5-1-2002

An Adaptable Benchmark for MPFS Performance Testing

Yubing Wang
EMC Corporation, wang_yubing@emc.com

Mark Claypool
Worcester Polytechnic Institute, claypool@wpi.edu

An Adaptable Benchmark for MPFS Performance Testing

Yubing Wang
EMC Corporation
Hopkinton, MA 01748, USA
wang_yubing@emc.com

Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609, USA
claypool@cs.wpi.edu

Abstract—Multiplex File System (MPFS) is a new network file access protocol that allows heterogeneous hosts to directly access shared data on a network-attached storage device. Current I/O benchmarks are not suitable to measure the performance of MPFS because they mainly measure the performance of NFS or CIFS. The MPFS split data-metadata architecture demands new ways to evaluate performance. We develop a new benchmark that can measure the performance of MPFS for a variety of applications. Our benchmark uses a series of application groups that can mimic the applications that use MPFS. The mix of these application groups generates the workload to stress the MPFS system. We illustrate the use of benchmark with a series of MPFS tests. We find that our benchmark does meet the criteria we apply to the ideal MPFS benchmark.

1 Introduction

Increasingly, enterprises require reliable, high-speed access from multiple computers to shared file systems. NFS (typically on UNIX systems) and CIFS (typically on Microsoft Windows systems) are widely used methods to share files among multiple computers. NFS and CIFS provide a logically shared view of file systems, allowing remote files to appear as if they were local files and allowing multiple computers to share the files. Unfortunately, accessing remote files over networks can be significantly slower than accessing local files. The network hardware and software introduces delays that tend to significantly lower transaction rates (I/O operations per second) and the bandwidth (megabytes per second). These delays are difficult to avoid in the client-server architecture of network-based file systems since the requests and data transfers must pass through the centralized file server.

In response to the demands for high-speed and highly scalable storage-to-server and server-to-server connectivity, the Fibre Channel interconnect protocol, an industry-networking standard, was developed in the early 1990s [KR00]. Using compatible Fibre Channel Host Based Adapters (HBAs) any workstation can address the storage and access data. The network, which includes HBAs, hubs, switches and the network-attached storage, is called a Storage Area Network (SAN). The distinct difference between using a SAN and a conventional Client-Server distributed systems is that in a SAN any workstation connected to the network fabric can directly access the network-attached storage devices.

The early SAN devices were used only in a server-attached shared storage environment in which disk storage systems were connected to network file servers using the Fibre Channel interconnect protocol, but in order to access the storage devices client workstations still sent messages to servers via a local area network (LAN). These network architectures provided improved disk and interconnect performance but failed to solve many of the shortcomings of conventional client/server distributed computing models where the server is the bottleneck to performance. In the last three or four years however, a number of vendors have begun developing and offering distributed file system solutions that exploit the SAN distributed environment [O98](SAN File Systems). A SAN File System is an architecture for distributed file systems based on shared storage that fully exploits the special characteristics of Fibre Channel-based LANs. The key feature of the SAN File System (SAN FS) is that clients transfer data directly from the device across the SAN.

EMC's Multiplex File System (MPFS) is a unique SAN FS that provides high-bandwidth access to shared files from heterogeneous hosts directly attached to the storage system. The hosts interact with an MPFS server for synchronization, access control, and metadata management, but transfer data directly to and from the storage device. Rather than replace traditional file access protocols, MPFS interoperates with, and uses, standard protocols such as NFS and CIFS for control and metadata operations. It adds a thin, lightweight protocol called the File Mapping Protocol (FMP) for exchanging file layout information between the hosts and the MPFS server, thus allowing the hosts to directly read and write file data.

The major design goal of MPFS is to achieve superior performance to NFS in terms of scalability and I/O throughput. The MPFS split data-metadata architecture demands new means to evaluate performance since current file system and I/O benchmarks are not suitable for MPFS performance evaluation. First of all, most current file system benchmarks only measure server performance. MPFS allows clients to directly access the storage subsystem, in contrast to NFS or CIFS in which the server is the only gateway to access the storage subsystem. Therefore, in order to measure MPFS performance, a benchmark should target both server and client. Second, most current file system benchmarks are protocol specific. They can be classified as either an NFS benchmark, such as SPEC SFS [SPEC97], or a CIFS benchmark, such as NetBench [ZD00]. Since MPFS is a different file access protocol, we can use neither of NFS or CIFS benchmarks to measure MPFS performance. Third, most current file system benchmarks cannot support multiple file access protocols. MPFS uses NFS or CIFS as an underlying file access protocol for metadata operations. In order to measure MPFS

performance and compare MPFS with other file access protocols, the benchmark should support multiple file access protocols. Fourth, most current file system benchmarks are not portable across different operating systems. MPFS supports various operating systems, such as Solaris, Windows NT, Linux, AIX, etc., so the benchmark should be portable across various operating systems.

In this paper, we propose a new benchmark for MPFS performance evaluation. Our benchmark consists of a number of application groups that generate I/O intensive workloads. The application groups target some typical real-world MPFS applications and mimic them at the file system level using a set of system calls in various combinations. The performance metrics of our benchmark includes the critical performance aspects of MPFS: throughput, response time, scalability, file sharing, metadata operations and data transfer. Our benchmark can dynamically adjust the mix of file operations according to different application performance characteristics. Our benchmark is highly scalable in terms of file set selection, being able to scale file sizes from small to large to mimic different applications. Since our benchmark is an application level benchmark, it supports both UNIX clients and Windows NT clients. Like SPEC SFS 2.0 [SPEC97], our benchmark reports the average response time at the delivered load.

Section 2 discusses the current file system and I/O benchmarks and shows how they fall short of criteria we develop for the ideal MPFS benchmark. Section 3 describes our benchmarking methodology in detail. Section 4 presents the performance data collected from our benchmark. Section 5 summarizes our conclusions and presents possible future work.

2 Related Work

In this section, we discuss the current file system and I/O benchmarks and show how these benchmarks fall short of the criteria for the ideal MPFS benchmark. We briefly describe four common benchmarks used in file system evaluation: SPEC SFS2.0 [SPEC97], NetBench [ZD00], IOStone [PB90], and Bonnie [B90].

SPEC SFS 2.0

SPEC SFS 2.0 is a synthetic benchmark that uses a mix of NFS operations to stress a network server. SPEC SFS 2.0 is highly parameterized; besides the percentage of each operation in the workload, SPEC SFS 2.0 gives a user the ability to change the number of clients issuing requests to the server, the rate at which each client issues requests, the total size of all files, the block size of I/O requests, and the percentage of write requests that append to

an exiting file. The metric for SPEC SFS 2.0 is a throughput (NFS operations per second) versus response time graph.

However, since SPEC SFS 2.0 measures NFS server performance by generating NFS client RPCs, it cannot be used to compare NFS against other protocols, nor can it be used to test client-side caching strategies [M99]. The ideal MPFS benchmark should support both CIFS and NFS protocols.

NetBench

NetBench is a portable I/O benchmark program that measures the performance of file servers as they handle network file requests from clients. NetBench accepts as clients PCs running Windows platforms. NetBench takes leading applications for Windows-based PCs and profiles them to determine the file operations they perform and how frequently they perform them. After profiling the applications, NetBench creates scripts to mimic the network file operations of those applications. NetBench performs a variety of network file operations across a large set of files in numerous directories.

NetBench is an application level I/O benchmark that measures file server performance only from Windows clients and therefore cannot be used to measure systems with clients running UNIX. The ideal MPFS benchmark should support both UNIX and Windows platforms.

IOStone

IOStone is a synthetic I/O benchmark based on system traces of Unix minicomputers and workstations and IBM mainframes. One process performs all the accesses without I/O parallelism. IOStone reports a single throughput result.

IOStone claims to be an I/O benchmark, but actually measures the memory subsystem; all of the tests fit easily in the cache since it only accesses a one MB file set. The ideal MPFS benchmark should be I/O intensive and access much larger file sets than does IOStone.

Bonnie

Bonnie runs six different workloads that show the performance difference between reads versus writes and block versus character I/O. One workload sequentially reads an entire file a character a time; another writes a file a character at a time. Other workloads exercise block-sized sequential reads, writes, or reads followed by writes (rewrite). The final workload uses three processes to simultaneously issue random I/Os. For each workload, Bonnie reports throughput, measured in KB per second or I/Os per second. Therefore, Bonnie is actually a disk benchmark.

While Bonnie focuses on disk I/O, it does not vary other aspects of the workload, such as the number of concurrent I/Os. The ideal MPFS benchmark should be a benchmark that measures both disk subsystems and file systems. The ideal MPFS benchmark should also be able to scale the number of processes issuing I/O and number of hosts that have concurrent I/O processes.

In summary, current I/O and file system benchmark suites fall short of the criteria we seek in the ideal MPFS benchmark. This suggests the need for a new benchmark to measure MPFS performance.

3 Approach

This section describes approaches to construct an MPFS benchmark.

3.1 Overview

Based on the performance metrics we define, we construct application groups that measure the critical performance characteristics of MPFS. The application groups target both I/O operations and metadata operations for MPFS. We derive the application mix percentage from the low-level NFS and CIFS operation mixes since MPFS uses NFS and CIFS as the underlying protocols for control and metadata operations. The file set for our MPFS benchmark is application-based, where each application group has its own file set. The file access pattern for our MPFS benchmark is based on a previous file access trace study [RLA00]. There are four major issues we consider when we design the file access patterns for our benchmark: file access order, file access locality, file access burst, and overwrite/append ratio. Our benchmark distributes the think time between requests exponentially and the load-generating processes in each load-generator are Poisson distributed to mimic a real-life workload. Our benchmark chooses the next operation to perform based on a probability derived from the operation mix and an operation context map that follows a common operation sequence. In order to measure the data sharing performance for MPFS, our benchmark has five application groups that interact with different types of sharing.

3.2 Workload Generation

3.2.1 Application Group Construction

Our application groups allow us to measure the critical performance characteristics of MPFS, especially for read/write operations and scalability. We create a total of 12 application groups where each group consists of a sequence of file I/Os or metadata operations. Since

MPFS has separate metadata and real data paths, our application groups include both I/O operation groups and metadata operation groups. Each application group is independent of each other. Our benchmark uses a sequence of application groups with various combinations to generate workloads.

3.2.1.1 I/O Operation Groups

The I/O operation groups focus on file I/O operations. Since MPFS data access can happen at channel speeds rather than at network speeds, MPFS should have significant improvement in I/O throughput and latency.

We have total of seven I/O operation groups: *read, write, read-write, memory mapping, read sharing, write sharing* and *read-write sharing.*

3.2.1.2 Meta Operation Groups

The metadata operation groups focus on file metadata operations. The performance of metadata operations in terms of latency and throughput is important to evaluate overall MPFS performance since high overhead for metadata operations may greatly degrade system performance.

We have total of five metadata operation groups: *file stat, get attribute, set attribute, directory,* and *symbolic link.*

3.2.1.3 Windows NT/2000 Operation Groups

The operation groups for Windows NT/2000 contain 16 I/O operations, which follow the file I/O calls used in the Disk Mix test of NetBench [ZD00]. The 16 I/O operations are determined by profiling 9 leading applications for Windows-based PCs [ZD00].

3.2.2 Application Mix Tuning

We have a total of 12 application groups that consist of 24 independent file operations including 16 metadata operations and 8 I/O operations. The mix of these 24 file operations generates the workload that stresses the MPFS system. The mix percentage of these 24 file operations is derived from the low-level NFS or CIFS operation mix percentage since MPFS uses the NFS or CIFS protocol for control and metadata operations.

3.2.2.1 Application Mix Tuning based on NFS Operation Mix

The NFS operation mix percentage we use is the NFS version 3 mix published by SPEC SFS 2.0 [SPEC97].

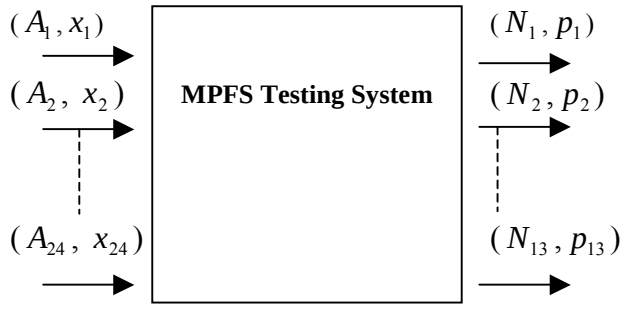

Figure 3.1 Derivation of the application mix from the NFS operation mix

Figure 3.1 shows how we derive the application mix from the NFS operation mix. (A_n, x_n) represents the file operation A_n, where $n = 1, 2, \cdots, 24$ corresponding to the 24 file operations, and the corresponding mix percentage x_n, where $x_1 + x_2 + \cdots + x_{24} = 1$. (N_n, p_n) represents the NFS operation N_n, where $n = 1, 2, \cdots, 13$ corresponding to 13 NFS operations defined by NFS Version 3, and the corresponding mix percentage p_n, which are obtained at the server by using the NFS profiling utility `nfs_stat`. We start by applying one application group, say A_1, to the MPFS Benchmark. The NFS profiling utility at the server captures the network trace with which we can calculate the corresponding NFS operation mix percentage: $(p_{1,1}, p_{1,2}, \ldots p_{1,13})$. Continuing the same procedure for other application groups, we obtain all NFS operation mix percentages to form the following matrix:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,12} & P_{1,13} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,12} & P_{2,13} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ P_{23,1} & P_{23,2} & \cdots & P_{23,12} & P_{23,13} \\ P_{24,1} & P_{24,2} & \cdots & P_{24,12} & P_{24,13} \end{bmatrix} \qquad (3.1)$$

Assuming the MPFS testing system is linear, we then have the following linear equation:

$$\begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,23} & P_{1,24} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,23} & P_{2,24} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ P_{12,1} & P_{12,2} & \cdots & P_{12,23} & P_{12,24} \\ P_{13,1} & P_{13,2} & \cdots & P_{13,23} & P_{13,24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{19} \\ x_{24} \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{12} \\ p_{13} \end{bmatrix} \qquad (3.2)$$

Solving this equation, with the constraint:

$$x_1 + x_2 + \cdots + x_{24} = 1 \qquad (3.3)$$

we obtain the application level mix percentage.

It is very hard to characterize this linear equation and present the solution using the conventional method. Instead we develop an algorithm to tune the application level mix percentage to approximate the desired one. Our algorithm tries to solve the following problem:

> *Given the desired NFS operation mix percentage, which is either the NFS version 3 mix published by SPEC SFS 2.0 or one specified by the users, find the file operation mix percentage* $(x_1, x_2, \cdots, x_{24})$ *such that the underlying NFS mix percentage approximates the desired NFS operation mix percentage.*

Our algorithm is based on the observation that a solution to the linear equation (3.2) lies in the compact region R defined by:

$$R = \left\{ (x_1, x_2, \cdots, x_{24}) : 0 \le x_i, \sum_{i=1}^{24} x_i = 1, i = 1, 2, \cdots, 24 \right\} \quad (3.4)$$

For each point $(x_1, x_2, \cdots, x_{24})$ in the region R, there is an underlying NFS operation mix percentage $P = (p_1, p_2, \cdots p_{13})$, which satisfies:

$$\min_{p \in R} D(P, \varphi) = 0 \qquad (3.5)$$

Where φ is the desired NFS operation mix percentage, and D is the Euclidean distance between P and φ defined by:

$$D(P, \varphi) = \sqrt{\sum_{j=1}^{13} (p_j - \varphi_j)^2} \qquad (3.6)$$

The algorithm searches for the root of the linear equation (3.2) by computing $D(P, \varphi)$ for all points $(x_1, x_2, \cdots, x_{24})$ in R. This algorithm does not require an initial guess and it gives an answer that satisfies the inequality constraints in equation (3.4).

The algorithm attempts to find a point in R that achieves the minimum in equation (3.5). However, the computer cannot scan the bounded region R continuously but must do so in increments of size $\delta > 0$ for each x_i coordinate for $i = 1, \cdots, 24$. Therefore, in practice, we compute $D(P, \varphi)$ only for points $(x_1, x_2, \cdots, x_{24})$ belonging to a finite lattice of isolated points in R, and not for all points in R. After each increment is made and the new $D(P, \varphi)$ is

computed, the running minimum value and its coordinate are updated. Once the lattice is exhausted, the final minimum value and its coordinates are declared as the final answer. Thus, in the vast majority of the cases the algorithm can only give an approximation to the root(s) of linear equation (3.2). If in the course of computing $D(P,\varphi)$ and updating the running minimum, $D(P,\varphi)$ becomes negligible, i.e., $D(P,\varphi) \le \varepsilon$, the program stops and declares $(x_1, x_2, \cdots, x_{24})$ as the solution.

3.2.2.2 Application Mix Tuning based on CIFS Operation Mix

As mentioned in section 3.3.1.3, the operation groups for Windows NT/2000 contain 16 file operations, which follow the file I/O operations used in the Disk Mix test of NetBench. NetBench took the leading applications for Windows-based PCs and profiled them to determine the types of file requests they performed and how frequently they preformed them. Since NetBench is the leading benchmark for CIFS and MPFS for Windows NT/2000 uses CIFS as the underlying protocol, our benchmark adjusts the application mix percentage to match the underlying CIFS operation percentage, which is obtained by profiling NetBench. We profile NetBench since, unlike SPEC SFS 2.0, NetBench did not publish its CIFS operation mix percentage. We captured the CIFS operation mix percentage by running NetBench against the EMC Celerra file server and using our own profiling tool to capture the underlying CIFS operations. Using the tuning procedure described in section 3.2.2.1, we tune the application level mix percentage for the 16 file operations.

3.2.3 File Set Construction

The file set for our benchmark is application-based where each application group has its own file set. Different application groups may have different requirements for their file set. For instance, the file set for read or write application groups are quite different from that for directory or symbolic link application groups. According to [RLA00], most accessed files are small, but the size of the largest files continues to increase. To reflect this trend and also consider the fact that a large number of MPFS applications access large files, each application file set consists of small, medium and large file sets, each of which has its own file size distribution and is assigned an access probability specified by the user.

3.2.4 File Access Patterns

3.2.4.1 Read and Write Patterns

Our benchmark supports four access patterns: file access order, file access locality, file access burst and overwrite/append ratio.

File Access Order

We define two types of file access order: sequential and random. We classify a file access as sequential if it reads or writes a file in order from beginning to end, and random otherwise. According to [RLA00], small files tend to be accessed sequentially, while large files tend to be accessed randomly, which we use as a guideline for file selection. In other words, we choose a small file for sequential access and a large file for random access.

File Access Locality

File access locality refers to the fact that the same files tend to get the same type of access repeatedly. According to [RLA00][RW93], overwrite and read have significant locality, where the same files tend to get repeatedly overwritten and many files are repeatedly read without being written. To emulate the overwrite locality, our benchmark typically chooses the most currently overwritten files for the next overwrite. The read locality is achieved by creating separate file sets for read and write operation groups and choosing the most currently read files for the next read operation.

File Access Burst

File access burst refers to the fact that certain file access pattern occurs in bursts. According to [RLA00], write operations tend to occur in bursts. To emulate the write burst, a portion of each write operation group comprises a sequence of consecutive write operations.

Overwrite/Append Ratio

Our benchmark performs two distinct types of write operations: writes that overwrite existing data in a file, and writes that append new data to the end of a file. These operations exercise different parts of the server's file system implementation. Depending on the data buffer alignment, over-writing data may require pre-fetching from the old data before overwriting part of it, while appending new data may require allocating space from the file system's pool of free storage space. To create an I/O intensive workload to measure MPFS I/O performance, the default parameters of our benchmark specify that 20% of write operations over-write existing data and 80% of write operations append new data.

3.2.4.2 Meta Data Access Patterns

The file access pattern for metadata operation groups follows a non-uniform, file access distribution algorithm that is implemented in LADDIS [Wk93]. During our benchmark initialization phase, the file set is partitioned into small groups of files or directories, and each group is assigned a probability of providing the next file or directory in order to be accessed. The probabilities are based on a Poisson distribution, which increase file access locality according to [Wk93]. Each group contains the same number of files or directories, and the files or directories within each group have an equal chance of being selected.

3.2.5 Work Load Management

Our benchmark distributes the think time between requests exponentially and the load-generating processes in each load-generator are Poisson distributed to mimic a real-life workload. The arrival rate is the load level measured in operations per second.

3.2.5.1 Load Distribution

Our benchmark distributes the load equally among the load-generators but instead of creating a fixed number of processes that share the workload equally, our benchmark creates a pool of threads, each of which performs certain operation asynchronously. The main thread in each load-generator waits a period of time, which is computed from an exponential distribution, and selects an idle thread to perform the next operation. After a thread finishes its task, it becomes idle and is put back into the thread pool.

3.2.5.2 Think Time Calculation

The think time is computed from an exponential distribution. We generate an exponential variable as follows:

$$t = -\frac{1}{\lambda}\ln(u)$$

Where u is a random variable uniformly distributed between 0 and 1, and λ is the load level measured in operations per second specified by the user.

3.2.5.3 Operation Selection

Our benchmark chooses the next operation to perform based on a probability derived from the operation mix and an operation context map that follows the common operation sequence. We build a hash-table in which each entry records the number of times each operation has been performed and contains a linked list that forms the operation context for that operation. The

operation context follows the common operation sequence, e.g., a READDIR followed by a GET ATTRIBUTE, or a LOOKUP followed by a READ.

Using the hash-table, our benchmark determines whether a selected operation has been performed the target number of times and what operation should be selected next. If a selected operation has reached its target, our benchmark randomly chooses another operation to perform. If a selected operation has not reached its target yet, our benchmark performs it and proceeds to the next operation in the linked list.

3.2.6 Read and Write Sharing

Sharing measures the system throughput and response time when multiple clients access the same data. Since MPFS allows clients to access the shared data directly, MPFS must provide some mechanisms to prevent data corruption, but may thereby reduce performance in terms of throughput and latency.

Our benchmark has four application groups associated with the performance measurement for data sharing: read sharing in a single client, read sharing in multiple clients, write sharing in a single client and write sharing in multiple clients.

3.2.7 Caching

Since our benchmark measures the overall MPFS system performance including server and clients, the variations in client hardware and MPFS implementation details such as caching may affect the accuracy. In order to avoid caching effect by the client, our benchmark creates huge file sets for each application group and makes the file selection as random as possible.

Each file set is partitioned into small groups of files, and each group is assigned a probability, drawn from a Poisson distribution, of providing the next file to be assessed. Each group contains the same number of files, and the files within each group have an equal chance of being selected.

To eliminate the caching effects due to the file set creations in each client, our benchmark use separate clients to create file sets. Typically, the cache hit rates in both client and server sides are kept below 10%.

4 MPFS Performance Data Analysis

Our MPFS benchmark has successfully been used internally by EMC for tuning Celerra File Servers. This section presents some MPFS performance data collected from our benchmarking testbed. The performance data collection is based on throughput, response time, and scalability.

4.1 Throughput and Response Time

$X \quad 24$

Figure 4.1Generated Load Vs. Response Time for a MPFS Benchmarking Testbed with 8 Solaris Clients

Figure 4.1 shows a graph of generated load vs. response time for our MPFS benchmarking testbed with 8 Solaris clients. Our MPFS benchmark slowly increases the load level (measured in Ops/Sec) on the MPFS benchmarking testbed while measuring the system's average response time. For each load level, we run 8 tests to get the average response time. The average response time per operation is slowly increased from 3.2 milliseconds to 5.7 milliseconds as the generated load is increased from 100 Ops/Sec to 3700 Ops/Sec. Beyond 3700 Ops/Sec the average response time increases rapidly, which indicates that 3700 Ops/Sec is the optimal operating point.

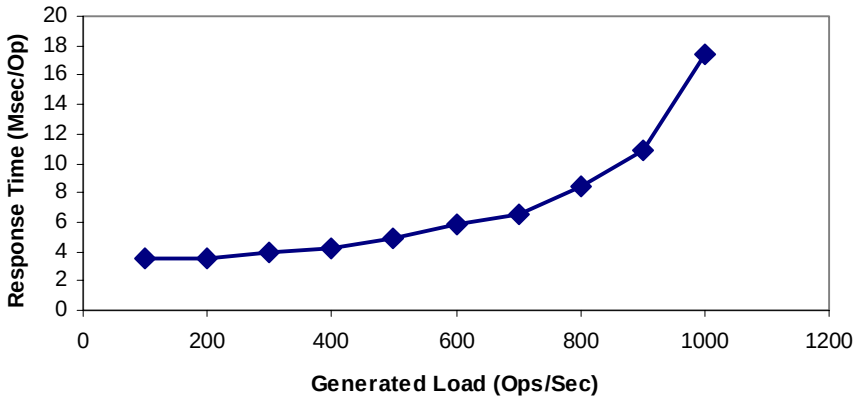

Figure 4.2 Generated Load Vs. Response Time for the MPFS Benchmarking testbed with 8 Windows NT Clients

Figure 4.2 shows a graph of generated load vs. response time for an MPFS benchmarking testbed with 8 Windows NT clients. The average response time per operation slowly increases from 3.5 milliseconds to 6.5 milliseconds as the generated load increases from 100 Ops/Sec to 700 Ops/Sec. Beyond 700 Ops/Sec the average response time increases rapidly, which indicates that 700 Ops/Sec is the optimal operating point.

4.3 Scalability

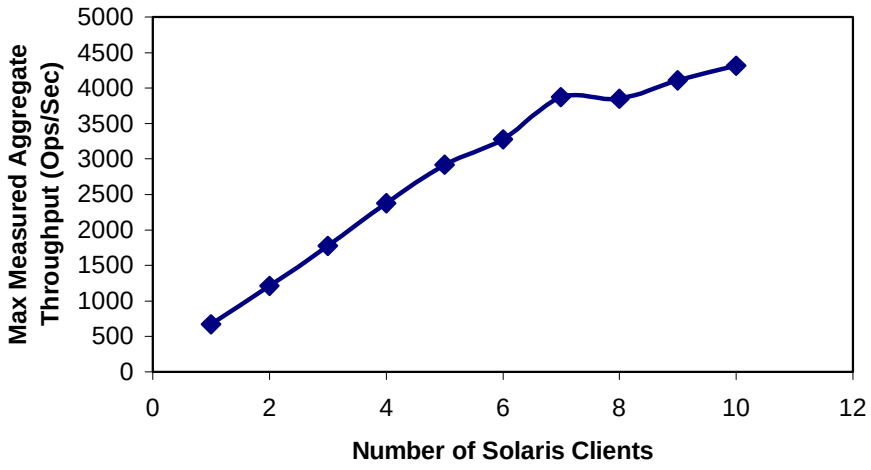

Figure 4.3 Measured Maximum Aggregate Throughput versus Number of Solaris Clients

Figure 4.3 shows a graph of measured maximum aggregate throughput versus number of Solaris clients. The measured maximum aggregate throughput represents the highest total throughput that can be achieved in our MPFS Solaris benchmarking testbed. As shown in the graph, the measured maximum aggregate throughput increases linearly as the number of clients in our testbed increases to from 1 to 7, but beyond 7 clients, the measured maximum aggregate throughput increases only slightly.

4.4 Comparison between MPFS and NFS

This section compares the performance of MPFS and NFS in terms of throughput, response time and scalability.

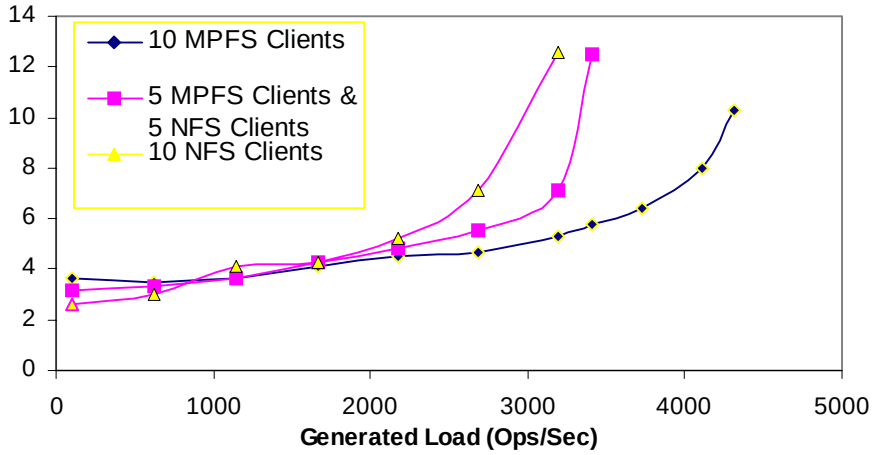

Figure 4.4 Generated Load Vs. Response Time for three different MPFS and NFS Solaris client combinations

Figure 4.4 shows the generated load versus response time for three different NFS and MPFS Solaris client combinations. It can be seen that MPFS provides better performance than NFS over the generated loads except for at very low load levels. The performance of the mix of MPFS and NFS is better than that of NFS but worse than that of MPFS.

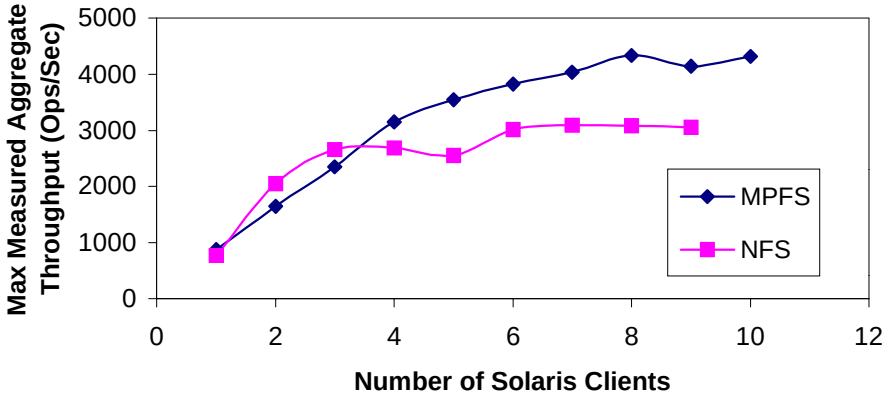

Figure 4.5 Measured Maximum Aggregate Throughputs versus Number of Solaris Clients for Comparison between MPFS and NFS

Figure 4.5 compares the scalability of MPFS and NFS. As depicted in figure 4.5, MPFS shows better scalability than does NFS after the number of Solaris clients in our testbed increases to 4. Below 4 clients, NFS shows slightly better scalability.

5 Conclusion and Future Work

EMC's Multiplex File System (MPFS) uses a split data-metadata architecture for distributed file systems based on shared storage, which fully exploits the special characteristics of Fiber Channel-based LANs. Therefore, in order to measure MPFS performance, an effective benchmark should target both the server and the client. Unfortunately, most current file system only measure server performance, which motivates us to develop a new file system and I/O benchmark for MPFS performance evaluation.

Our MPFS benchmark consists of a number of application groups that can generate I/O intensive workloads. The application groups target representative real-world MPFS applications and mimic them at the I/O level using a set of system calls in various combinations. Our MPFS benchmark uses a variety of approaches to mimic real-life MPFS applications. These approaches include allowing users to specify the application group mix for their specific applications, developing an algorithm for tuning the application group mix percentage to match the underlying NFS operation mix derived from an empirical study done by SUN [SPEC97], creating a scalable file set for our benchmarking system, using an exponentially distributed think time and simulating various real world file access patterns.

The ideal benchmark should be relevant to a wide range of applications and be able to predict a system's performance in a production environment. Currently, we have no way verifying that the synthetic workload generated by our MPFS benchmark actually represents current or future MPFS application workloads. Future work might include building up a large set of MPFS trace archives and developing a profiling model to characterize the traces. The characterization of the traces can used to generate a stochastic workload that can be executed as a benchmark.

Scalability is a major metric for MPFS performance. Currently, our benchmark uses the number of clients (load generators) to represent the scalability. This method has a clear drawback in that this is not a one-to-one mapping of a load generator and a client in a real application. This is due to our load-generating mechanism in which the synthetic workload generated by a load generator may mimic the workloads of more than one client. The mapping between the load generator and client in a real-world application is a subject to further investigate.

References

[B90] M. Berry, Bonnie source code, http://www.textuality.com/bonnie/intro.html, 1990.

[KR00] Robert W. Kembel, "Fibre Channel, A Comprehensive Introduction", Northwest Learning Associates, Inc., 2000.

[M99] Jeffrey C. Mogul, "Brittle Metrics in Operating System Research", Proceedings of 7th IEEE Workshop on Hot Topics in Operating systems, Rio Rico, AZ, (March, 1999).

[O98] M. T. O'Keefe, "Shared (Disk) File Systems", 1998, http://www.lcse.umn.edu/GFS.

[PB90] A. Park and J. C. Becker, "IOStone: A Synthetic File System Benchmark", Computer Architecture News, 18(2), pp. 45-52, June, 1990.

[RW93] Chris Ruemmler, John Wilkes, "Unix Disk Access Patterns", Proceedings of USENIX Winter Technical Conference, San Diego, CA, pp. 45-52, (January 25-29, 1993).

[RLA00] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson, "A Comparison of File System Workloads", Proceedings of USENIX Technical Conference, San Diego, California, June, 2000.

[SPEC97] SFS 2.0 Documentation Version 1.0, Standard Performance Evaluation Corporation (SPEC), 1997.

[WK93] Mark Wittle and Brian Keith, "LADDIS: The Next Generation in NFS File Server Benchmarking", Usenix, 1993.

[ZD00] Understanding and Using NetBench® 6.0, ZD Inc., 2000.