FICUS: A File System for Inter Cluster Unified Storage

Arun Ramakrishnan and Mario Lauria
Dept. of Computer and Information Science
Ohio State University, OH 43210
(ramakria,lauria)@cis.ohio-state.edu

Abstract

There are no high performance file systems that allow sharing of data between different clusters presently. In this paper, we address this issue by developing FICUS (File System for Inter Cluster Unified Storage). FICUS provides the convenience of file system level access to remote data while preserving the performance of striped file systems such as PVFS. We achieve a bandwidth of 80 MB/s for local access using four I/O nodes, and 75 MB/s in accessing the same number of I/O nodes on a remote storage cluster. A parallel data intensive application achieves comparable performance when accessing 6 GB of data in local and remote storage. We show that careful pipelining of data transfer and proper integration with underlying file system and communication layers are crucial for preserving the performance of remote access.

1.Introduction

The advances in inter-disciplinary sciences and technology trends have resulted in data-intensive applications coming to the forefront of the high performance computing (HPC) community. A rapidly growing number of applications in domains such as genomics/proteomics [3, 7, 9, 13, 15], astrophysics [4], physics [1], computational neuroscience [21], or volume rendering [2], need to archive, retrieve, and process increasingly large datasets. Therefore, the storage of large amounts of data has become a primary concern in the design of architectures for HPC.

At the same time, clusters of PCs have become a popular platform for high performance computation at all institutional levels – from a single research group to a supercomputer center or a government laboratory. It is the advent of "killer micros" and "killer networks" that has allowed clustered commodity machines to compete with supercomputers in aggregate performance. However not all areas of the cluster technology have enjoyed the same dramatic advancements – one aspect that has lagged behind is the performance of the I/O subsystem.

Early models of disk storage organization are still being used today and have not been significantly improved over the years. Two models account for most of the I/O system organization found on contemporary clusters. In the simpler approach, the Networked File System (NFS) is used to export shared file systems from one or more file servers to all the nodes, on each cluster node a local disk is used to store the operating system and for temporary local storage. Since today distributed file system services are integral part of every operating system, such scheme has the benefits of immediate installation and simple

management. The trade-offs are inadequate performance for parallel disk access due to the small client-to-servers ratio, and a severely limited scalability due to the single server NFS model. NAS (Network Attached Storage) boxes have tried to speed up NFS using proprietary disk controller technology and TOE (TCP Offload Engine) NICs. In spite of all these improvements, NFS is still not adequate for parallel access patterns.

In large scale installations, a large number of disks are located in a "storage island" physically separated from the computing cluster, and connected to it by a fast network. In one approach, the island consists of disks mounted in racks, and directly connected to a Storage Area Network (SAN). In another approach, the island consists of a dedicated storage cluster, with disks distributed across all its nodes, typically connected to the computing cluster through a fast LAN such as Gigabit Ethernet. Both approaches ensure that any node in the compute cluster can access any disk in the storage island. However, performance is still limited to single disk access bandwidth when using traditional file systems. The larger aggregate bandwidth available at the physical level can be made available to applications only through the use of parallel I/O libraries or a recoding of applications to explicitly enable concurrent multiple disk accesses.

The solution we envision is based on a striped file system that not only spans all the nodes in the cluster, but also has the ability to export directories to remote clusters. The export of file systems is done in a way that preserves striping while accessing the remote storage. Our file system maintains the familiar Unix I/O interface, and is compatible with the existing Linux VFS specifications.

FICUS is an extension of the basic PVFS design. While PVFS stripes each file on designated I/O nodes within the cluster (in the following called I/O servers), FICUS adds the possibility of designating remote nodes as I/O servers. By distributing the load over a sufficient number of either local disks or storage cluster nodes, the performance bottleneck issue is mitigated. We believe that cluster-wide striping is the most performance effective approach to make these aggregate resources available to the applications. The reasons for striping are still valid outside the cluster, and therefore the same techniques can be used to achieve high performance access to remote storage. While the notion of distributed file systems is not new, preserving the striping for the remote access is.

The main contributions of this paper are the design of a distributed file system that preserves the performance advantage of the striping across cluster boundaries. The main design issues we had to solve were how to implement our design on top of stock file system and communication layers, and how to preserve performance when large amounts of data need to traverse several hops and processing stages.

The paper is organized as follows: In Section 2, we provide a brief overview of PVFS. The FICUS file system is introduced in Section 3. The evaluation results are presented in Section 4. We present some related work in this area in Section 5. New directions for future work and our conclusions are presented in Section 6.

2. Parallel Virtual File System(PVFS) Overview

PVFS is an open source cluster file system developed for the Linux operating system [10, 16]. It uses an out-of-band approach, where file metadata is handled separately from the actual file data. PVFS also does data striping (also known as RAID 0) to achieve good scalability and performance in a cluster.

There are four main components in PVFS:

- **Metadata manager** : This handles all the metadata pertaining to a file like its owner, creation time, permissions etc. In addition, it also has knowledge about the manner in which the file is striped and the stripe size involved. It is located outside the main data pathway and thus doesn't become a performance bottleneck.
- **I/O Servers** : These serve as the main handlers of file data. The file data is typically striped across a set of I/O servers for maximizing bandwidth and minimizing network bottlenecks. These usually memory map(mmap) frequently accessed files and also send file data using the sendfile interface. These also have transaction queues for handling simultaneous requests effectively.
- **User Level Access Library**: Clients use this in order to access the PVFS file system directly from user space. This minimizes the amount of kernel overhead and also simplifies the task of building new applications on top of this file system. This enables the user to control striping of files and also facilitates easy integration with existing I/O file systems like ROMIO.
- **Kernel Module** : This is the most critical component for enabling a VFS (Virtual File System) interface to PVFS. This enables existing applications to be run on top of PVFS without any modification (by using the standard UNIX I/O interface), while using the high bandwidth bank of I/O servers. The familiar UNIX file tools (ls,chmod,rm, touch etc) can also be used on PVFS files and directories without modification.

The PVFS file system also has a 64-bit interface by means of which huge files (bigger than 2 GB) can be created and accessed. The convenience of using PVFS also stems from the fact that the I/O servers use the local file system for holding the striped file data. This means that they can automatically take advantage of improvements in local file systems. So we could run PVFS over the XFS file system for better reliability and performance. Thus PVFS is a viable option for boosting the I/O bandwidth within a cluster. The PVFS file system also serves as a good reference for evaluating inter-cluster I/O approaches. We would discuss the relevance of PVFS in our file system in the following sections.

3. FICUS: A File System for Inter Cluster Unified Storage

In this section, we will elaborate on the design and functionality of the FICUS file system. While developing this file system, we gave utmost priority to the crucial aspects of performance and ease of accessibility. We will also try to establish the stability and viability of this file system, by running an actual application (parallel data mining) on top of it.

3.1 Limitations of cluster file systems

FTP is still commonly used to transfer data manually across clusters before starting the application. This is clearly an inconvenient approach particularly for data intensive applications. Ideally the application should be able to instantly access the data as if it were located on a local file system.

In principle one could achieve inter-cluster I/O access by first running PVFS internally in a cluster and then enabling port forwarding to the metadata and I/O servers from the gateway node. This would allow an external cluster to access the PVFS file system directly. However this approach has the following limitations :

- *PVFS servers normally run as root*. Thus it would be risky to expose these ports to an external network directly as it may lead to a root compromise easily. We also have to expose a large number of internal machines when we have a large bank of I/O servers.
- *The data pathway may not be uniform*. The bandwidth of the interconnect inside a cluster may be much higher than the inter-cluster link bandwidth. This essentially leads to a non-uniform data pathway with bottlenecks at the edges. This cannot be effectively managed using the stock PVFS implementation.
- *Access control* : PVFS was designed to allow data transfer between trusted hosts. But when we have clients sitting on a remote cluster, we may want to selectively control access to the file system. This can't be done specifically for the file system alone on the gateway nodes.

3.2 FICUS basic design

We present FICUS as a viable inter-cluster file system here. It is designed for dealing with the issues discussed in section 3.1 from ground up. This file system uses the internal PVFS file system in a cluster to enable high performance inter-cluster I/O.

Its main components are:

- **Metadata proxy** : This acts as a protocol proxy, which translates external metadata requests to internal PVFS metadata requests. It presents itself as a metadata server to a client in an external cluster and then internally uses the PVFS metadata server for its operations. The PVFS metadata server is not exposed to the external network at all. Also the proxy runs as a non-root user and doesn't need root privileges. It also prevents corruption to existing data in the PVFS file system because the PVFS metadata server is responsible for allocation and destruction of file inodes. It is also responsible for boot strapping the entire file system by supplying the mapping information to the FICUS information server.
- **I/O proxy** : This handles the data requests from external clients and then fetches the data from the appropriate I/O servers. It acts like a PVFS I/O server to an external cluster and then stripes the data using a bank of I/O servers located internally. Here

too the PVFS I/O servers are not exposed externally.

- **Information Server** : This allows an administrator to control access to the file system. The administrator can selectively allow/disallow a remote client using ACL-like entries. Access can also be granted at a domain or subnet granularity. It also serves as a repository for maintaining the mapping between the proxies and the respective PVFS servers i.e. which I/O proxy is mapped to which particular I/O server et al. This enables us to maintain minimum state information on the proxies.

The entire FICUS file system is designed to be totally transparent to both the remote client as well the local PVFS setup. FICUS appears to be a normal PVFS file-system to an external cluster. The internal PVFS file system thinks that it is getting requests from a normal client located within the cluster. This transparency ensures that minimum amount of changes are needed for exporting and accessing data remotely.

3.3 Bootstrapping the FICUS file system

The very first step involved in installing FICUS is to setup the PVFS file system in one of the clusters, which would act as the data repository. Then the FICUS information server is started with the appropriate ACL entries enabling access selectively to specific remote clusters/clients. The FICUS information server can also add permission entries dynamically upon receipt of a special user-defined signal. This allows us to add more clients to a running FICUS file system. Then a file specifying the following mappings is prepared :

1. *Metadata mapping* : This specifies the ports and location of the FICUS metadata proxy and the PVFS metadata server.
2. *I/O mapping* : This specifies the ports and location of the FICUS I/O proxy and the PVFS I/O server. It also specifies which I/O servers are used by which I/O proxy(and they become a part of its stripe set).
3. *Mount point mapping* : This specifies the mount point of the FICUS file system.

FICUS has the capability to accommodate multiple exported file systems. The above mapping information is a complete description of each FICUS exported file system and hence is attached to the metadata proxy of that file system. The Information server acts as a central repository of information about all the exported FICUS file systems. When the I/O proxies are started (giving only the location of the FICUS information server), they in turn contact the FICUS Information Server in order to obtain their stripe set (the set of PVFS I/O servers under their management). The above design enables us to manually specify only the location of the FICUS information server to the remote client, which can then gather the remaining proxy information automatically. The design also let us use a single information server for handling multiple FICUS file systems, as all the information is hashed based on the mount point.

The remote client is started specifying the location of the remote FICUS information server and the desired FICUS mount point. The

information server then compares the client address against its ACL entries in order to ensure that the remote client/cluster can be allowed to access the FICUS file system. If an appropriate ACL entry is present, then the addresses of the metadata and I/O proxies are sent to the client. The client thinks that it is contacting a conventional PVFS setup and thus has no knowledge about the actual PVFS servers in the remote cluster. This allows us to hide critical internal PVFS information while still allowing access to the file system.

4. Performance Evaluation

In this section, we test the performance of FICUS by running some synthetic applications generating data streams. We also test the stability of this file system by running a parallel incremental data-mining algorithm performing 2D-Discretization on dynamic data sets, with the data located at a remote location.

4.1. Experimental Testbed

Our testbed consists of a cluster of 9 dual 1 GHz Pentium-III servers running Linux 2.4 operating system, with 1 GB of SDRAM each. These machines were interconnected using a 1.2 Gb/s Myrinet network and we were running TCP/IP over Myrinet. The cluster also had a fast Ethernet network, which was used for carrying metadata traffic. Each machine has two 60 GB of disks mounted on a 3Ware controller in a RAID 0 (striped) configuration, yielding 120 GB of usable partition space. We installed the XFS Linux file system on these partitions for better data reliability and performance. The cluster was logically partitioned into two halves so that one half could access the other half only through designated gateway nodes. This helped us simulate two clusters within a single cluster. Then we installed a PVFS file system on one half with one metadata server (accessible through the Ethernet network) and 4 I/O servers (accessible through the Myrinet network).

It is important to note that both PVFS and FICUS use TCP for data communication. We measured a peak bandwidth of around 45 MB/s using TCP over Myrinet.

4.2. Local Disk Bandwidth

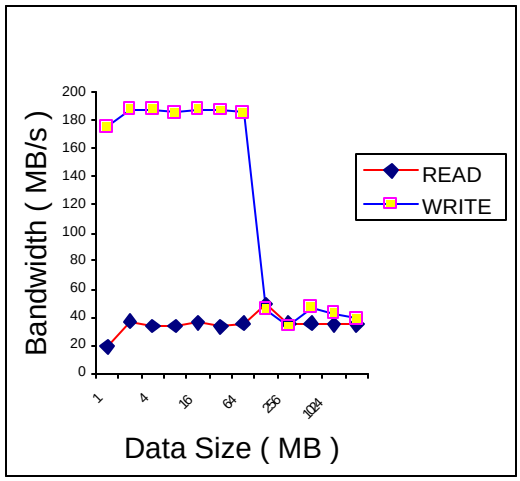

Figure 1. Local file system performance

The above figure shows that for small files sizes, there is lot of caching involved at the file system level. The 3Ware controllers have an internal write buffer, which makes writes faster than reads. The aggregate bandwidth for the entire partition is around 40 MB/s due to the usage of disk striping (RAID 0). The XFS file system on this partition also helps the writes by means of delayed writes and extent based allocation techniques.

4.3. PVFS performance

We measured the performance of the PVFS file system file system with 4

I/O servers and varying number of clients. In all the following figures, the I/O bandwidth measured is the aggregate bandwidth across all the nodes.

We observe in Fig. 2., the file system cache plays a crucial role in attaining high write bandwidth for small files. In the case of larger files, the 2 and 4 client cases are able to utilize the bandwidth available from striping the data across 4 I/O servers fully and thus exhibit better performance when compared to the single client scenario. The bandwidth of the 4-client case drops sharply after data sizes of 64 MB. Each I/O server is not able to handle the load of writing four concurrent data streams to disk and thus the limitation of the write pathway and disk bandwidth are clearly exposed.

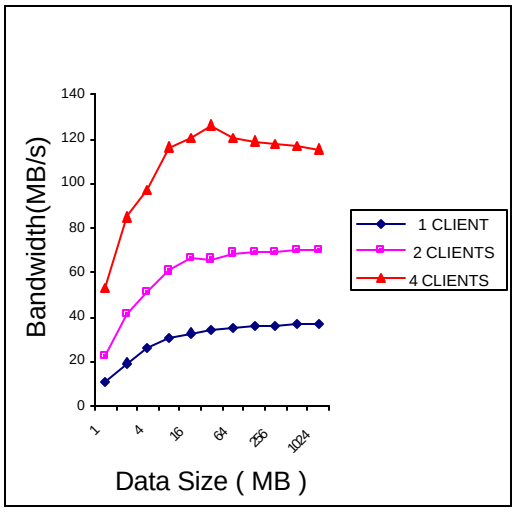

Figure 2. PVFS Read performance

The read scenario shows much better scalability up to four clients because the PVFS I/O server memory-maps the file (using mmap) after the first read request arrives and then serves the remaining read requests from memory itself.

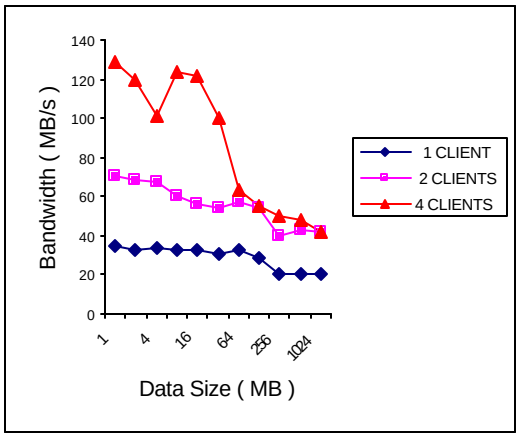

Figure 3. PVFS Write performance

Thus, the read pathway of PVFS is efficient. A larger number of clients is able to fully utilize the available read bandwidth and thus the 4-client case performs much better than the other two scenarios, as shown in Fig.3.

4.4 FICUS performance.

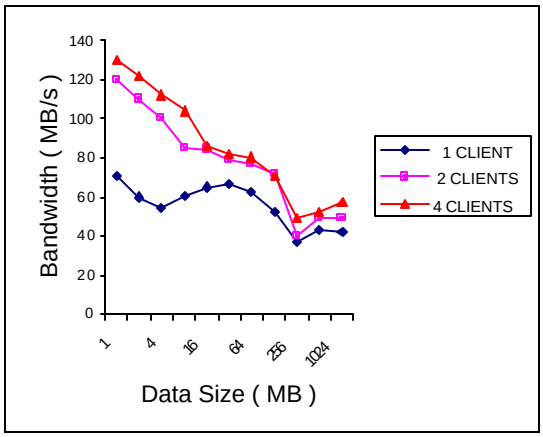

Figure 4. FICUS Write performance

The most important design aspect of the FICUS I/O proxy is its use of threads for servicing the I/O requests. We used pthreads and were able to get asynchronous I/O behavior on reads and writes. This has been very crucial in the performance of FICUS as illustrated in Fig. 4. and Fig.5.

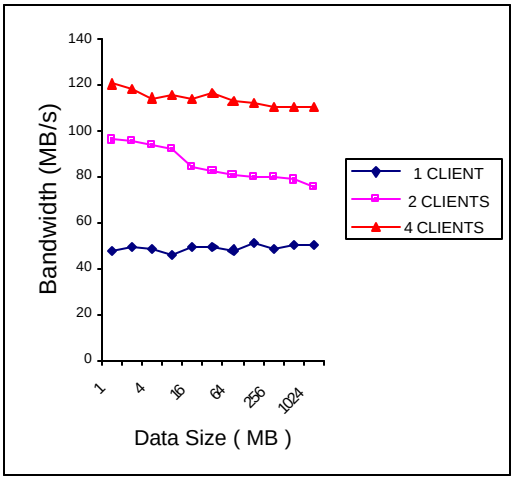

Figure 5. FICUS Read performance

Here we ran 2 I/O proxies, each of which striped the data across 2 I/O servers (yielding a total of 4 I/O servers for the entire setup). In Fig. 4, we see that the FICUS write performance is slightly better than the PVFS write performance, in the case of 4 clients. The important point to note here is that the PVFS I/O servers are single threaded applications. This proves that FICUS is able to handle high client load without great degradation in performance. This fact is further clarified in figure 5. The read performance for all cases reaches the peak value for comparatively small reads. The clients need not perform large I/O requests in order to attain the full bandwidth of the file system. This behavior is in contrast to the original PVFS behavior, where the read performance reached the peak value only from 32 MB reads onwards. The above two figures illustrate that FICUS is able to handle heavy I/O loads as well. We can also observe that a larger number of clients will not get better performance because four clients already saturate the I/O bandwidth available.

4.5 Multithreading and Pipelining

We mentioned in the previous section that the FICUS I/O proxy is a multi-threaded application. This allows FICUS not only to achieve asynchrony in the main I/O pathway, but also allows us to pipeline the I/O requests effectively. The data pathway in the FICUS file system consists of two main stages :

1. The stage with the remote client and the FICUS I/O proxy. This stage includes the external link between the two clusters.
2. The stage with the I/O proxy and the PVFS I/O server. This stage includes the internal network in the cluster(Myrinet in our case).

The two stages are not equally balanced because typically stage 2 has much more bandwidth available than stage 1. The use of threads allows us to pipeline data across these two stages effectively, because while one thread is servicing an I/O request from a client, another thread could be getting data from the I/O server and the other thread could be writing data to a third client. This means that at a single point of time, both stages could be kept busy with the use of threads, yielding a big performance boost, as shown in [22] and [23].

We ran the I/O proxies on a dual processor machine. The figures(Fig. 6 and Fig. 7) show that threading yields a performance boost when compared to the non-threaded case. These figures also reveal that the number of threads should be equal to the number of physical processors available, for obtaining peak performance. We observe that 4 threads do not yield any great benefit over 2 threads in this experiment. We also observe that the clients can attain peak bandwidth faster by the use of threads (as shown in Fig.6), because we are able to keep both stages of the data pathway busy with pipelined requests.

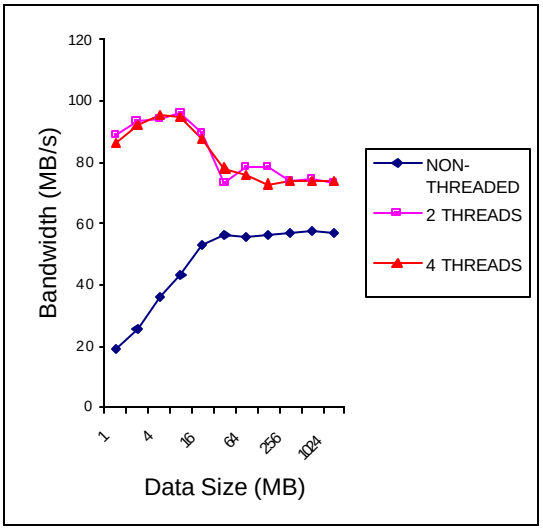

Figure 6. Impact of threading on Write performance

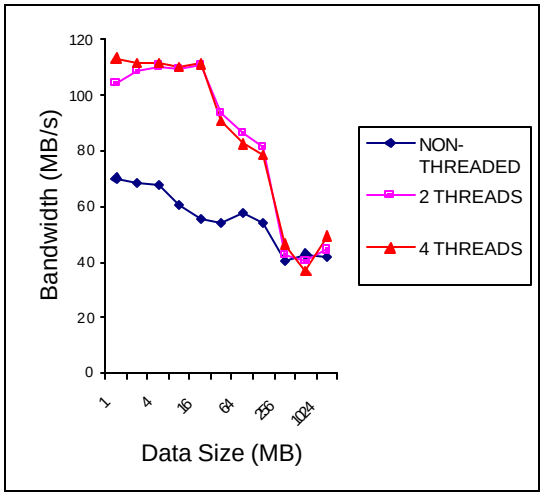

Figure 7. Impact of threading on Read performance

4.6 Data Mining using FICUS

The ultimate test of the effectiveness of any file system is the performance benefit for applications using it. We ran a parallel data mining application with the data sitting remotely and the application acting as a remote client. The details of this application can be found in [19].

We ran the mining algorithm over a dataset containing 10 million records and the net amount of data was close to 6 GB. We measured the time taken for the completion of the I/O phase of the algorithm.

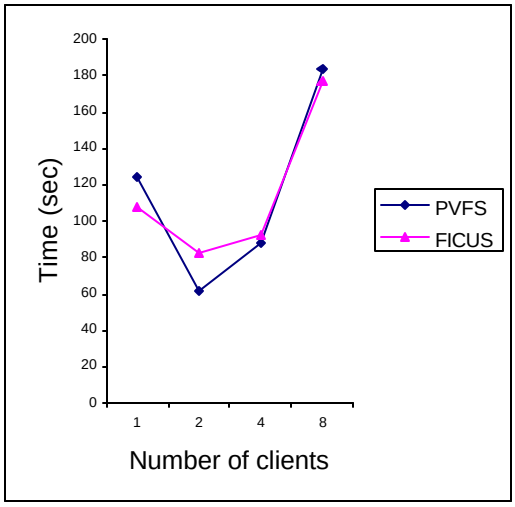

Figure 8. Time taken for mining 10 million records

We observe in Fig. 8. that the time taken for mining remote data is comparable to the time taken under PVFS. We also see that the time increases as we increase the number of processes involved in data mining because of the large load on the I/O servers.

4.7 Impact of Read Caching

We also wanted to measure the impact of caching on application performance. We implemented a simple caching algorithm on the FICUS I/O

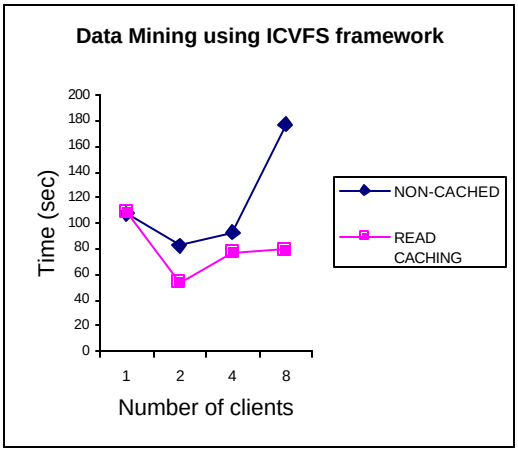

Figure 9. Data mining using FICUS

We observe in Fig. 9 that read caching has a noticeable effect on the time taken to complete the data mining process when the data is delivered to the remote client by the I/O proxy itself. This not only reduces the time taken to obtain the data, but also reduces the amount of load on the PVFS I/O server. This results in better performance particularly in the case of 8 clients, while the non-cached implementation is affected adversely.

4.8 Inter-cluster cooperation

The main aim of this file system was to develop inter-cluster cooperation. We have performed all our experiments on our partitioned cluster because of connectivity and security issues with other clusters. We have a 100 Mbits/s link between our cluster and the main gateway node of the Ohio Super Computing Center. We configured our cluster as the data repository, ran a client on the remote OSC node and accessed the FICUS file system.

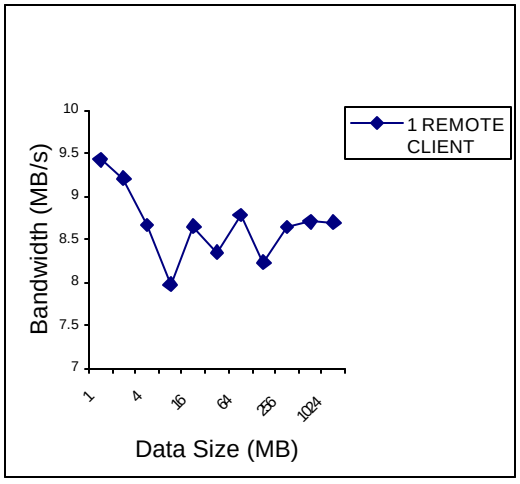

Figure 10. Remote client access using FICUS

We were using 2 I/O proxies and thus could attain a theoretical peak bandwidth of around 12 MB/s. We could achieve around 10 MB/s out of the theoretical peak of 12 MB/s as shown in Fig. 10. The main aim of this experiment is to prove the feasibility of performing large remote I/O under FICUS (around 1 GB here).

5.Related Work

Frangipani [11] is based on the simple model of a set of cooperating machines using a shared storage space. It also virtualizes access to raw disk and allows for standard UNIX semantics, which is quite similar to FICUS. However, it is essentially serverless in nature with the service being distributed over all the machines, while FICUS has server code for metadata and I/O handling. FICUS is mainly geared towards high performance obtained by data striping with simple POSIX semantics for locking. Frangipani is a more sophisticated distributed file system with emphasis on reliability and fault tolerance and boasts of advanced features like a separate lock service for concurrency control.

The *Andrew File System* (AFS) [18] also provides for a global namespace (root directory like FICUS) and uses caching to boost performance. FICUS maintains the data at a single location and uses the proxy file system to stripe the data to remote clients. In contrast, AFS moves the data to a server near the client access point. Moreover, AFS is geared towards operations in a wide area network, while FICUS utilizes the local bandwidth available in a cluster (in the form of the local PVFS file system). AFS also provides for advanced security and authentication techniques like Kerberos, while FICUS uses simple ACL entries on the Information Server to control access.

Lustre [5] is a next generation cluster file system, which has similar design level features like FICUS. It stores the data for directories(essentially metadata) on the metadata server itself and also includes a striping descriptor as a part of metadata. However, Lustre is designed as an object based file system, while FICUS is block based. It also virtualizes the memory and volume management functionality to an object based disk, which in turn requires special intelligent disk controller support. FICUS utilizes the local file system on the I/O server node in order to manage its storage. Lustre is also a SAN (Storage Area Network) based shared file system, while FICUS can use common System Area Networks like Myrinet or Gigabit Ethernet.

Sistina'a GFS [14] is also a SAN based cluster file system, which offers the ability to have concurrent readers to shared storage, like FICUS. However, GFS offers advanced features geared towards reliability and fault tolerance, like distributed metadata format, redundant data pathways and journaling. GFS also requires a SAN fabric between the I/O servers and the storage devices. FICUS is a simple implementation, which is geared towards high-speed access to remote data using commodity components. It relies on the local file system on the I/O server in order to manage its data and thus is cheaper to implement. Its code is also open-source enabling people to add enhancements or changes freely, unlike GFS, which is a vendor specific implementation.

Slice [8] is a fast I/O layer built on top of the Trapeze file system. It redirects I/O requests to an array of I/O servers incorporating a simple striping methodology like FICUS. It also offers full compatibility at the kernel level. FICUS mainly uses TCP over a System Area Network like Myrinet for attaining good performance, while Slice uses NetRPC to send data blocks (thus bypassing the kernel TCP/IP stack). This feature of using a User Level protocol for data communication could be incorporated into FICUS in the future. Slice was also a minimal implementation and thus used the kernel NFS layer for its file system interactions, while FICUS is a full fledged file system implementation that offers integration from the VFS layer itself.

GirdFTP [6] is an enhancement to FTP optimized for high-bandwidth wide- area networks. It supports parallel file transfers using multiple servers and provides for automatic negotiation of parallelization. The FICUS file system also provides for default striping patterns in order to optimizes data striping across a set of I/O servers. FICUS is a full-fledged file system with tight kernel integration. In contrast, GridFTP is just an FTP enhancement with no file system semantics or operations supported.

6.Conclusions and Future Work.

In this paper, we have proposed a new I/O file system that could be used for achieving inter-cluster cooperation. We also discussed various design details of this file system like threading and caching, which were critical in attaining better I/O performance. We also discussed the feasibility of performing large amount of I/O from a remote client without any stability problems. This file system has complete compatibility with the Linux kernel module supplied along with PVFS. This means that the user could run UNIX commands like ls, cp etc on remote files or directories under this file system, which makes

administration of this file system effortless.

We plan to run more experiments between our cluster and the OSC 72-node Itanium cluster. We will be studying further enhancements to the caching algorithm, which we have shown has the potential to yield better read performance. We are also examining the possibility of using ULNP (User Level Networking Protocols) like VIA in the main data pathway instead of TCP, for better performance.

We understand that data striping has limitations in the area of reliability. Our group is working towards incorporating RAID-like redundancy to the PVFS file system, which in turn would benefit FICUS. We are also looking at the feasibility of integrating two clusters to act as the data repository.

We are using pthreads in our current implementation but we feel that this file system could greatly benefit from the introduction of kernel level threading libraries and asynchronous I/O calls into the Linux kernel.

Acknowledgement: This work was partially supported by the Ohio Supercomputer Center grant # PAS0036-1. We are grateful to Dr. Pete Wyckoff and Troy Baer of OSC for their help in setting up the experiments with the OSC clusters. We would like to thank Dr. Rob Ross of Argonne National Labs, for clarifying many intricate details of the PVFS protocol and for making available the PVFS source to the research community.

Repository of FICUS Distribution: We plan to create a public repository of the FICUS code and distribute them. If you are interested in participating in this effort, please contact Dr. Mario Lauria (lauria@cis.ohio-state.edu).

References

[1] GriPhyN project, in http://griphyn.org

[2] NPACI Scalable Visualization Tools Webpage, in http://vistools.npaci.edu/.

[3] Protein Data Bank Webpage, in http://www.rcsb.org/pdb/.

[4] Sloan Digital Sky Survey Webpage, in http://www.sdss.org/.

[5] "Lustre – The inter-galactic cluster file system" – A technical overview of Lustre. *http://www.lustre.org/docs/lustretechnical-fall2002.pdf*

[6] Globus: GridFTP Protocol and Software. *http://www.globus.org/datagrid/gridftp.html*

[7] Altschul, S., et al., Basic local alignment search tool. *Journal of Molecular Biology*, 1990. **215**: p. 403-410.

[8] Darrell Anderson, Jeff Chase et al, "Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet", *Proceedings of the 1998 USENIX Technical Conference*, June 1998.

[9] Benson, D.A., et al., GenBank. *Nucleic Acids Research*, 2000. **28**: p. 15-18.

[10] P.H. Carns, W.B. Ligon III, R.B. Ross and R. Thakur. PVFS: A Parallel File System for Linux Clusters. *In*

PROC of the 4th Annual Linux Showcase and Conference, 2000.

[11] A.Thekkath Chandramohan, Mann Timothy and K.Lee Edward, "Frangipani: A scalable distributed file system", *Systems Research Center, Digital Equipment Corporation*, California.

[12] Avery Ching, Alok Choudhary, Wei-keng Liao, Robert Ross, and William Gropp, "Noncontiguous I/O through PVFS," *Proceedings of 2002 IEEE International Conference on Cluster Computing*, September 2002.

[13] Durbin, R., et al., Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, A tutorial introduction to hidden Markov models and other probabilistic modeling approaches in computational sequence analysis. 1998.

[14] Preslan Kenneth, Barry Andrew et al, ", A 64-bit shared disk file system for Linux", *Sixteenth IEEE Mass Storage Symposium*, March 1999.

[15] Krogh, A., et al., Hidden Markov models in computational biology: Applications to protein modeling. *JMB*, 1994. **235**: p. 1501-1531.

[16] Ligon, III, W.B., and Ross, R. B., "Server-Side Scheduling in Cluster Parallel I/O Systems", *Calculateurs Parallèles Journal Special Issue on Parallel I/O for Cluster Computing*, accepted for publication October 2001.

[17] R.B. Ross Providing Parallel I/O on Linux Clusters. *In Second Annual Linux Storage Management Workshop*, 2000.

[18] Satyanarayanan M, **"**Scalable, Secure, and Highly Available Distributed File Access", *IEEE Computer*, May 1990.

[19] S. Parthasarathy and A. Ramakrishnan, "Parallel Incremental 2D Discretization", *International Parallel and Distributed Processing Symposium*, 2002.

[20] Stephen Lord. Porting XFS to Linux, *Ottawa Linux Symposium*, July 2000.

[21] Stiles, J.R., et al. Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes. In *Computational Neuroscience*. 1998. New York, NY.

[22] E. Nallipogu, F. Ozguner, M. Lauria, *"Improving the Throughput of Remote Storage Acccess through pipelining"*, *3rd International Workshop on Grid Computing (GRID 2002) at Supercomputing `02*, Baltimore, November, 2002.

[23] K. Bell, A. Chien, M. Lauria, *"A High-Performance Cluster Storage Server"*, *HPDC-11*, Edinburgh, July 2002