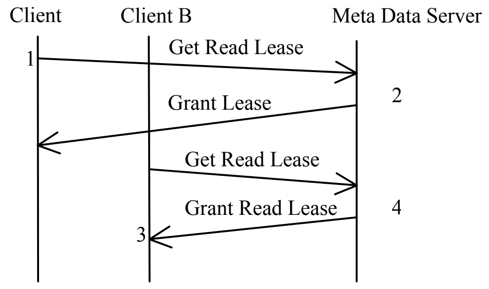

Figure 4: Sequence Diagram for Lease Protocol Interactions (Read-Sharing Case)

Metadata caching is augmented with "write allocation gathering". This is the process of deferring disk block allocations during file writes. In DirectNFS, we do write allocation by gathering write requests at the client. Smaller byte-range requests are merged into larger requests, thereby reducing the number of metadata requests to the server. This significantly improves performance, by reducing the number of requests to the server that the server has to service. "Write gathering" [6] performed by NFS is similar in its approach and it is used to exploit the fact that there are often several write requests for the same file presented to the server at about the same time.

3.1.3. Write Gathering

Distributed-system file access patterns have been measured many times[7]. It has been found that sequential access is the most common access pattern.

Under DirectNFS, for every write request, a cache miss would result in a GETBLKLIST RPC being sent to the metadata server. To improve write performance, a technique called write gathering is employed that exploits the fact that there are often several write requests for the same file called about the same time. With this technique the data portions of these writes are combined and a single metadata update is done that applies to them all. In this way, the number of RPCs being sent out would dramatically reduce, and considerably improve write performance.

The performance for write gathering depends on the periodicity of the deferred write requests to the server. Two events can trigger this: the write back cache being flushing periodically and an eviction notice received at the client.

3.1.4. File Virtualization

One of the major issues of merging SAN and NAS is the basic unit upon which they operate. The legacy NAS protocols operate at a "File" level abstraction. However, the SAN systems normally present the block level interfaces that are leveraged by filesystems.

In the DirectNFS design, we were faced with the problem of maintaining support for legacy clients, which meant that we needed to maintain the file level abstraction. On the

other hand, the benefits of the SAN can be leveraged if and only if we went down to the block level. In order to solve this problem we created a "virtualized file interface over SAN", where the legacy NAS clients are under the impression that the NAS server stores the files, but the DirectNFS clients went below the file abstractions to leverage the SAN performance by using block device interface directly. In order to implement this duality, we had to achieve the data-metadata split and create other mechanisms like the lease framework in order to tackle complexities arising out of the merger of SAN into NAS.
The DirectNFS file system had to merge these two different worldviews to create a high performance distributed file system, which offered a NAS interface. This was achieved by maintaining a "Virtual File Interface". However, the DirectNFS client behavior can be compared more to block device driver, than really a NAS file system client. In other words, we introduced the SAN abstractions and performance to the NAS protocols without breaking it. This unification of SAN of under NAS is what is referred to as file virtualization in DirectNFS.

3.1.5. Security Considerations

There are certain assumptions that are critical to DirectNFS architecture that need to be pointed out while understanding the security mechanisms in DirectNFS. They are

- The base NFS protocol operates on atomic data entities known as files.
- DirectNFS does not alter the semantics of NFS protocol
- DirectNFS relies on the file system and block device layer to provide security that is needed.

DirectNFS has modified the VFS layer[8] of NFS communication not the NFS semantics. The real physical file system must be present for DirectNFS to work. This is a strict requirement because we still rely on the file abstraction to maintain the coherency of data.
In DirectNFS, the file system layer is responsible for security and data coherency. In order to solve the coherency problem at file level, we have created a framework of leases ensuring that coherency is maintained at the file system level.

However, in case of rogue agents who can access the storage system at the block interface by bypassing DirectNFS completely, the possibility of unauthorized access remains, unless the block access mechanism (block device driver) provides security. We currently provide only file level security but do not provide block level security. NASD [9] addresses the issue of block level security with the help of special hardware. If the shared storage contains security mechanisms, for example iSCSI [10] has security mechanisms built in and when DirectNFS operates on those environments it can be made to run in a secure mode by leveraging these underlying mechanisms. Thus DirectNFS relies on existing infrastructure to take care of security (iSCSI, Fiber channel[11], NFS). This is a conscious design decision made in favor of making this protocol run on extremely varied range of hardware.

4. Implementation of the Linux Prototype

The implementation philosophy of DirectNFS was to reuse existing libraries as far as possible and to maintain portability. It was implemented as a kernel loadable module on Linux 2.4.4, and it consists of roughly 8000 lines of code on the client and 1500 lines of code on the server.

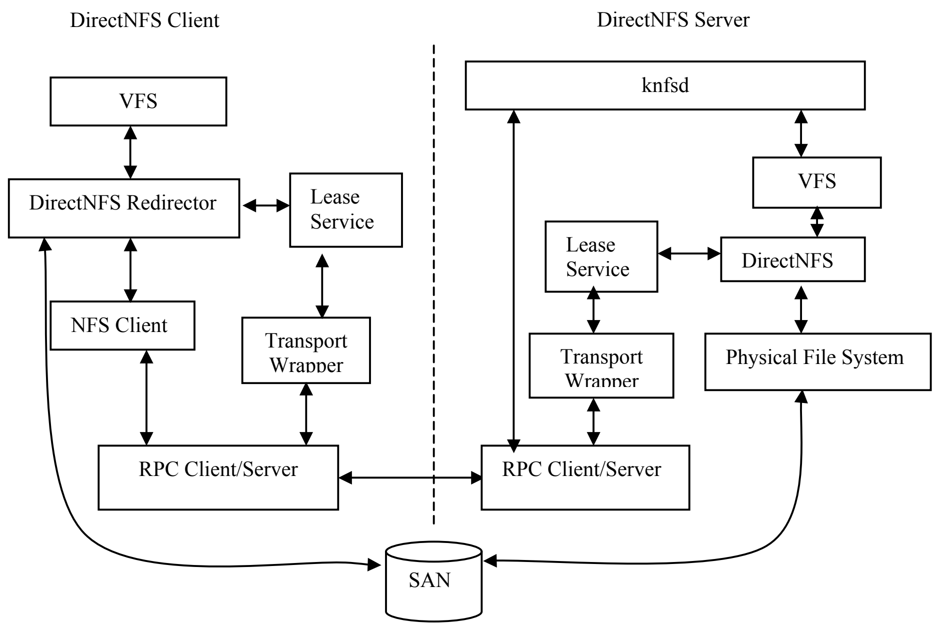

Figure 6: DirectNFS Software Architecture

4.1.DirectNFS with FiST

In order to make the implementation easier and portable we have used FiST (File System Translator). FiST [12] is a stackable file system generator. It defines its own highly abstract Domain-Specific Language (DSL) for describing file-system filters. A compiler translates the DSL description to C code for various operating systems. FiST also provides the necessary infrastructure for interposing the generated filter between the VFS (Virtual File System) and the natively installed file systems in the kernel. FiST played an important role in the initial phase of the implementation, when we used it to generate a code skeleton for a simple, pass-through file system that interposed itself between the VFS layer and the NFS client.

On the DirectNFS client, the Linux DirectNFS module can be thought of as consisting of these sub-modules:

1. **The DirectNFS Filter/Redirector** – This component interposes itself between the VFS and the NFS client module. It intercepts all file I/O operations (read, write) and redirects them as block I/O requests over the SAN. This was achieved by modifying the basic FiST-generated filter to enable us to intercept I/O operations instead of passing them down the file system stack, which is the default FiST policy. The I/O interception code in the redirector is system-dependent. The redirector also contains the Block Number Cache, where the client caches location information for each file that is accessed over DirectNFS.

2. **Leasing Service** – This is a distributed protocol, which allows multiple DirectNFS clients to keep their cached metadata coherent. The leasing service has been built as a library that is independent of the transport mechanism underneath it. This allows us to plug in any transport mechanism by writing a transport wrapper for the mechanism.
3. **Transport Wrapper** – This provides an interface between the leasing service and the transport layer, in this case - RPC. This wrapper allows the file system client to query file location information (i.e. block numbers) from a central server and to communicate lease requests to the server.

The DirectNFS server module consists of:

1. **Leasing Service** – This is the server-side counterpart of the leasing service. It is responsible for maintaining a list of lessees for each file, and to resolve lease conflicts.
2. **Transport Wrapper** – The transport wrapper on the server as on the client provides an interface between the leasing service and the transport layer. This wrapper allows the server to interface with file system clients that query for file location information and to communicate lease rejections or grants to them.
3. **DirectNFS client** – A DirectNFS client is interposed between VFS and the physical file system, to provide lease-based coherency for locally originating file accesses. This could be from local applications trying to access the physical file system or from knfsd while it is serving legacy NAS clients.

The DirectNFS module on the client is responsible for trapping file open, close, sync, unlink, read, and write calls. Since these operations access the location information of the file, the file's lease is tested for validity. If the lease is invalid, it is acquired by issuing a GETLEASE RPC to the metadata server. For read and write operations, the Block Number Cache is looked up for cached block numbers. On a cache miss, a request is sent to the server, with a piggybacked lease request, if required. This is done with the GETBLKLIST RPC. Once the client is granted a valid lease on the file, and receives the requisite file location information, it accesses those blocks directly over the SAN.

In the event that the client receives a VACATE RPC, which signals the server ordering an eviction of the lease that the client holds on the metadata, the client flushes the cache that is associated with the file, and then proceeds to inform the DirectNFS server by sending the VACATED RPC.

Note that the DirectNFS Leasing Service makes the following assumptions:

1. The lease is time-bound, has a fixed duration, and must be renewed explicitly at the server in order for its time period to be extended.
2. The clock skew between the participating entities in the lease protocol is bounded.
3. The time taken by the client to flush its cached after eviction is bounded.

Lease conflicts are resolved by the lease server using the matrix in Table 1.

	Read	Write
Read	Shareable	Non Shareable
Write	Non Shareable	Non Shareable

Table 1: Compatibility Matrix for DirectNFS Leases

5. Performance

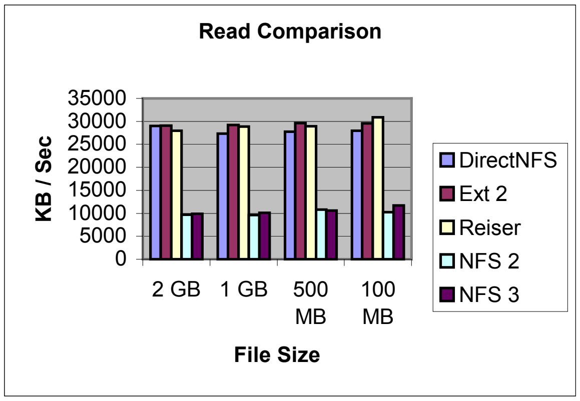

Figure 8: Read Comparison

One of the principal objectives of DirectNFS is performance. In this section, we present the performance numbers that we obtained from the prototype implementation. We have measured the performance of DirectNFS against other file systems like ReiserFS[13], ext2 and NFS versions 2 and 3[14]. The systems under test were three HP Netserver LC 2000, Pentium III's -933 Mhz with 128 MB RAM and 256KB L2 cache. The machines were running Redhat Linux, with custom-built kernels from the 2.4.x series. They were connected to a JBOD (HP Rack Storage/12) of four Ultra 3 Hot-Swap SCSI[15] disks 9 GB each. The system was set up in a SCSI multi-initiator arrangement, with two machines acting as DirectNFS clients, and one machine as the DirectNFS metadata server, with all three machines sharing access to the JBOD through a shared SCSI bus. This was used to emulate a SAN. The benchmarking utility that we used was Iozone [16].

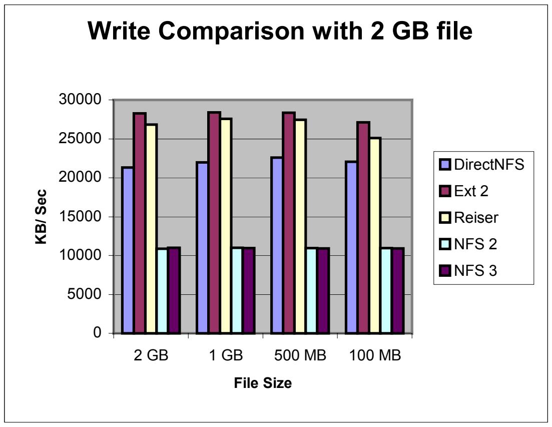

Figure 7: Write Comparison

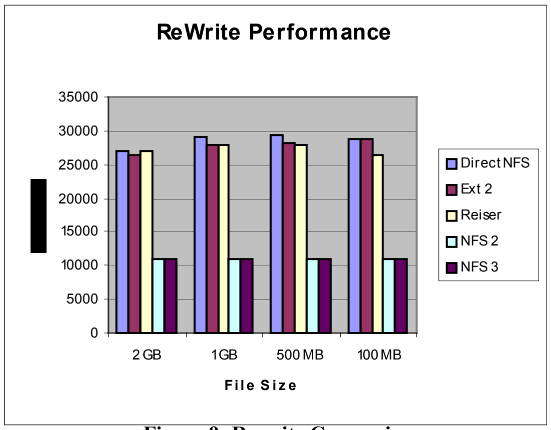

Figure 9: Rewrite Comparison

We benchmarked the performance of DirectNFS with varying file sizes and record sizes. From the data, we observed no significant variations in the comparative figures. Hence, we have included the performance figures of read, write, reread and rewrite of a 2GB file over ResierFS, DirectNFS, Ext2, NFS2 and NFS 3. Figure 7 is a the performance graph of various file system read throughputs for varying file sizes, with fixed record size of 256 KB. The rest of the graphs - Figures 8, 9 and 10 - carry comparisons of write, re-read and re-write operations. These figures indicate that DirectNFS performances are comparable to local file systems.

The write performance of DirectNFS shown in Figure 8 is slightly worse than Ext2 and ReiserFS. Re-read and re-write were tested so that we could measure the effects of the Linux page cache.

We have measured throughput for these four operations with varying file sizes starting from 100 MB up to 2GB and varying record sizes starting from 4 KB up to 256 KB. Since the throughput figures we obtain did not vary significantly across these series, we reproduce data for 256KB record sizes only. The file sizes selected were suitable large, as we expect the primary use of DirectNFS to be multimedia applications (e.g. streaming media servers), which use large files.

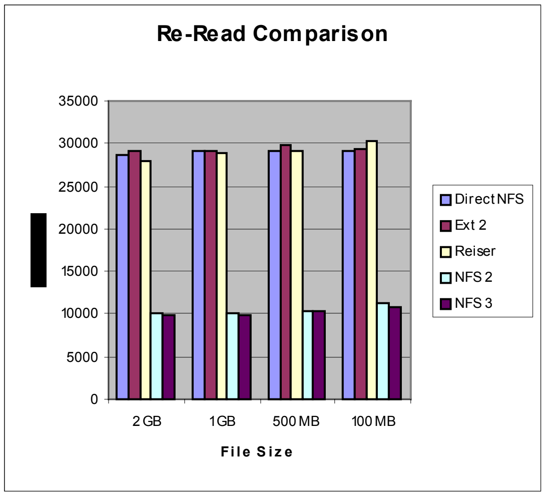

Figure 10: Re-Read Comparison

Note that NFS v2 and v3 throughput figures that we measured were very close to each other. Even though NFS 3 implements Asynchronous writes, NFS 2 clients under the Linux use write caching and by default run with synchronous writes set to off. This hides the RPC latency of NFS from client applications. However, we wanted to compare against real world performance and hence we tried to measure against the fastest NFS performance possible.

From a glance at the throughputs for read and re-write tests, it appears that DirectNFS performance comes close to matching the performance of both ReiserFS as well as ext2. This can be accounted for by the metadata cache, which contains logical to physical block translations, and improves the performance of DirectNFS, bringing it close to ext2 and in some cases surpassing it (this is because the mapping function for the cache is less expensive than the corresponding lookup operation in EXT2 or ReiserFS). We also examined the effect of record size on performance. Figure 11 is a comparative graph for the read operation for various file systems with fixed file size but with varying record size. We did not observe any significant effect of record size on throughput of any of the file systems under consideration. This is most likely due to the pre-fetching in the VFS layer.

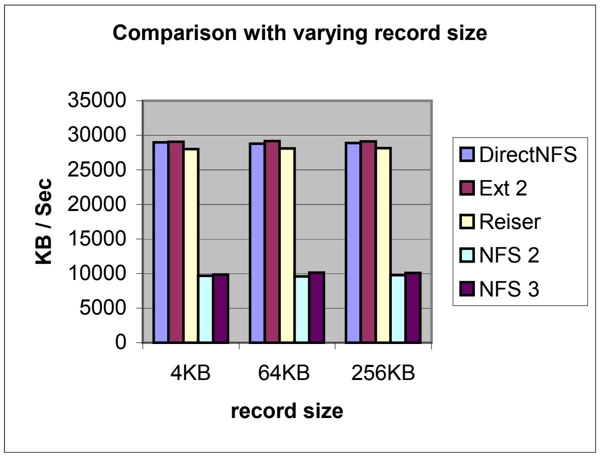

Figure 11: Comparison with varying record size

If we look closely at the performance relative to NFS2 or NFS3, we see that the performance improvements that are achieved are significant, and are 2 to 3 times that of the Linux implementation of NFS.

There are two measures of goodness for a network file system, the first is the throughput that each client can expect from the file system, and the second is the server scalability. DirectNFS

addresses both of them by increasing the client throughput by a factor of 2 to 3 as compared with competing NAS technologies like NFS, and increases the server scalability significantly by reducing CPU utilization at the server.
A look at Figure 12 shows the relative CPU utilization of DirectNFS with NFS. The tests that were carried out were sequential read, sequential reread, sequential write, and sequential rewrite. Now, if we look at the NFS performance, we can conclude that NFS (with a single client running Iozone tests on a file of size 1GB) requires a mean CPU utilization of more than 20%. Thus, the scalability of the server is limited to the number of clients that access the NFS server at any point of time.
However, a look at the DirectNFS numbers for the same test conditions shows a radically different scenario. One can see that there is an initial period where the CPU utilization is roughly at an average of 10%, with a peak utilization of 20%. This is because of aggressive pre-fetching of metadata by the DirectNFS client during the start of file I/O. This accounts for the lower CPU utilization on the server when servicing a DirectNFS client as compared to a NFS client.

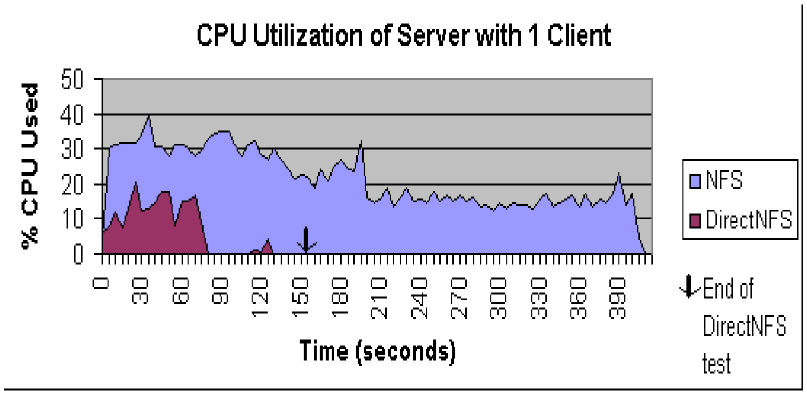

Figure 12: CPU utilization figures for a single client setup

Thus, it can be seen that the CPU utilization is significantly lower than NFS utilization for the same one client setup that we used to measure NFS utilization. This indicates that the DirectNFS metadata server may scale better than NFS servers.
Another key parameter by which scalability can be judged is the amount of network traffic, expressed in terms of the number of RPCs that are required for a given operation to take place. A measurement of the number of RPCs that are required to run the given set of tests reveals that DirectNFS uses about a tenth of the total number that is required for NFS. This can be explained by the fact that the number of metadata requests in DirectNFS is drastically lower than NFS because of write allocation gathering and the metadata pre-fetching performed by the client. This makes the data-metadata split attractive, as this considerably reduces the traffic on the network and makes DirectNFS a lot more scalable.
Overall, DirectNFS performs significantly better than NFS for all of the tests, outperforming it by a factor of 2 to 3.
DirectNFS has been designed to counter network bottlenecks and ‘store-and-forward’ overheads on NAS servers. So, the server CPU and I/O subsystem are no longer the bottleneck. Introducing parallelism to storage access also means that the system will scale as the available bandwidth for the storage network increases. Isolating storage traffic on to a separate network allows for better utilization of the messaging network by

other network application protocols.

6. Future Work

1. **Client Side Disk Caching:** To further improve performance, the size of the cache that holds the physical block translations should be made as large as possible. To overcome the memory size limitations that we will come across when dealing with large files and clients with multiple such workloads, the block translations can be stored on disk. Thus, the limitation that currently exists on the number of cacheable translations increases greatly, helping us to achieve greater scalability.
2. **Volume metadata caching:** When the metadata server receives a GETBLKLIST request, the DirectNFS filter uses the physical file system's *bmap* operation to obtain the physical block numbers for the requested byte range. Normally, the block buffer cache would cache the most frequently used blocks in the storage system. Servers normally have a large amount of RAM, and we feel that caching the entire metadata for the file volume is feasible. In fact, for a file system formatted with 4KB-sized blocks, the cost of caching all the physical block numbers of the volume is about 1MB per GB.

7. Related Work

There are some interesting existing systems in the distributed File Systems space. Storage Tank [17] follows a similar approach for moving the data access path away from the server. However, the design of Storage Tank lacks the portability of DirectNFS. This is because DirectNFS uses a portable approach leveraging the ability of a code generator like FiST to drastically reduce the porting of the file system to multiple platforms. Many cluster file systems such as the Veritas Cluster File System [18] are layered above and integrated with a proprietary physical file system. CMU's Network Attached Secure Disks requires Intelligent Devices, which embed some file system functionality in the Storage devices thus handling various issues like security, scalability and object management. NASD addresses the security aspects of a SAN based file system well, but the need for manufacturers to incorporate these changes into disks highlights the problem associated with this approach.

Other similar work in the area includes Frangipani/Petal, Tivoli's SANergy [19] and EMC's Celerra[20].

8. Conclusion

DirectNFS presents an optimum blend of NAS and SAN storage technologies. It uses traditional distributed file system protocols such as NFS for meta-data access, with extensions for direct data access using SANs. The end result is a distributed file system that scales much better at high loads and has a data throughput that is a factor of 2 to 3 better than existing NAS protocols. In fact, this performance was comparable to that of a local file system.

The portable design of DirectNFS makes it relatively simple to port to other operating systems. In the future, we plan to port DirectNFS to other platforms such as HP-UX, Windows2000 and FreeBSD and add CIFS compatibility.

Acknowledgments
We would like to take this opportunity to thank Anandamoy Roychowdhary, who played an important role in both the design as well as the implementation of Direct NFS.
We are grateful to Sunu Engineer, who helped with the design.
We would also like to thank Alban Kit Kupar War Lyndem, Tanay Tayal and Gurbir Singh Dhaliwal who helped with the implementation.

References

[1] Sun Microsystems, NFS: Network File System Protocol Specification, *RFC 1094*, 1988.
[2] P. J. Leach, A common Internet file system (CIFS/1.0) protocol*, Technical report, Network Working Group, Internet Engineering Task Force*, December 1997.
[3] C. A. Thekkath, T. Mann, and E. K. Lee., Frangipani: A Scalable Distributed File System., *In Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Oct. 1997.
[4] Kenneth W. Preslan, A 64-bit, Shared Disk File System for Linux*, Proceedings of the Sixteenth IEEE Mass Storage Systems Symposium held jointly with the Seventh NASA Goddard Conference on Mass Storage Systems & Technologies*, 1999
[5] Sun Microsystems., Open Network Computer: RPC Programming., *The official documentation for Sun RPC and XDR.IBM Inc.*
[6] Chet Juszczak, Improving the Write Performance of an NFS Server (1994), *Proceedings of the USENIX Winter 1994 Technical Conference,* 1994
[7] M.G. Baker, J.H. Hartman, M.D. Kupfer, K.W. Shirriff, and J.K. Ousterhout. Measurements of a distributed file system., *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles. pages 198-212*, 1991
[8] D. S. H. Rosenthal., Requirements for a "Stacking" Vnode/VFS Interface", *UNIX International*, 1992
[9] G. Gibson et al., File Serving Scaling with Network-Attached Secure Disks, *Proceedings of the ACM Int. Conf. on Measurements and Modeling of Computer Systems (SIGMETRICs `97), Seattle, WA*, June 15-18, 1997.
[10] Y. Klein and E. Felstaine., Internet draft of iSCSI security protocol. *http://www.eng.tau.ac.il/~klein/ietf/ietf-kleiniscsi -security-00.txt*, July 2000
[11] ANSI, Fiber Channel Transmission Protocol (FC-1), *ANSI draft standard X3T9.3/90-023, REV 1.4,* July 6, 1990.
[12] Erez Zadok, FiST: A System for Stackable File System Code Generation*, PhD thesis. Columbia University*, May 2001.
[13] NameSys Inc., The ResierFS file system, http://www.resierfs.org, 2001
[14] B. Callaghan, B. Pawlowski and P. Staubach, NFS v3 Protocol Specification, *RFC 1813*, June 1995.
[15] ANSI, SCSI-3 Fast-20 Parallel Interface, *X3T10/1047D Working Group, Revision 6.*

[16] W. Norcutt, The IOZone file system benchmark, *Available from* *http://www.iozone.org/*, April 2000
[17] Storage Tank Software*, http://www.ibm.com/*, 2000
[18] Veritas Inc. Veritas Cluster File System, *http://www.veritas.com*, 2001
[19] Mercury Computer Systems Inc., High Speed Data Sharing among Multiple Computer Platforms, *http://www.sanergy.com,* 2001
[20] EMC Corporation, Celerra, *http://www.emc.com*, 2001