# Performance Measurements of Networked Files Systems

Winfred Byrd and John Moser
wbyrd@cs.wisc.edu, jmoser@erc.wisc.edu

SOS-Please Conference
cs736 Advanced Topics in Operating Systems
Fall 2000

Professor: Remzi Arpaci-Dusseau

December 18, 2000

**Abstract**

The inability of common benchmarks to gracefully scale to evaluate systems of various size and configuration contributes greatly to the incapacity usually shown in comparing varied systems under a common metric. This paper looks at an older, self-scaling I/O benchmark whose results can be normalized using techniques of predicted performance for comparative evaluation of a diverse range of stand-alone computer systems. We assert that there is no reason these comparisons need stop at local disk workstations and show the comparison to produce a valid evaluation of network file-system enabled computers using SUN's NFS protocol.

## 1. Introduction

The history of performance evaluation is littered with benchmarks unable to stress the larger and more powerful computers that evolved in their aftermath. This inability to scale greatly contributes to inaccuracies when evaluating and comparing varied systems under a common metric.

In the discussion that follows, we examine what characteristics a benchmark need have so that it may be used to realistically evaluate modern and future computer systems. We look at what we gain from each of these characteristics and why earlier proposed benchmarks lacking in them have grown obsolete. We then turn to a proposed benchmark that meets all of our stated criteria. Following this, we discuss the changes that were necessary to convert this benchmark for running over a distributed file system.

The remainder of this paper is divided into the following sections: section 2 sets the standards for an ideal benchmark, section 3 looks at related work, section 4 addresses our testing procedure, section 5 presents our results, section 6 considers future work, and in section 7 conclusions are drawn from the results.

## 2. Ideal Benchmarks

When attempting to evaluate systems, several questions naturally present themselves: What are common shortcomings among existing benchmarks? Is there a benchmark that is not limited by these shortcomings; or, more generally, what would be our ideal benchmark? Fortunately, the literature is rich with attempts to answer that question. One such claim obliges that an ideal I/O benchmark should:

- Increase understanding of the system: This includes helping designers isolate points of poor performance as well as instructing users in optimal machine usage under different workloads,
- Be I/O limited: Intuitively, the facet of the system we are attempting to evaluate should be the bottleneck of our system,
- Scale gracefully: We cannot expect comparative information between diverse systems to be useful if our evaluation program is ineffective at measuring systems in the performance range of one or more of the platforms in question. Further, we would like our evaluation to remain useful with the next generation of computers. Hence, scaling to stress more powerful systems is a must,
- Allow a fair comparison across machines,
- Relevant to many applications: We would like to be able to comparatively evaluate systems under all types of workloads: CAD, office automation, software development, and so on. Though design decisions are arguably much easier when we target a specific application, there is obvious value in very general applicability,
- Tightly Specified: For benchmarks to have meaning, they must be well defined, reproducible and reported with rigor. What optimizations, such as caching, are allowed? How is the physical machine environment configured? If we are going to hold our results as absolutely reliable, these questions become very important. [2]

With this more thorough specification of the required characteristics, we search for our ideal benchmark.

## 3. Related Work

In the past, the benchmarking of various components of computers has invariably followed a trend towards one or two standards. With NFS performance measurement, this hasn't changed. The computing industry has seen two distinct methods for judging NFS performance in two pieces of software, NFS/NHFSStone, and LADDIS.

### 3.1. NFS/NHFSStone

Barry Shein initially published NFSStone at the 1989 USENIX conference in a paper entitled "NFSSTONE - A Network File Server Performance Benchmark". Legato Systems created NHFSStone, which was incorporated as a NFS load-generating program. Both share many features. And while both may have been used in the past as a popular means of describing NFS performance, both also have many deficiencies. We will speak in general of NHFSStone, though the references apply to NFSStone as well (unless otherwise noted).[7]

First, both benchmarks utilized the synthetic duplication of an average user workload, on top of an average NFS resource utilization level. While this may give us an indication of what *might* be the performance of our system under a median-level workload, it does not indicate any sort of performance measurement to the extremities. Additionally, NHFSStone is limited in the number of clients it supports (one). Any attempt to document the performance of a large-scale network wouldn't find any help within NHFSStone. Without any network contention, the main bottleneck to NFS couldn't be accurately measured (that being the network infrastructure itself). Even if control coordination software were implemented to maintain several client copies of NHFSStone, the results reported would lose reliability as additional levels of influence were introduced.

Second, both NHFSStone and NFSStone are sensitive to the differences in NFS client implementation. The generated NFS command and request sequence produced for any given test could vary (sometimes greatly) amongst clients, depending on how their manufacturer decided to implement the NFS protocol. As the client workload increases, more inconsistencies are introduced into the results of the benchmark. NHFSStone attempted to work around these inconsistencies by implementing various algorithms, depending on what client was used. As can be expected, the effectiveness varied greatly, and the results produced weren't reliable. In fact, the only operating system that the benchmark truly worked well on (since it was written originally for that operating system) was SunOS. The benchmark utilized several operating specific algorithms, along with a high level access to kernel data structures (file name information, attributes, caching policies, etc). While this definitely simplifies use of the benchmark on a group of SunOS computers, it makes porting difficult unless entire libraries were ported as well. Even then inconsistencies abounded. Since the computing industry is moving away from homogeneous networks, it becomes even more important that a standardized benchmark be widely applicable.[7]

Finally, there hasn't been a standard issued for running the benchmark and reporting the results. Without a recognized standard, results can't be compared without a concern for more inconsistencies. Although there weren't many variable arguments, results become meaningless amongst different configurations and vendor platforms.

### 3.2. LADDIS

In order to fix many of the deficiencies of NFS/NHFSStone, several companies developed a new benchmarking standard. In 1992, Legato Systems, Auspex Systems, Data General, Digital Equipment, Interphase, and Sun Microsystems all contributed to the creation of LADDIS NFS. In 1993, the benchmark was adopted into the System File Server benchmark suite version 1.0 (developed by the Standard Performance Evaluation Corporation). They succeeded to a certain extent in fixing many of the NFS/NHFSStone deficiencies.[7]

First, LADDIS decreased sensitivity to the client's implementation of NFS because the NFS protocol is encapsulated and implemented in the benchmark itself. This eliminated the reliance on customizations and tweaks to ensure (semi-)fair operating conditions amongst varying client implementations. The request sequence generated by the benchmark was consistent amongst every instance that ran. Since the reliance on the client was removed, a benchmark was finally created that could be used on a heterogeneous network without fear of incorrect results.

Also, LADDIS is tunable to a certain degree. NFS request packet sizes, file working set size, and other parameters were fully tunable to aid in the understanding of system operation. However, while these parameters are fully tunable, they are also difficult to tune towards any significant result. There hasn't been a substantial documentation of how certain parameters should effect system operation. And without knowing what changing these parameters is supposed to accomplish, a user is better off leaving the benchmark in its stock configuration in order to ensure meaningful results (and thus negating the benefit of tunable parameters). The added benefit of being able to determine the effect of certain parameters on system operation is also lost. In addition, LADDIS lacks the ability to change the workload behavior, creating yet another synthetic workload pattern.

Finally, LADDIS is still only good at measuring the response time of an NFS server. It gives no indication as to why certain results are reported. If results are reported that seem inconsistent or incorrect, the user is left guessing as to whether it could've been due to the NFS server, the network infrastructure, or something else. There also is a lack of client side benchmarking.

### 3.3. A Self-Scaling I/O Benchmark

In 1993, Peter M. Chen and David A. Patterson proposed a new self-scaling benchmark. This benchmark operates in two easy steps. First, it explores the system and determines a workload that would properly stress the I/O subsystem of the system it runs on. It then proceeds to run a set of workloads and use the results to produce a set of performance graphs. This workload is characterized by five parameters:

- UniqueBytes – in essence, the total size of the data being operated on,
- SizeMean – the mean of the distribution over which the size of the I/O request ranges,
- ReadFrac – the fraction of read operations (as opposed to writes),
- SeqFrac – The fraction of requests that immediately follow the prior request on the disk,
- ProcessNum – the number of processes simultaneously issuing I/O. [1]

Looking at these five characteristics, processNum is least useful to us immediately. The number of processes should have little impact, unless we are accessing a concurrent disk such as certain types of RAID disks (which was not a system in our set of tested systems). However, taken broadly, we can now specify any number of user processes accessing as either read or write, and any number of any size of requests in any layout on the disk from any size working file set. From this discussion, we argue two points. First, this benchmark scales well. Since it can adapt itself to produce as little or as much workload as is needed, this is fairly intuitive. Second, this benchmark is I/O limited. If it didn't initially reach a limitation due to I/O, it could simply scale up to produce more I/O requests. Since the requests move at processor speed and the responses move at disk speed, at some point, the requests must begin stressing the system.

This however is a long way from establishing this as an ideal benchmark. We have said nothing as to how this evaluation provides an increased understanding of system operation. Further, the entire premise of self-scaling is that every system runs the benchmark under its own load. This seems to go a long way toward removing the ability to produce a fair comparison between differing systems. Finally, while we have stressed that this system is highly configurable, we have not shown that the parameters with which we work are either necessary or sufficient to model all relevant workloads. Chen addresses all of these issues in his proposal of the benchmark. [1]

This only leaves the question of the care taken in specifying the benchmark. The authors claim that flexibility in comparing diverse systems and tightness in specifications are competing processes. To this end, it is our policy that configuration information and reporting procedure should not be built into the tool itself, but should be established as a standard outside of the benchmark. Should this not be possible, the specification of the use of the program should be left to the user, granting them complete control over the comparison. This meets head on two problems: First, suppose the benchmark establishes a configuration for testing. However, the system in question simply does not have the hardware or software to be put in that configuration. We have effectively eliminated the benchmark's ability to scale such that it can evaluate this system. Second, an assessment of two base systems is often times artificial. We might find use in evaluating two systems with full optimizations turned on. We do not need to limit the cases where one system can be optimized further than the others.

From this discussion, the Chen/Patterson benchmark meets our stated requirements. Having found a suitable local file-system benchmark, we can set about converting it to a network-based file-system benchmark. First, however, it is instructive to consider how the benchmark itself runs.

On execution, the program begins an exploration of the system. Specifically, several tests are run varying each parameter slowly over the course of the runs. The acceptable ranges of performance are made as narrow as possible. Inside of those acceptable ranges, a point is selected at which the throughput is 75% of its maximum value for each parameter. The collection of these all together is dubbed the focal vector. The benchmark then makes a graph of uniqueBytes and identifies the plateaus on that graph. A plateau is defined as a region where performance is roughly the same across the entire range. It is most common to have two performance regions, one where all of uniqueBytes fits into file cache and another where frequent disk accesses must be made. To allow cache to remain operational but make sure that it does not bias the data, cache is flushed in between individual runs by unmounting and remounting the file-system being evaluated. Once plateaus have been selected, the exploration phase is over, and the benchmark produces graphs depicting throughput versus the change in a characteristic (holding all other characteristics constant to the values in the focal vector). There will be one graph for each characteristic for each plateau in uniqueBytes (i.e. the usual case produces 8 graphs). For a more complete description of selecting focal vectors, see [1][2]. Note that each benchmark is run on a workload that gives its system approximately the same amount of stress, and then varies the same parameters. While the scale and quantifiable results of each run should vary, the general shape of the resultant graph should not. For a more complete description of the operation of the benchmark see [1].

# 4. Methodology

Now we turn our attention to the steps required to port the selected evaluation software to a networked file-system and the pitfalls encountered in the process. As a first step, we stress that our goal is not to produce a comparative evaluation of several systems, but to show that the techniques involved in the porting of this benchmark are valid.

Our procedure can be divided into three phases:

- Establishing a local disk baseline on our systems,
- Running a “local NFS network” version,
- Running a benchmark on a full-scale network installation.

## 4.1. Local disk machines

Clearly, this serves as our control group. It was run on an Intel PIII-667 with 128 MB of main memory running the RedHat Linux operating system, version 6.2 (kernel version 2.2.14). The complete results of these trials are included in the appendix.

To begin trying to anticipate potential dilemmas, we consider how we would have to modify a program so that it works over NFS instead of from the local disk. The answer to this is that there are very few considerations that need to be taken. Great pains were taken with the NFS protocol to ensure an implementation transparent to the user. The only really pressing issue comes with the fact that for performance improvement, we tend to cache everywhere we possibly can. The question of how to flush the cache on a remote machine must now be decided. Techniques for dealing with this problem are discussed later in this section.

### 4.2. Local NFS

By this we mean a system running the NFS protocol such that the client machine is its own server. The hardware for this test is identical to the hardware for the local disc machine. This perturbation serves to isolate several possible points of trouble. First, should there be a performance hit at this stage of our testing, it can only be due to the execution of the NFS protocol. There is no possibility of network delay, as the request never leaves the local Network Interface Controller. Note that even though the implementation may try and maintain both server and client cache, they are taken from the same memory system. Hence the problem of flushing the cache is easily dealt with by modifying the script to umount both the server access to the block device and the client reference to the file system maintained on that device. The full results from these runs can be found in the appendix.

### 4.3. NFS Networks

Here we deal with the issue of moving the client and server to different machines. We utilized an SGI Indy with one 150Mhz R5000 processor, 192MB of main system memory, running Irix 6.5 as our client machine connected via 10Mb/s Ethernet. We utilized an SGI Origin 2000 with 32 300Mhz R12000 processors, 16GB of main system memory, running Irix 6.5 as our server machine connected via 100Mb/s Ethernet. Note that while this will still scale to the system and remain I/O limited, the bottleneck could now be due to the network rather than the disk system. The other issue of importance here is flushing the server cache. The most straightforward solution found by the authors is to divide our script from the local NFS trials into two separate scripts:

- Edit the one on the client so that it unmounts the client file system and invokes the script on the server via RPC.
- Edit the server script so that it umounts the device and remounts it.
- When the RPC call returns on the client side, have it remount the client file system.

This is the natural solution, as NFS is written over RPC. Thus we have a priori knowledge that all of the tools to carry out this task are in place on our system. The full results from these runs can be found in the appendix.

# 5. Results

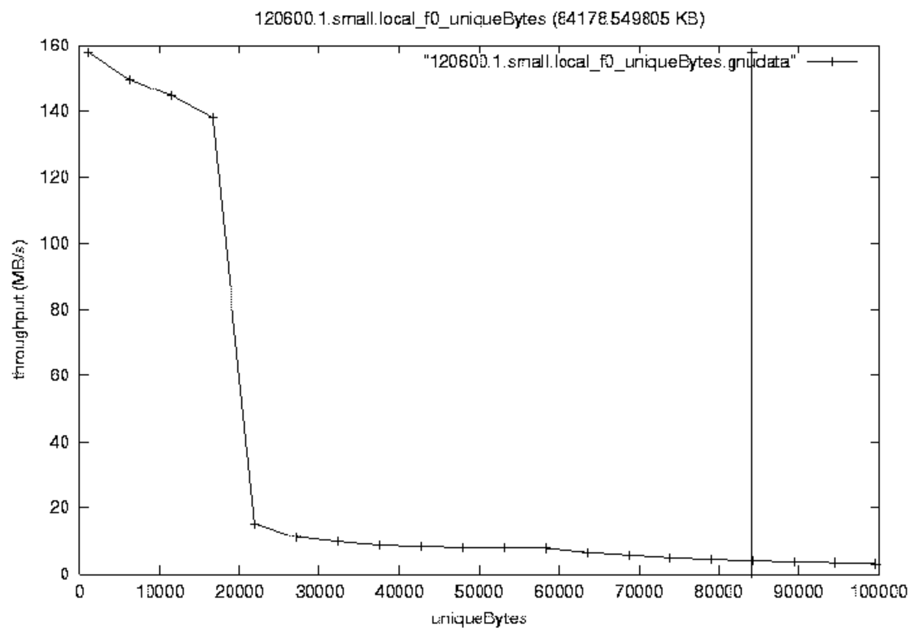


**Figure 1 - Local File System UniqueBytes**

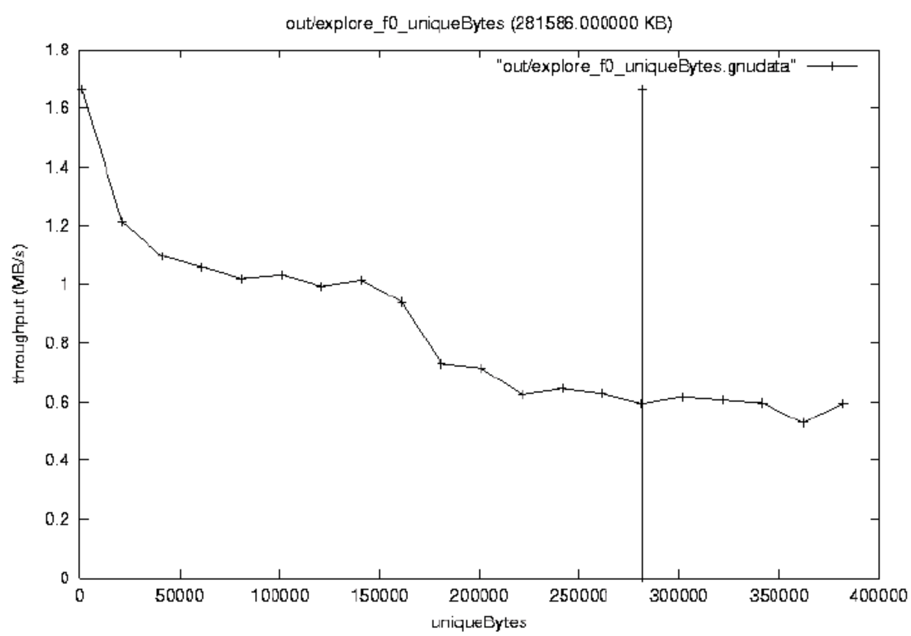


**Figure 2 - Irix NFS UniqueBytes**

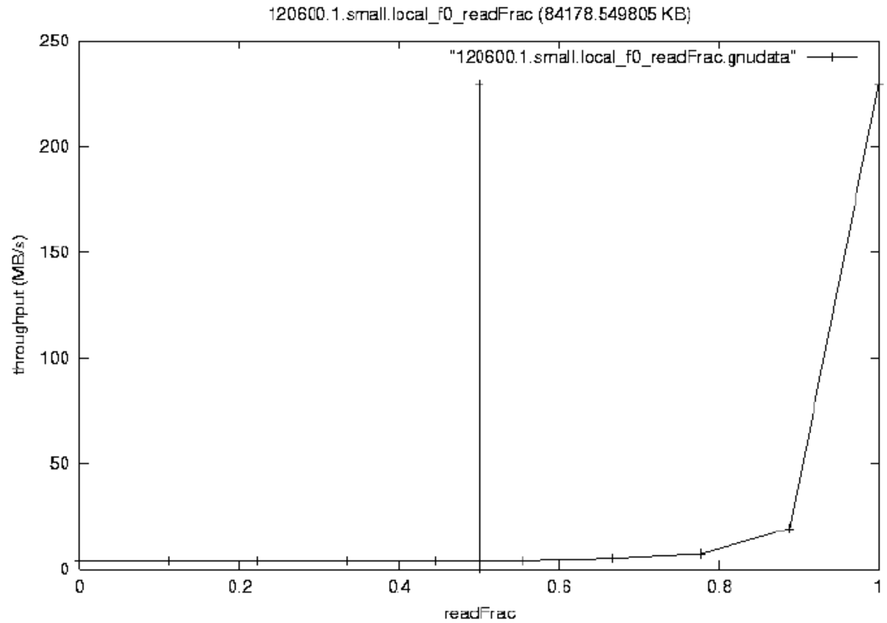


**Figure 3 - Local File System ReadFrac**

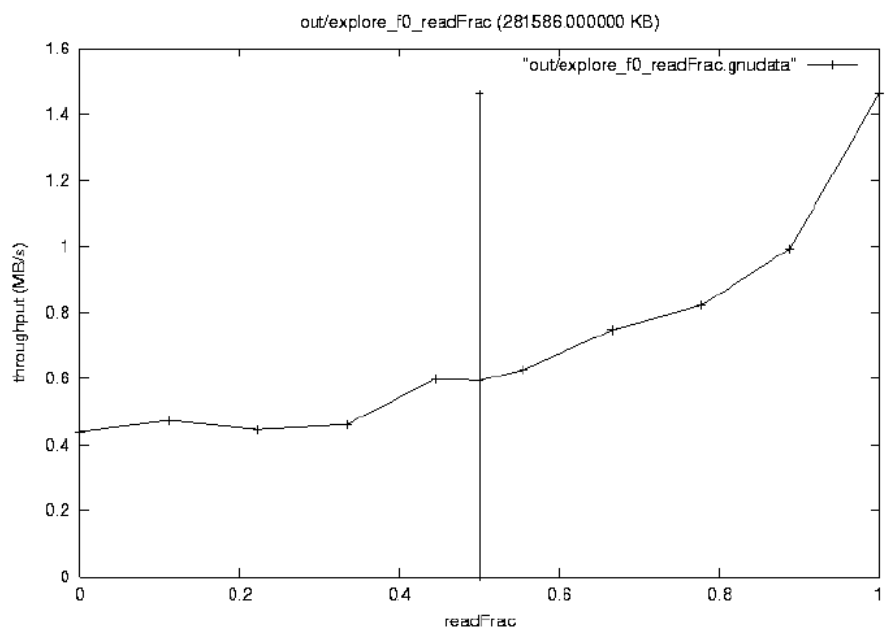


**Figure 4 - Irix NFS ReadFrac**

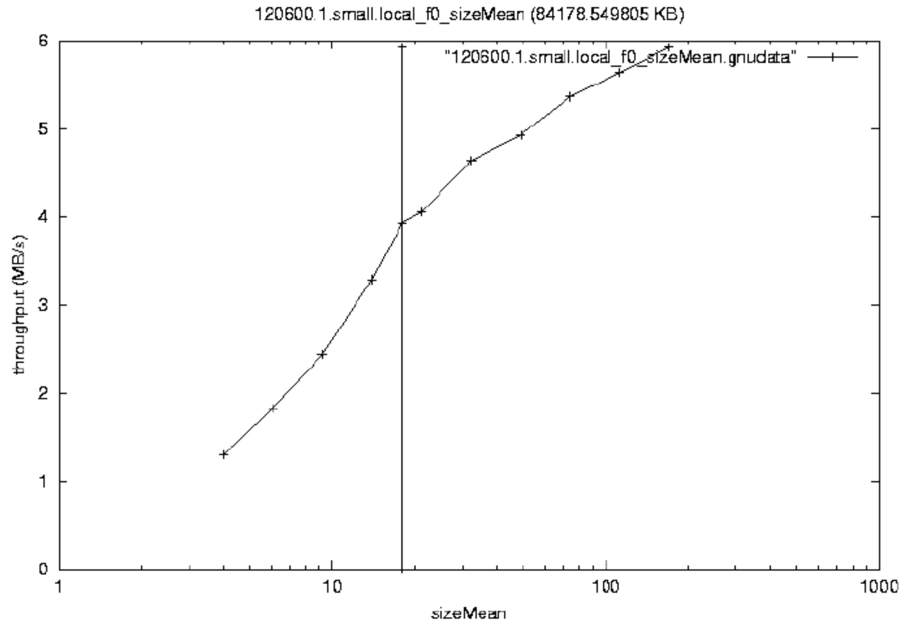


**Figure 5 - Local File System SizeMean**

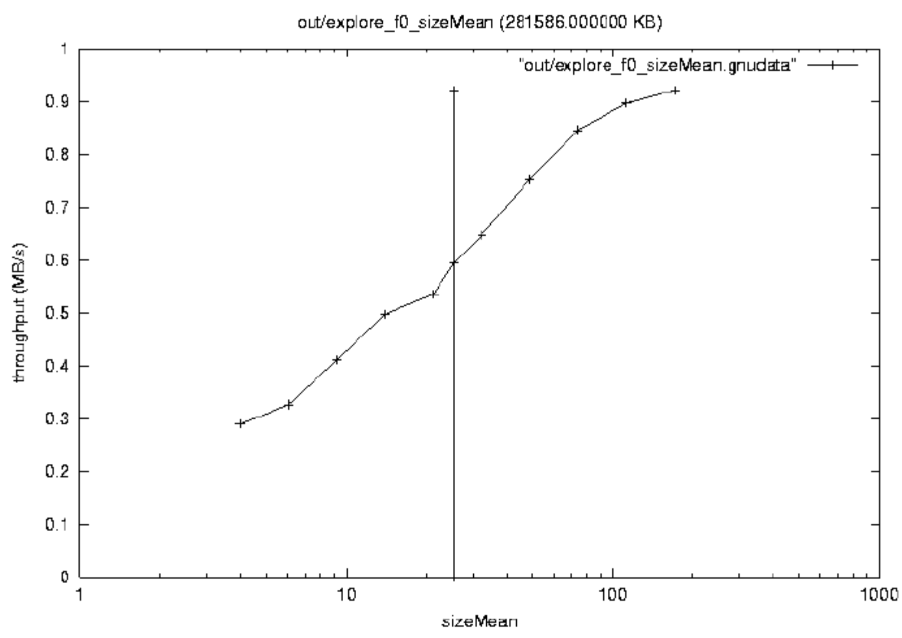


**Figure 6 - Irix NFS SizeMean**

The graphs above are from the results of the benchmarks run on a local Linux ext2fs file-system and an Irix network running NFS version 3 (NFS3). Though requests over the network produce rougher results, you can see that the graphs all retain approximately the same shape. Figure 1 shows the expected results with very high throughput until the edge of disk cache is reached, and then more modest results. Figure two is a bit more problematic. Irix will allow all of main memory to be used as the file cache. This explains the dip just shy of 200 MB (since our client has 192MB of available memory). The spike on the low edge (<50 MB) is believed to be a spurious result due to lack of sampled data. Note there are no data points between this small request and the next one where a steady state has been reached, so the cause of this is inconclusive. However, the initial transmission could've been small enough that throughput was miscalculated. Throughput is consistent with 10 Mb/s Ethernet between 50 MB and 192 MB. In figures 3 and 4, we can compare the effects of readFrac on both systems. Keeping in mind that reads are cached and writes are not, the results make a lot of sense. As more reads than writes are requested, performance gradually increases. When there is an intervening network (figure 4), the effects of cache are much less pronounced. Notice again we have the same general shape. Figures five and six show that as the average size of each request increases, the throughput increases. Again, the overall performance is relatively low through the network, though the curves are basically the same shape.

This idea of similar shaped graphs is a key idea in self-scaling benchmarks. The performance graphs are produced from different workloads. Hence, comparing them really does not tell us anything about how the two systems compare to one another in general.

However, we can make the simplifying assumption that these similar shapes are due to the fact that the effects of the shape of one parameter are independent of the value of the other parameters. This allows us to calculate values for workloads we did not actually measure. We do this by selecting a set of parameters we would like to know the throughput for (say the value of the focal point on the other machine). We can then look up the value of the throughput at each parameter in the focal point on the graph where that parameter is not held constant. Multiply the throughput on the graph for which we are trying to generate the unmeasured value by the ratio of the values on the two graphs. This will give us an intercept from which we can extrapolate the new curve. In this way, we have reclaimed the ability to compare systems on which we have measured different workloads. For a more complete description on performance-prediction, see [1][5].

## 6. Future Work

There are several interesting questions on this topic that are still unanswered. Issues that have yet to be addressed include:

- How is performance impacted by the presence of another server on the network? Would the network competition between two servers and multiple clients significantly affect the performance of the protocol?

- The entire experiment needs to be replicated for the Andrew File System (AFS). The same can be said for the especially interesting cross-platform protocol SAMBA. Further, there is the related possibility of benchmarking Network Attached Storage Devices (NASD), or any other distributed file system of your choosing.
- Of course, now that a feasibility study has been completed, there is the question of doing the actual comparison of networked systems with local disk systems using this benchmark.

## 7. Conclusions

In the midst of the current push towards Networked File-Systems, the ability to evaluate the performance differences between networked and local file-systems becomes ever more desirable. The implementers of NFS knew that there would be an inherent impact on performance involved with using a networked file-system. The time for quantifying that decrease in performance has come, and as we have presented here, reliably carrying this out is not an unreasonable task.

# References


[1] Chen, P. M., and Patterson, D. A.”A New Approach to I/O Performance Evaluation – Self-Scaling I/O Benchmarks, Predicted I/O Performance”, *ACM Transactions on Computer systems*, Vol. 12 No. 4, November 1993, 308-339.

[2] Chen, P. M., and Patterson, D. A. “Storage Performance – Metrics and Benchmarks”, *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Santa Clara, California, May 1993, 1-12.

[3] Callaghan, B., Pawlowski, B., and P. Staubach, “NFS Version 3 Protocol Specification”, *Network Working Group RFC 1813*, June 1995.

[4] Miller, Ethan, “Towards Scalable Benchmarks for Mass Storage Systems”, *Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, College Park, MD, September 1996, pages 515-528.

[5] Saavedra-Barrera, et al., “Machine characterization based on an abstract high-level language machine”, *IEEE Trans. Comput.* 38, December 1989, 1659-1679.

[6] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., “Design and Implementation of the Sun Network File-system” *Proceedings of the Summer 1985 USENIX Conference*, Portland OR, June 1985, pp. 119-130.

[7] Wittle, Mark and Keith, Bruce, “LADDIS: The Next Generation In NFS File Server Benchmarking”, *Proceedings of the USENIX Summer 1993 Technical Conference*, Cincinnati, Ohio, June 1993.