A New Approach to I/O Performance Evaluation — Self-Scaling I/O Benchmarks, Predicted I/O Performance

PETER M. CHEN
University of Michigan
and
DAVID A. PATTERSON
University of California at Berkeley

Current I/O benchmarks suffer from several chronic problems: they quickly become obsolete; they do not stress the I/O system; and they do not help much in understanding I/O system performance. We propose a new approach to I/O performance analysis. First, we propose a self-scaling benchmark that dynamically adjusts aspects of its workload according to the performance characteristic of the system being measured. By doing so, the benchmark automatically scales across current and future systems. The evaluation aids in understanding system performance by reporting how performance varies according to each of five workload parameters. Second, we propose predicted performance, a technique for using the results from the self-scaling evaluation to estimate quickly the performance for workloads that have not been measured. We show that this technique yields reasonably accurate performance estimates and argue that this method gives a far more accurate comparative performance evaluation than traditional single-point benchmarks. We apply our new evaluation technique by measuring a SPARCstation 1+ with one SCSI disk, an HP 730 with one SCSI-II disk, a DECstation 5000/200 running the Sprite LFS operating system with a three-disk disk array, a Convex C240 minisupercomputer with a four-disk disk array, and a Solbourne 5E/905 fileserver with a two-disk disk array.

Categories and Subject Descriptors: B.4.2 [**Input / Output and Data Communications**]: Input/Output Devices—*channels and controllers*; B.4.4 [**Input / Output and Data Communications**]: Performance Analysis and Design Aids; C.4 [**Computer Systems Organization**]: Performance of Systems; D.4.2 [**Operating Systems**]: Storage Management; D.4.3 [**Operating Systems**]: File Systems Management; D.4.4 [**Operating Systems**]: Communications Management

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Benchmark, client-server, disk array, file cache, input/output, RAID

This research was supported in part by NASA/DARPA grant NAG 2-591 and NSF grant MIP 8715235.
Authors' addresses: P. M. Chen, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122; email: pmchen@eecs.umich.edu; D. A. Patterson, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720; email: pattrsn@cs.berkeley.edu.

1. INTRODUCTION

As processors continue to improve their performance faster than I/O devices [Patterson et al. 1988], I/O will increasingly become the system bottleneck. There is therefore an increased need to understand and compare the performance of I/O systems, hence the need for I/O-intensive benchmarks. The benefits of good benchmarks are well understood—when benchmarks are representative of users' applications, they channel vendor optimization and research efforts into improvements that benefit users. Good benchmarks assist users also in purchasing machines by allowing fair, relevant comparisons.

Recent efforts to standardize benchmarks, such as SPEC [Scott 1990] and Perfect Club [Berry et al. 1989], have increased our understanding of computing performance and helped create a fair playing field on which companies can compete. These standardization efforts have focused on CPU-intensive applications, however, and intentionally avoided I/O-intensive applications.

In this article, we develop criteria for ideal I/O benchmarks and show how current I/O benchmarks fall short of these. We describe a new approach to I/O benchmarks—*a self-scaling benchmark*, which dynamically adjusts its workload to the system being measured, and *predicted performance*, which estimates the performance for unmeasured workloads based on the performance from a small set of measured workloads. The self-scaling benchmark reports how performance varies with each of five workload parameters. This helps evaluators to understand systems and helps users to choose systems that perform well for their workload. Predicted performance allows performance evaluators to accurately estimate the performance one could expect on a workload different than the exact ones measured in standard benchmarks.

2. THE IDEAL I/O BENCHMARK

In this article, an I/O benchmark measures the data I/O performance seen by an end user issuing reads and writes. Specifically, we are *not* trying to measure the performance of file system commands, such as deleting files, making directories, or opening and closing files. This definition dictates that we issue user I/O requests, which in UNIX typically go through the file or buffer cache.

The ideal I/O benchmark will have several characteristics. First, a benchmark should help system designers and users understand why the system performs as it does. Computer architects and operating system programmers need benchmarks to evaluate design changes and isolate reasons for poor performance. Users should be able to use benchmarks as well to understand optimal ways to use the machine. For instance, if a user wanted to avoid thrashing the file cache, the ideal I/O benchmark should be able to provide information on the file cache size for any machine. This criteria may require reporting results for several different workloads, enabling the user to compare these results. These multiple workloads should require little human interaction to run.

Second, to maintain the focus of measuring and understanding I/O systems, an I/O benchmark should be I/O limited. By our definition of an I/O benchmark, this implies that most of the time should be spent doing data I/O. In systems that mask response time with read prefetching or write-behind, *I/O limited* implies that taking out all the reads and writes should decrease running time more than taking out all non-I/O components.

Third, the ideal I/O benchmark should scale gracefully over a wide range of current and future machines. Without a well-planned scaling strategy, I/O benchmarks become obsolete quickly as machines evolve. For instance, *IOStone* tries to exercise the memory hierarchy but touches only 1 MB of user data. Perhaps at the time IOStone was written 1 MB was a lot of data, but no longer. One recent example of how I/O systems are evolving is disk arrays [Patterson et al. 1988]. Disk arrays allow multiple I/Os to be in progress simultaneously. Most current I/O benchmarks do not scale the number of processes issuing I/O, and hence are unable to properly stress disk arrays. Unfortunately, it is difficult to find widespread agreement on a scaling strategy, especially for benchmarks intended for a wide range of audiences.

Fourth, a good I/O benchmark should allow fair comparisons across machines. This comparison has two aspects. First, a fair comparison across machines should be able to be made for I/O workloads identical to the benchmark. However, users rarely have the same workload as a standard benchmark. Thus, it should be possible also to use the results from a benchmark to make meaningful comparisons for workloads that differ from the benchmark.

Fifth, the ideal I/O benchmark would be relevant to a wide range of applications. It is certainly easier to target a benchmark to a specific audience, but it would be better for a benchmark to be usable by many audiences.

Finally, for results to be meaningful, benchmarks must be tightly specified. Results should be reproducible; optimizations that are allowed and disallowed must be explicitly stated; the machine environment on which the benchmarking takes place must be well defined and reported, and so on. In this article, we leave this aspect of benchmarking to standardization organizations such as SPEC [Scott 1990] and the Transaction Processing Performance Council [1989; 1990].

In summary, the six characteristics of the ideal I/O benchmark are as follows: it should help in understanding system performance; it should be I/O limited; it should scale gracefully over a wide range of current and future machines; it should allow fair comparisons across machines; it should be relevant to a wide range of applications; and it should be tightly specified.

3. CURRENT I/O BENCHMARKS

In this section, we examine current benchmarks used to evaluate I/O systems. The benchmarks we consider are Andrew [Howard et al. 1988], TPC-B [Anon et al. 1985; TPPC 1989; 1990], Sdet [Gaede 1981; 1982; SPEC 1991b;

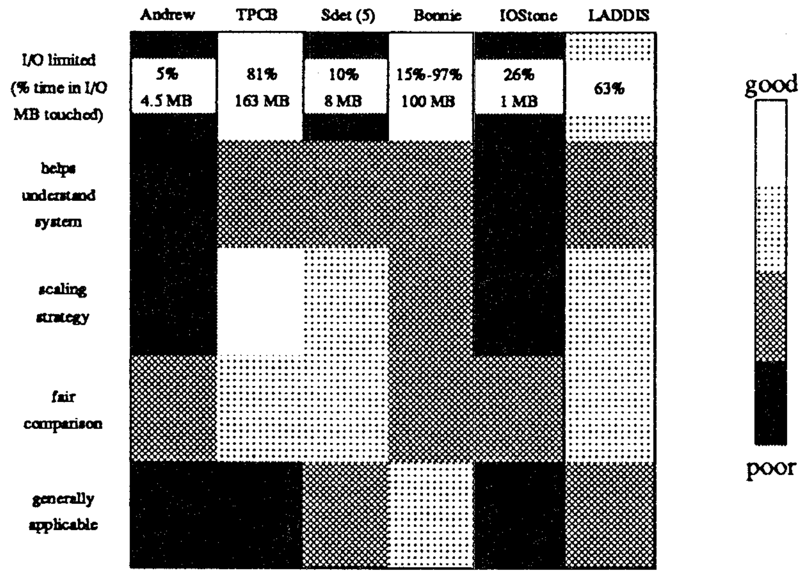


Fig. 1. Current state of I/O benchmarks. In this figure, we show a qualitative evaluation of benchmarks used today to evaluate I/O systems. We see that several are not I/O bound and that most do not provide understanding of the system, lack a well-defined scaling strategy, and are not generally applicable. The percent time spent in I/O was measured on the DECstation 5000/200 of Figure 2. LADDIS was not available for execution at this time, but a prerelease beta version spends 63% of its execution time doing reads and write; the rest of the time is spent in other NFS operations, such as lookup (17%) and getattr (6%).

1991a], Bonnie [Bray 1990], and IOStone [Park and Becker 1990].[1] Of these, only Bonnie and IOStone specifically focus on measuring I/O performance. Andrew is meant as a convenient yardstick for measuring file system performance; TPC-B is a transaction-processing benchmark; Sdet, part of the SPEC System Development Multiuser (SDM) Suite, is designed to measure system throughput in a multitasking software development environment. Although some of these benchmarks are not focused solely on measuring I/O performance, they are nonetheless used today in I/O performance evaluation. In applying our list of benchmark goals from the previous section to current I/O benchmarks, we see that there is much room for improvement. We show a qualitative evaluation of today's I/O benchmarks in Figure 1 and make the following observations:

—*Many I/O benchmarks are not I/O limited.* On a DECstation 5000/200 (Figure 2) running the Sprite operating system [Ousterhout et al. 1988], Andrew, Sdet,[2] and IOStone spend 25% or less of their time doing I/O.

[1] LADDIS, a new benchmark being developed under SPEC, is not yet available for public performance disclosure. We include it in our qualitative critique of benchmarks, however (Figure 1). The only other I/O benchmark known to the authors is the AIM III Suite, whose code was not available. The AIM III suite is similar to Sdet in that it scales the number of scripts running in parallel but no other workload parameter. Each simultaneously running script uses 3.5 MB.
[2] This refers to Sdet running at the peak throughput concurrency level of 5.

System Name	SPARCstation 1+	DECstation 5000/200	HP 730
Year Released	1989	1990	1991
CPU	SPARC	MIPS R3000	PA-RISC
SPECmarks	8.3	19.9	76.8
Disk System	CDC Wren IV	3 Wrens (RAID 0)	HP 1350SX
I/O Bus	SCSI-I	SCSI-I	Fast SCSI-II
Mem. Peak Speed	80 MB/s	100 MB/s	264 MB/s
Memory Size	28 MB	32 MB	32 MB
Operating System	SunOS 4.1	Sprite LFS	HP/UX 8.07

System Name	Convex C240	Solbourne 5E/905
Year Released	1988	1991
CPU	C2 (4 processors)	SPARC (5 processors)
Speed	220 MIPS	22.8 SPECint
Disk System	4 DKD-502 RAID 5	2 Seagate IPI
I/O Bus	IPI-2	IPI-2
Mem. Peak Speed	200 MB/s	128 MB/s
Memory Size	1024 MB	384 MB
Operating System	ConvexOS 10.1 (BSD derived)	SunOS 4.1A.2 (revised)

Fig. 2. System platforms. This table shows the five systems on which we run benchmarks. The DECstation [DEC 1990] uses a three-disk RAID disk array [Patterson et al. 1988] with a 16-KB striping unit [Chen and Patterson 1990] and is configured without redundancy. The SPECmark rating is a measure of the processor speed; ratings are relative to the speed of a VAX 11/780. The full name of the HP 730 is the HP Series 700 Model 730 [Hewlett-Packard 1992].

Further, many of the benchmarks touch very little data. IOStone touches only 1 MB of user data; Andrew touches only 4.5 MB.

—*Today's I/O benchmarks do not help much in understanding system performance.* Andrew and IOStone give only a single bottom-line performance result. TPCB and Sdet fare somewhat better by showing the user system performance under various loads. Bonnie begins to help the user understand performance by running six different workloads. These workloads show the performance differences between reads versus writes and block versus character I/O, but do not vary other aspects of the workload, such as the number of I/O occurring in parallel.

—*Most of today's I/O benchmarks have no general scaling strategy.* Several make no provision for adjusting the workload to stress machines with larger file caches, for example. Without a well-defined scaling strategy, I/O benchmarks grow obsolete quickly. Several exceptions are notable. TPC-B has an extremely well-defined scaling strategy, made possible by TPC-B's narrow focus on debit-credit style transaction processing and the widespread agreement on how databases change with increasing database throughput. Sdet has a superior scaling strategy also, varying the number of simultaneously active scripts until the peak performance is achieved. This idea of scaling aspects of the workload automatically is a major improvement over single-workload benchmarks. However, Sdet does not scale any other aspects of the benchmark, such as request size or read/write ratio. LADDIS, when formally defined, will likely have a scaling strategy similar to Sdet. It will probably scale a few workload parameters, such as disk space or number of clients, but will leave other parameters fixed.

—*Today's I/O benchmarks make fair system comparisons for workloads identical to the benchmark but do not help in drawing conclusions about the relative performance of machines for other workloads*. It would be useful if results form the benchmark could be applied to a wider range of workloads.

—*Today's I/O benchmarks focus on a narrow application range*. For example, TPC-B is intended solely for benchmarking debit-credit transaction-processing systems.

4. A NEW APPROACH FOR I/O BENCHMARKS—AN OVERVIEW

We propose two new ideas in I/O benchmarks. First, we propose a benchmark that scales its workload automatically to the system being measured. During evaluation, the benchmark explores the workload space automatically, searching for a relevant workload on which to base performance graphs.

Because the base workload resulting from self-scaling evaluation depends on the characteristics of each system, we lose the ability to directly compare performance results for multiple systems. We propose using *predicted performance* to restore this ability. Predicted performance uses the results of the self-scaling benchmark to estimate performance for unmeasured workloads. The ability to accurately estimate performance for arbitrary workloads has several advantages. First, it allows fairer comparisons to be drawn between machines for their intended use—today, users are forced to apply the relative performance from benchmarks that may be quite different from their actual workload. Second, the results can be applied to a much wider range of applications than today's benchmarks. Of course, the accuracy of the prediction determines how effectively prediction can be used to compare systems. We explore the method and accuracy of prediction in Section 9.

5. WORKLOAD MODEL

The workload that the self-scaling evaluation uses is characterized by five parameters. These parameters lead to the first-order performance effects in I/O systems. See Figure 4 for examples of each parameter.

(1) **uniqueBytes**—the number of unique data bytes read or written in a workload, essentially the total size of the data.

(2) **sizeMean**—the average size of an I/O request. We choose sizes from a normal distribution with a coefficient of variation equal to 1.

(3) **readFrac**—the fraction of reads; the fraction of writes is 1–*readFrac*.

(4) **seqFrac**—the fraction of requests that follow the prior request sequentially. For workloads with multiple processes, each process is given its own thread of addresses.

(5) **processNum**—the concurrency in the workload, that is, the number of processes simultaneously issuing I/O.

Fig. 3. Representativeness of workload. This table shows how accurately our synthetic workload mimics the performance of two I/O-bound applications, Sort and TPC-B. All runs were done on a DECstation 5000 running Sprite. The input to sort was four files totaling 48 MB.

Application	Throughput	Response Time
Sort	0.20 MB/s	11.7 ms
Workload Model	0.20 MB/s	11.0 ms
TPC-B	0.13 MB/s	14.0 ms
Workload Model	0.13 MB/s	12.3 ms

In this article, a *workload* refers to a user-level program with parameter values for each of the above five parameters. This program spawns and controls several processes if necessary.

The most important question in developing a synthetic workload is the question of representativeness [Ferrari 1984]. A synthetic workload should have enough parameters such that its performance is close to that of an application with the same set of parameter values.[3] To show that our workload captures the important features of an I/O workload, Figure 3 compares the performance of two I/O-bound applications to the performance of the synthetic workload with those two applications' parameter values. We see that both Sort and TPC-B can be modeled quite accurately. Throughput and response time are both accurate within a few percent. This accuracy increases our confidence that the parameters of the synthetic workload capture the first-order performance effects of an I/O workload.

6. SINGLE-PARAMETER GRAPHS

Most current benchmarks report the performance for only a single workload. The better benchmarks report performance for multiple workloads, usually in the form of a graph. TPC-B and Sdet, for example, report how performance varies with load. But even these better benchmarks do not show in general how performance depends on parameters such as request size or the mix of reads and writes.

The main output of our self-scaling benchmark is a set of performance graphs, one for each parameter (uniqueBytes, sizeMean, readFrac, processNum, and seqFrac) as in Figure 4. While graphing one parameter, all other parameters remain fixed. The value at which a parameter is fixed while graphing other parameters is called the *focal point* for that parameter. The vector of all focal points is called the *focal vector*. In Figure 4, for example, the focal vector is {uniqueBytes = 21 MB, sizeMean = 10 KB, readFrac = 0, processNum = 1, seqFrac = 0}. Hence, in Figure 4(a), uniqueBytes is varied while the sizeMean = 10 KB, readFrac = 0, processNum = 1, and seqFrac = 0.

[3] Of course, given the uncertain path of future computer development, it is impossible to determine a priori all the possible parameters necessary to ensure representativeness. Even for current systems, it is possible to imagine I/O workloads that interact with the system in such a way that no synthetic workload (short of a full trace) could duplicate that I/O workload's performance.

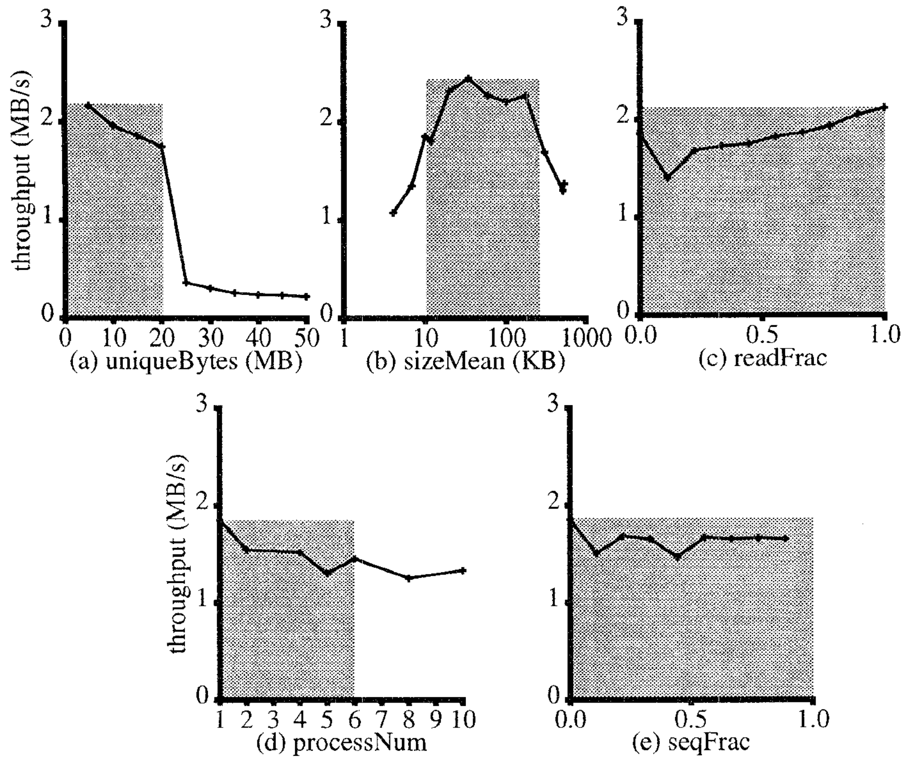


Fig. 4. Results from a self-scaling benchmark that scales all parameters. In this figure, we show results from a self-scaling benchmark of a SPARCstation 1 with 28 MB of memory and a single SCSI disk. The benchmark reports the range of workloads, shown as the shaded region, which perform well on this system. For example, this SPARCstation performs well if the total number of unique bytes touched is less than 20 MB. It also shows how performance varies with each workload parameter. Each graph varies exactly one parameter, keeping all other parameters fixed at their focal point. For these graphs, the *focal point* is the point at which all parameters are simultaneously at their 75% performance point. The 75% performance point for each parameter is defined to be the least restrictive workload value that yields at least 75% of the maximum performance. The range of workloads which perform well (shaded) is defined as the range of values that yields at least 75% of the maximum performance. The 75% performance point found by the benchmark for each parameter is uniqueBytes = 21 MB, sizeMean = 10 KB, readFrac = 0, processNum = 1, seqFrac = 0.

Figure 5 illustrates the workloads reported by one set of such graphs for a three-parameter workload space. Although these graphs show much more of the entire workload space than current benchmarks, they still show only single-parameter performance variations; they do not display dependencies between parameters. Unfortunately, completely exploring the entire five-dimensional workload space requires far too much time. For example, an orthogonal sampling of six points per dimension requires 6^5, almost 8000, points. On the Sprite DECstation, each workload takes approximately 10 minutes to measure; thus 8000 points would take almost 2 months to gather! In contrast, measuring six points for each graph of the five parameters

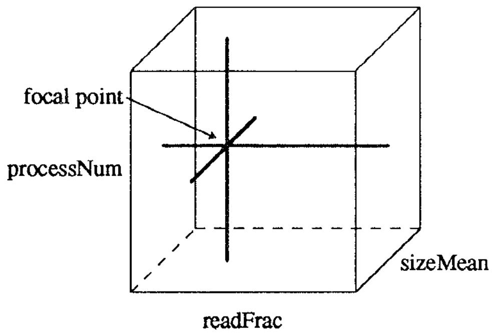


Fig. 5. Workloads reported by a set of single-parameter graphs. This figure illustrates the range of workloads reported by a set of single-parameter graphs for a workload of three parameters.

requires only 30 points and 5 hours. The usefulness of these single-parameter graphs depends entirely on how accurately they characterize the performance of the entire workload space. In the section on *predicted performance* we shall see that, for a wide range of I/O systems, the *shapes* of these performance curves are relatively independent of the specific values of the other parameters.

7. FIRST TRY — SELF-SCALING ALL WORKLOAD PARAMETERS

A self-scaling benchmark is one that adjusts the workloads that it runs and reports based on the capabilities of the system being measured. Sdet and TPC-B both do this for one aspect of the workload, that is, load (processNum) [SPEC 1991a; TPPC 1990]. Sdet reports the maximum throughput, which occurs at different loads for different systems. TPC-B reports maximum throughput subject to a response time constraint; this also occurs at different loads for different systems. This section describes our first attempt to create a self-scaling benchmark by generalizing this type of scaling to all workload parameters.

The basic idea behind TPC-B and Sdet's attempts to avoid obsolescence is to scale one aspect of the workload load, based on what performs well on a system. Similarly, as we vary any one of our five workload parameters (uniqueBytes, sizeMean, readFrac, processNum, and seqFrac), we can search for a parameter value that performs well on the system. Of course, the parameter value must be practically achievable (this rules out infinitely large request sizes and extremely small values for uniqueBytes). One way to choose this value is to look for a point in a parameter's performance curve that gives good, but not maximum, performance. In this section, we set this performance point at 75% of the maximum performance.

Using a simple iterative approach, it is possible to find a focal vector for which each workload parameter is simultaneously at its 75% performance point [Chen 1992]. Figure 4 shows results from a benchmark that self-scales all parameters. The system being measured is the one-disk SPARCstation of Figure 2. In Figure 4 the shaded region on each graph is the range of workloads that perform well for this system, that is, the workload values that yield at least 75% of the maximum performance. When a parameter is varied, the other parameters are fixed at their focal point chosen by the benchmark (uniqueBytes = 21 MB, sizeMean = 10 KB, readFrac = 0, processNum = 1, seqFrac = 0). These are the least restrictive[4] values in the range of workloads that perform well. Conclusions that these results help us reach about this system are as follows:

—The effective file cache size is 21 MB. Applications that have a working set larger than 21 MB will go to disk frequently.

—I/O workloads with larger average sizes yield higher throughput. To get reasonable throughput, request sizes should be at least 10 KB but no larger than 200 KB. This information may help operating system programmers choose the best block size.

—Workloads with almost all reads perform slightly better than workloads with more writes, but there is not much difference.

—Increasing concurrency does not improve performance. As expected, without parallelism in the disk system, workload parallelism is of little value.

—Sequentiality does not affect performance at the global 75% performance point.

Self-scaling all parameters gives interesting results and helps us understand a system. However, there are several problems with self-scaling all workload parameters. First, the iterative process of finding the global 75% performance point may be slow. Second, there are really two criteria in choosing the focal vector point for a parameter. Take readFrac as an example. The first criterion is what has been mentioned above—scaling readFrac to the value that performs well on this system. But, there is also a second criterion: *the performance curves while readFrac is at its focal point should apply to workloads where readFrac differs from its focal point*. In other words, the shape of the performance curves at the focal point of readFrac should be representative of the shape of the performance curves at other values of readFrac. This can preclude choosing an extreme value. To illustrate, if reads were 100 times faster than writes, self-scaling all parameters might pick readFrac = 1.0 as the focal point. But, readFrac's focal point of 1.0 may yield performance graphs for the other parameters that do not apply to workloads with some writes. It would be better to pick an intermediary value as the focal point than the 75% performance point. In fact, as we shall see in the section on predicted performance, general applicability is a more important criterion than scaling to what performs well, because, if the shape of the performance curve is generally applicable, we can use it to estimate performance for any workload.

The third problem is that the uniqueBytes parameter often has distinct performance regions. These regions correspond to the various storage hierar-

[4] Least restrictive refers to workloads that are either easier for programmers to generate or easily transformable from more restrictive workloads. For instance, programmers can always break down a large request into several, smaller requests; thus smaller sizes are less restrictive than larger sizes.

chies in the system. In Figure 4(a), uniqueBytes smaller than 21 MB uses the file cache primarily, while uniqueBytes larger than 21 MB uses the disk primarily. When uniqueBytes is on the border between the file cache and disk region, performance is often unstable—small changes in the value of uniqueBytes can lead to large changes in performance. The 75% performance point is usually on the border between the storage hierarchy regions, so choosing the focal point for uniqueBytes to be that point makes it likely that performance will be unstable. Graphing in the middle of a hierarchy level's performance region should be more stable than graphing on the border between performance regions. Another problem that arises from the storage hierarchy regions is that each level of the storage hierarchy may give different performance shapes. Figure 8 shows how, depending on the storage hierarchy region, reads may be faster than writes (uniqueBytes 15 MB), slower than writes (uniqueBytes 36 MB), or about the same speed as writes (uniqueBytes 2MB).

8. A BETTER SELF-SCALING BENCHMARK

There are a variety of solutions to the problems listed above. For the distinct performance regions uncovered by uniqueBytes, we measure and report multiple families of graphs, one family for each performance region (Figure 4 reported a single family of graphs). For instance, the first family of graphs is shown in Figure 7(a)–(d) and has uniqueBytes = 12 MB. The second family of graphs, shown in Figure 7(f)–(i), has a separate focal point with uniqueBytes = 42 MB. The self-scaling benchmark delineates performance by measuring the slope of the uniqueBytes curve (Figure 6).

To improve general applicability of the graphs, we choose the focal point of each parameter to be more in the "middle" of its range. For parameters such as readFrac and seqFrac, the range is easily defined (0 to 1); hence a midpoint of 0.5 is the chosen focal point. The remaining parameters are uniqueBytes, sizeMean, and processNum. For each performance region, the focal point for uniqueBytes is set at the middle of that region. For the last two parameters, sizeMean and processNum, the focal point is set at the value that yields performance half-way between the minimum and maximum.

After these solutions and modifications the revised program is called simply the *self-scaling benchmark*.

8.1 Examples

This section contains results from running the self-scaling benchmark on the five systems described in Figure 2.

8.1.1 *SPARCstation* 1+. Figure 7 shows results from the self-scaling benchmark on the SPARCstation 1+. The uniqueBytes values that characterized the two performance regions are 12 MB and 42 MB. Figure 7(a)–(d) show the file cache performance region, measured with uniqueBytes = 12 MB. Figure 7(f)–(i) show the disk performance region, measured with uniqueBytes = 42 MB. In addition to what we learned from our first self-

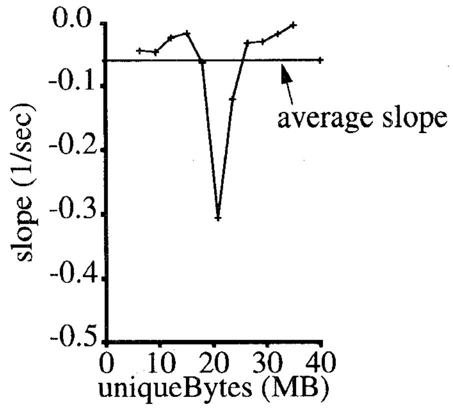


Fig. 6. Slope of uniqueBytes curve for SPARCstation 1+. This figure plots the slope of the uniqueBytes curve (Figure 7). The one area of the graph with slope below the average slope (−0.06 per second) is near 20 MB. The self-scaling benchmark uses this information in a heuristic algorithm to delineate performance regions.

scaling benchmark, we see the following:

—Larger request sizes yield higher performance. This effect is more pronounced in the disk region.

—Reads are faster than writes, even when all the data fits in the file cache (Figure 7(b)). Although the data fits in the file cache, writes still cause i-node changes to be written to disk periodically for reliability in case of a system crash. This additional overhead for writing causes writes to be slower than reads.

—Sequentiality offers no benefit in the file cache region (Figure 7(d)) but offers substantial benefit in the disk region (Figure 7(i)).

8.1.2 *DECstation* 5000/200. Figure 8 shows selected self-scaling benchmark results for the DECstation 5000/200 (a full set of graphs are in the Appendix, Figures 18 and 19). The uniqueBytes graph (Figure 8(a)) shows three performance plateaus, uniqueBytes = 0 to 5 MB, uniqueBytes = 5 to 20 MB, and uniqueBytes > 20 MB. Thus, the self-scaling benchmark gathers three sets of measurements: at uniqueBytes 2 MB, 15 MB, and 36 MB. The most interesting phenomenon involves readFrac (Figure 8(b)).

In the first performance level (uniqueBytes = 2 MB), reads and writes are the same speed. At the next performance level (uniqueBytes = 15 MB), reads are much faster than writes. This is due to the effective write cache of Sprite's LFS being much smaller than the read cache, so reads are cached in this performance region while writes are not. The write cache of LFS is smaller because LFS limits the number of dirty cache blocks to avoid deadlock during cleaning. The effective file cache size for writes is only 5–8 MB, while for reads it is 20 MB (personal communication, M. Rosenblum, July 1992).[5] In contrast, when uniqueBytes is large enough to exercise the disk for both reads and writes, writes are faster than reads. This phenomenon is due

[5] The default limit was tuned for a machine with 128 MB of memory; in production use, this limit would be changed for the 32-MB system being tested.

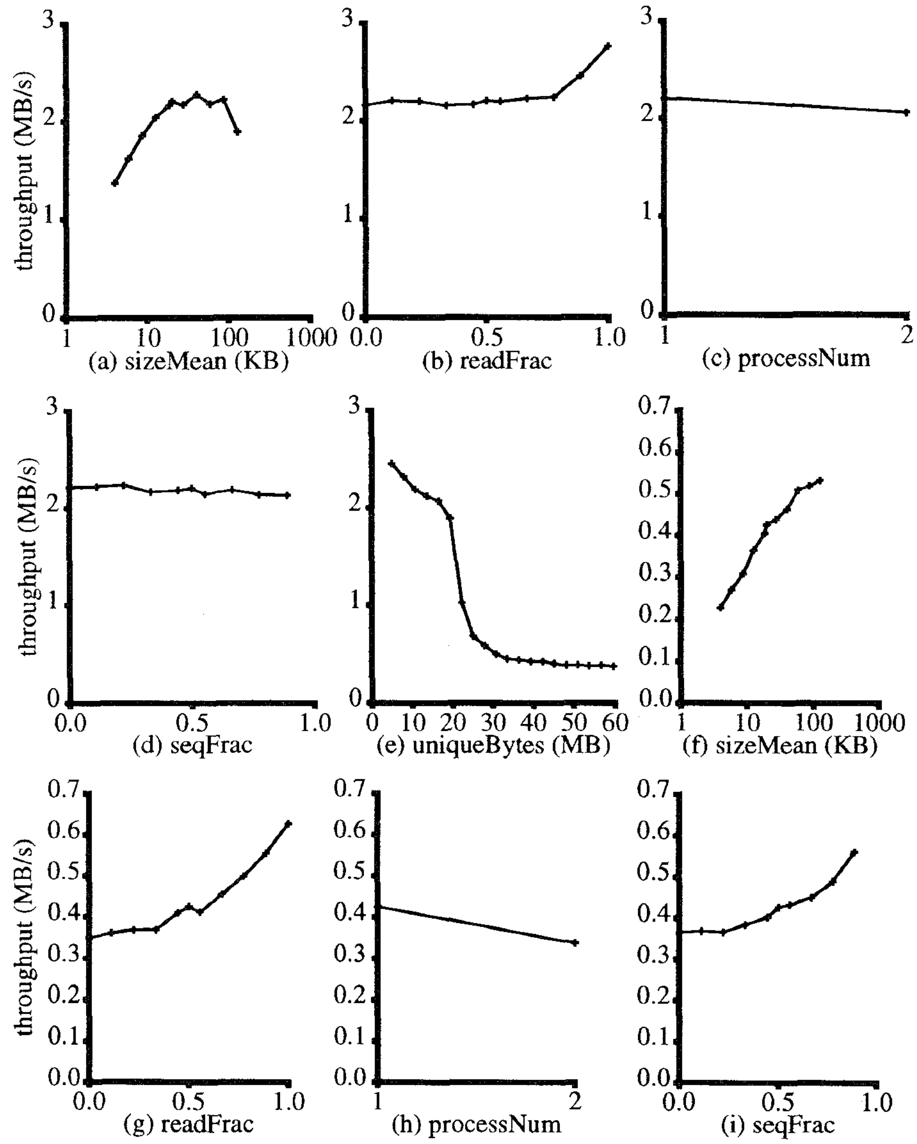


Fig. 7. Results from a better self-scaling benchmark of a SPARCstation 1+. This figure shows results from the revised self-scaling benchmark of a SPARCstation 1+. The focal point for uniqueBytes is 12 MB in graphs (a)–(d) and 42 MB in graphs (f)–(i). For all graphs, the focal points for the other parameters is sizeMean = 20 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. Increasing sizes improve performance, more so for disk accesses than file cache accesses (Figures (a) and (f)). Reads are faster than writes, even when data is in the file cache. This is because inodes must still go to disk. Sequentially increases performance only for the disk region (Figure (i)).

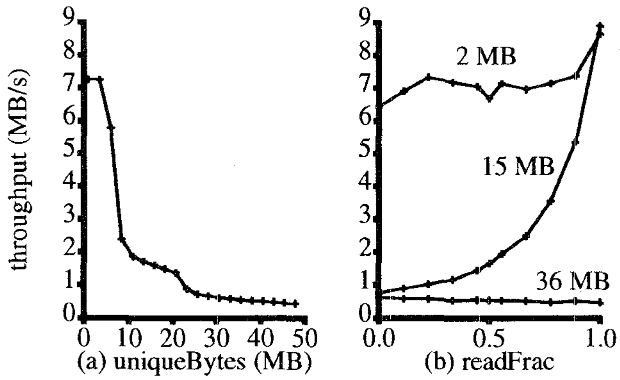


Fig. 8. Selected results from self-scaling benchmark of DECstation 5000/200. In this figure, we show selected results from the revised self-scaling benchmark of the DECstation 5000/200. Graph (a) shows three plateaus in unqiueBytes, due to the different effective file cache sizes for reads and writes. The focal points chosen for uniqueBytes are 2 MB, 15 MB, and 36 MB. The focal points for the other parameters are sizeMean = 40 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. Note in graph (b) how reads are much faster than writes at uniqueBytes 15 MB, slightly slower than writes at uniqueBytes 36 MB, and approximately the same speed at uniqueBytes 2 MB. These results helped us understand that, due to the default limit on the number of dirty file cache blocks allowed, the effective file cache size for writes was much smaller than the file cache size for reads.

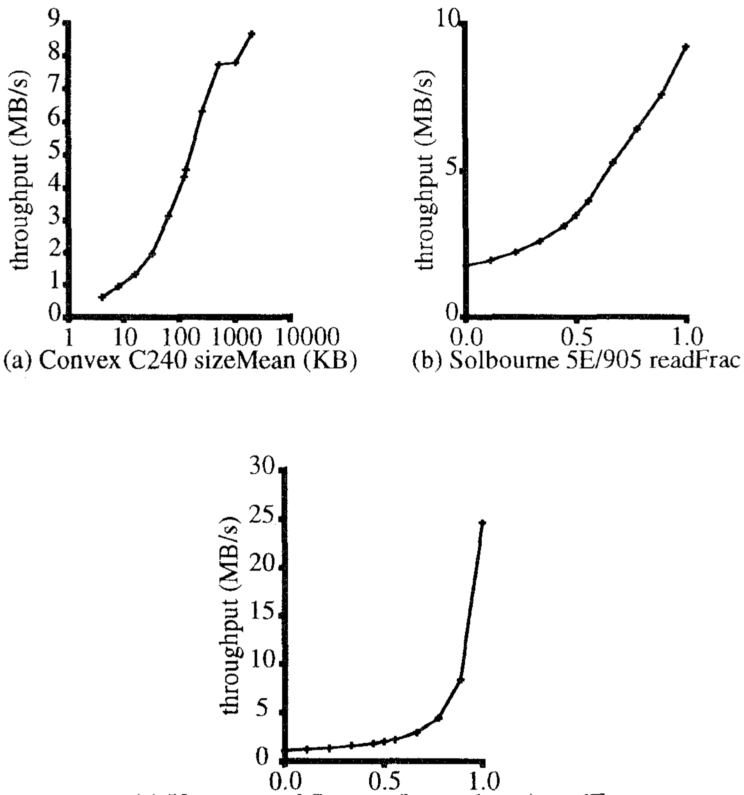
to Sprite's LFS, which improves write performance by grouping multiple small writes into fewer large writes [Rosenblum and Ousterhout 1991].

8.1.3 *HP* 730, *Convex C*240, *Solbourne* 5*E*/905. Figures 9 and 10 give selected graphs from self-scaling benchmark runs on an HP 730, Convex C240, and Solbourne 5E/905 (complete results for these machines, as well as results for two client-server file systems, can be found in the Appendix). In this section, we highlight some insights gained from these benchmark results.

Figure 10 shows that the HP 730's file cache, though extremely fast, is quite small (3 MB). The HP/UX operating system severely limits the memory available to the file cache. SunOS maps files into virtual memory, which allows the file cache to fill the entire physical memory. HP/UX, on the other hand, reserves a fixed amount of space, usually 10%, as the file cache. Since this system has 32 MB of main memory, the file cache is approximately 3 MB. The self-scaling benchmark thus uses two focal points, uniqueBytes = 2 MB and uniqueBytes = 8 MB. Note the high throughput of the HP 730 when accessing the file cache, peaking at almost 30 MB/s for large accesses. This high performance is due to the fast memory system of the HP 730 (peak memory bandwidth is 264 MB/s) and to the use of a VLSI memory controller to accelerate cache memory write backs [Horning et al. 1991].

Figures 9(a) and 10 shows results from the self-scaling benchmark of a Convex C240. The curves are similar to the SPARCstation 1+, with three main differences:

(1) Absolute performance is very high. File cache performance reaches 25 MB/s (Figure 21); disk performance reaches almost 10 MB/s (Figure 10).



(a) Convex C240 sizeMean (KB)
(b) Solbourne 5E/905 readFrac
(c) Uannounced System (beta release) readFrac

Fig. 9. Selected results from Convex, Solbourne and an unannounced system. This figure gives selected results from running the self-scaling benchmark on a Convex C240, a Solbourne 5E/905, and an unannounced system. Figure (a) shows how the performance on the Convex continues to improve with larger sizes, even to average sizes of 1 MB. Figures (b) and (c) show how writes in the these systems' file caches are much slower than reads, possibly due to a write-through file cache. For all these systems, the focal point chosen by the benchmark was readFrac 0.5, seqFrac 0.5, processNum 1. The focal points for size were 120 KB for the Convex, 45 KB for the Solbourne, and 40 KB for the unannounced system.

This high performance is due to Convex's 200 MB/s memory system and performance-focused (as opposed to cost performance) implementation.

(2) The effective file cache for the Convex is 800 MB. This is due to the 1 GB of main memory resident on the computer and an operating system that gives the file cache use of the entire main memory.

(3) Disk performance continues to improve with increasing request size until requests are 1 MB (Figure 9(a)), while most other computers reach their peak performance with sizes of a few hundred kilobytes.

Figures 9(b) and 10 shows self-scaling benchmark results for the Solbourne 5E/905. We see two differences from the other graphs.

(1) The file cache is quite large, about 300 MB (Figure 10(b)). This matches our expectations, since the main memory for this system is 384 MB.

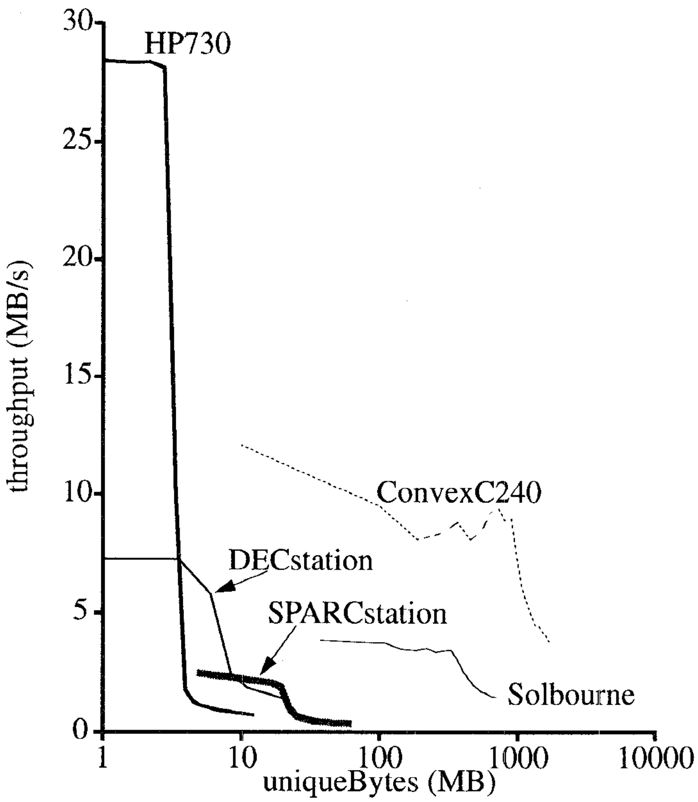


Fig. 10. Summary of performance of all systems. This figure compares performance for each of our five experimental platforms plotted against uniqueBytes. Remember that the self-scaling evaluation chooses a different workload for each machine; each system in this figure is graphed at different values of sizeMean. Note that the X axis is graphed in log scale. The HP 730 has the best file cache performance but the smallest file cache; the Convex C240 has the best disk performance.

(2) When accessing the file cache, writes are much slower than reads (Figure 9(b)). It appears that the Solbourne file cache uses a writing policy, possibly write-through, that causes writes to the file cache to perform at disk speeds. Because writes have essentially no benefit from the file cache, performance when varying uniqueBytes changes more gradually than for the other systems. This effect is even more pronounced in an unannounced system running a beta release operating system (Figure 9(c)). We notified the operating system developers of this performance problem, and they expect to fix it in forthcoming versions.

Figure 10 compares performance for each of our five experimental platforms plotted against uniqueBytes. Remember that the self-scaling evaluation chooses a different workload for each machine; each system in Figure 10 is graphed at different values of sizeMean. The HP 730 has the best file cache performance but the smallest file cache; the Convex C240 has the best disk performance.

9. PREDICTED PERFORMANCE

The self-scaling benchmark increases our understanding of a system and scales the workload to remain relevant. However, it complicates the task of comparing results from two systems. The problem is the benchmark may choose different workloads on which to measure each system. Also, though the output graphs from a self-scaling evaluation apply to a wider range of applications than today's I/O benchmarks, they stop short of applying to all workloads. In this section, we show how predicted performance solves these problems by enabling us to estimate the I/O performance accurately for arbitrary workloads based on the performance of a small set of measured workloads (that is, those measured by the self-scaling evaluation).

A straightforward approach for estimating performance for all possible workloads is to measure a comprehensive set of workloads. However, measuring all possible workloads is not feasible within reasonable time constraints. A more attractive approach is to use the graphs output by the self-scaling evaluation (such as Figure 7) to estimate performance for unmeasured workloads. This is similar in concept to work done by Saavedra-Barrera et al. [1989], who predict CPU performance by measuring the performance for a small set of Fortran operations.

We estimate performance for unmeasured workloads by assuming that the *shape* of a performance curve for one parameter is independent of the values of the other parameters. This assumption leads to an overall performance equation of $Throughput(X, Y, Z \cdots) = f_X(X) \times f_Y(Y) \times f_Z(Z) \cdots$, where $X, Y, Z, \cdots$ are the parameters. Pictorially, our approach to estimating performance for unmeasured workloads is shown for a two-parameter workload in Figure 11. In the self-scaling evaluation, we measure workloads with all but one parameter fixed at the focal point. In Figure 11, these are shown as the solid-line throughput curves $Throughput(processNum, sizeMean_f)$ and $Throughput(processNum_f, sizeMean)$, where $processNum_f$ is processNum's focal point and $sizeMean_f$ is sizeMean's focal point. Using these measured workloads, we estimate performance for unmeasured workloads $Throughput(processNum, sizeMean_1)$ by assuming a constant ratio between $Throughput(processNum, sizeMean_f)$ and $Throughput(processNum, sizeMean_1)$. This ratio is known at $processNum = processNum_f$ to be $Throughput(processNum_f, sizeMean_f) / Throughput(processNum_f, sizeMean_1)$. To measure how accurately this approximates actual performance, we measured 100 workloads, randomly selected over the entire workload space (the range of each parameter is shown in Figure 7).

Figure 12 shows the prediction accuracy of this simple product-of-single-variable-functions approach. We see that, over a wide range of performance (0.2 MB/s to 3.0 MB/s), the predicted performance values match extremely well to the measured results. Fifty percent of all workloads have a prediction error of 10% or less; 75% of all workloads had a prediction error of 15% or

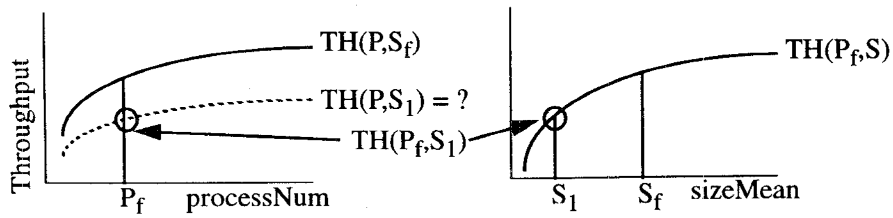


Fig. 11. Predicting performance of unmeasured workloads. In this figure, we show how to predict performance with a workload of two parameters, processNum and sizeMean. The solid lines represent workloads that have been measured; the dashed line represents workloads that are being predicted. The left graph shows throughput graphed against processNum with sizeMean fixed at $sizeMean_f$. The right graph shows throughput versus sizeMean with processNum fixed at $processNum_f$. We predict the throughput curve versus processNum with sizeMean fixed at $sizeMean_1$ by assuming that $Throughput(processNum, sizeMean_f)/Throughput(processNum, sizeMean_1)$ is constant (independent of processNum) and fixed at $Throughput(processNum_f, sizeMean_f)/Throughput(processNum_f, sizeMean_1)$.

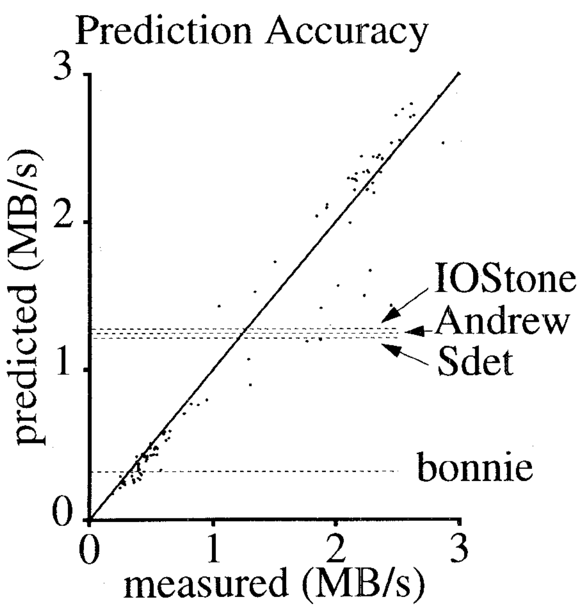


Fig. 12. Evaluation of prediction accuracy for SPARCstation 1+ with 1 disk. This figure graphs the predicted performance against the actual (measured) performance for the SPARCstation in Figure 7. Each point represents a single workload, with each parameter value randomly chosen from its entire range shown in Figure 7. The closer the points lie to the solid line, the better the prediction accuracy. Median error is 10%. Performance for each workload ranges from 0.2 MB/s to 3.0 MB/s. For comparison, we show the single performance point predicted by Andrew, IOStone, and Bonnie (sequential block write), and Sdet as horizontal dashed lines. Clearly these single-point benchmarks do not predict the performance of many workloads.

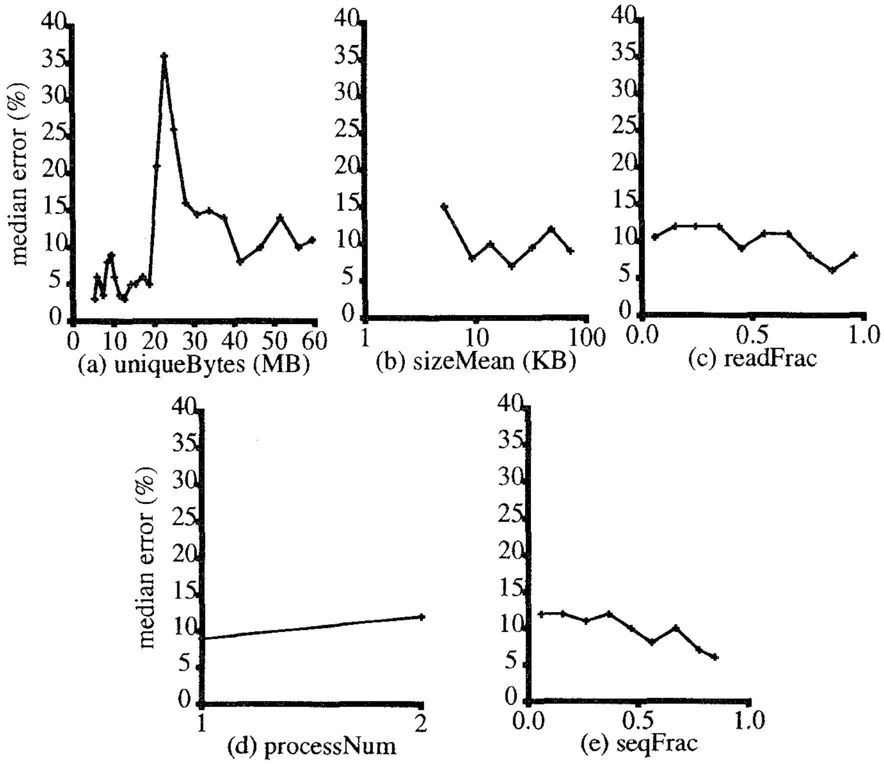


Fig. 13. What parameters cause error? This figure shows the correlation between parameter values and prediction error for the SPARCstation 1+. Error is most closely correlated to the value of uniqueBytes (the analogous graphs of error versus sizeMean, readFrac, processNum, and seqFrac are flat by comparison). Prediction is particularly poor near the border between performance regions. As expected, sharp drops in performance lead to unstable throughput and poor prediction.

less. In contrast, any single-point I/O benchmark would predict all workloads to yield the same performance. For example, Andrew's workload and IOStone's workload both yield performance of 1.25 MB/s, leading to a median prediction error of 50%. Bonnie's sequential block write yields a performance of 0.32 MB/s, for a median prediction error of 65%. These are shown by the dashed lines in Figure 12.

Where do the points of high error occur? Is there a correlation between certain parameters and regions of high error? Figure 13 shows how median error varies with each parameter. Error is most closely correlated to the value of uniqueBytes. Prediction is particularly poor near the border between performance regions. As expected, sharp drops in performance lead to unstable throughput and poor prediction. This confirms our use of two distinct uniqueBytes regions for prediction versus a single focal point. Other than uniqueBytes, prediction accuracy is fairly independent of the parameter

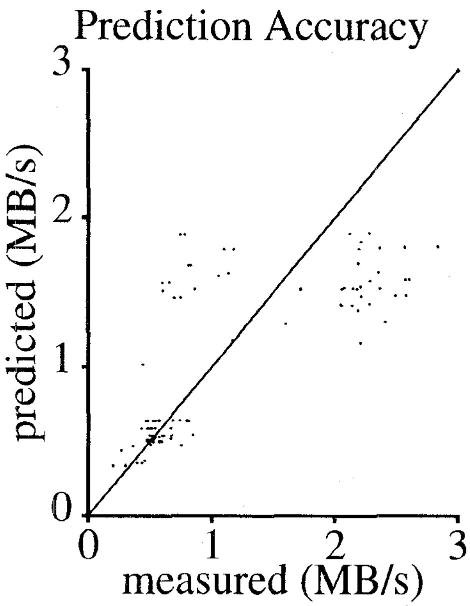


Fig. 14. Prediction accuracy using interpolation on an orthogonal sample. This figure shows that interpolation using an orthogonal sample gives poorer prediction than does a prediction model based on products of single-parameter functions. Even when using twice as many workloads to do prediction, median error is twice as high (22%).

values. Taking out the points of high error increases the prediction accuracy significantly (Figure 24 in the Appendix).

A skeptic would claim that the enhanced accuracy over single-point benchmarks is due entirely to using more points to make predictions, rather than the validity of the product-of-independent-functions model. To rebut such a claim, we show next the accuracy of predicting performance using more workload points but *not* using the *product-of-single-parameter-functions* approach. This prediction is done by using multidimensional interpolation on an orthogonal sampling of workloads. Orthogonal sampling means selecting a fixed number of evenly spaced values from each parameter's range and measuring all combinations of workloads with those parameter values. For example, assume that a workload consists of two parameters, readFrac, ranging from 0–1, and uniqueBytes, ranging from 1 MB to 9 MB. Taking three values from each parameter would lead to an orthogonal sampling of nine total workloads of (readFrac, uniqueBytes) pairs: (0, 1 MB), (0, 5 MB), (0, 9 MB), (0.5, 1 MB), (0.5, 5 MB), (0.5, 9 MB), (1, 1 MB), (1, 5 MB), and (1, 9 MB).

For the SPARCstation 1+, our original approach of using a *product of single-parameter functions* requires the 84 points, each taking approximately 10 minutes to measure, that are returned by the self-scaling benchmark.[6] As can readily be seen, orthogonal sampling inherently requires many more

[6] Six of the nine graphs have 10 points each; one (uniqueBytes) uses 20 points; two (processNum) use only two points each, due to the limited range of processNum.

System	Median Error	Enhanced Error	Repeatability Error
SPARCstation 1+	10%	7%	2%
DECstation 5000/200	12%	10%	3%
HP 730	13%	8%	3%
Convex C240	14%	8%	5%

Fig. 15. Summary of median prediction errors. This table summarizes the prediction errors on all systems. "Enhanced Error" on all machines but the Convex refers to the prediction error on the sample of workloads not in the thrashing region between the file cache and disk locality regions. For the Convex, prediciton error was most closely correlated with sizeMean, so enhanced error refers to points with sizes smaller than 300 KB. The last column in the table lists the inherent measurement error, which was measured by running the same set of random workloads twice and using one run to "predict" performance of the other run.

points. Sampling three values per dimension requires 162 points.[7] Figure 14 shows that even with twice as many points, interpolation using an orthogonal sampling gives much worse prediction accuracy than does the *product of single-parameter functions*: the median error using orthogonal sampling is 22% versus 10% using the *product of single-parameter functions*, and it takes twice as long to collect the necessary information.

Figure 15 summarizes prediction accuracy for the other systems (Figure 24 shows the prediction scatter plots with enhanced accuracy). Median error is low for all systems: 12% for the DECstation 5000/200, 13% for the HP 730, and 14% for the Convex C240. Not including the points of highest error, which in most systems is the border region between the file cache and disk, median error drops to 7–10%.

Performance evaluators sometimes want to measure more than absolute performance. Their end goal is often to compare two systems head to head, generally as the ratio of performances. Once the self-scaling benchmark is run on two systems, predicted performance can be used to estimate ratios for arbitrary workloads without further measurement. Figure 17 shows the accuracy of estimating the ratio of performance between our test systems over a variety of workstation, scientific, and database I/O workloads, described in Figure 16. In particular, note Figure 17(a), which compares the HP 730 and the Sprite DECstation 5000/200. Depending on the workload, the HP can be three times faster to five times slower than the DECstation. Predicted performance gives accurate estimates over this wide range of ratios.

10. CONCLUSIONS

We have proposed a new approach to I/O benchmarking—self-scaling evaluation and predicted performance. Self-scaling evaluation scales automatically to all current and future machines by scaling the workload to the system under test. It gives insight also on a machine's performance characteristic by revealing its performance dependencies for each of five workload parameters.

[7] Normally this would be 3^5, or 243 points. However, due to the limited range of processNum, only two points are required for this dimension, which brings the total required to 162.

Title	Example	unique Bytes (MB)	seq Frac	read Frac	sizeMean (KB)	process Num
workstation	compile grep	1	0.8	0.8	4	1
large_utility	sort	10	0.9	0.6	8	1
scientific_write	satellite data	250	0.5	0.2	100	1
scientific_read	image process	250	0.5	0.8	100	1
database	TPC	500	0.1	0.2	4	2

Fig. 16. Workloads to be run on all systems. This table describes the workloads we use when comparing two systems.

Predicted performance restores the ability to compare two machines on the same workload lost in the self-scaling evaluation. Further, it extends this ability to workloads that have not been measured by estimating performance based on the graphs from the self-scaling evaluation. We have shown that this prediction is far more accurate over a wide range of workloads than any single-point benchmark.

We believe self-scaling evaluation and predicted performance could fundamentally affect how manufacturers and users view I/O evaluation. First, it condenses the performance over a wide range of workloads into a few graphs. If manufacturers were to publish such graphs over a range of I/O options, users could use predicted performance to estimate, *without further measurements*, the I/O performance of their specific workloads.

Second, by taking advantage of the self-scaling evaluation's ease of use, manufacturers could easily evaluate many I/O configurations. Instead of merely reporting performance for each I/O configuration on a few workloads, the evaluation would report both the performance for many workloads and the I/O workloads that perform well under this configuration. Hence, manufacturers could better identify each product's target application area. As the price of each configuration is easily calculated, the price/performance of systems that match the users needs are also easily calculated. This can help buyers make choices such as many small disks versus a few large disks, more memory, and a larger file cache versus faster disks, and so on.

Third, system developers could benefit by using the self-scaling evaluation to understand the effects of any hardware and software changes. Unlike traditional benchmarks, these effects would be shown in both the performance and the workload selection from the self-scaling evaluation.

Future work for this research includes finding ways to shorten the running time, feeding back information from predicted performance into which focal points the self-scaling benchmark chooses, and, of course, running the benchmark on more systems, particularly network file servers and mainframes.

We look forward to having others try this evaluation tool on a variety of systems. A copy of the benchmark is available for anonymous ftp on ftp.cs.berkeley.edu under ucb/benchmarks/pmchen/benchmark.tar.Z.

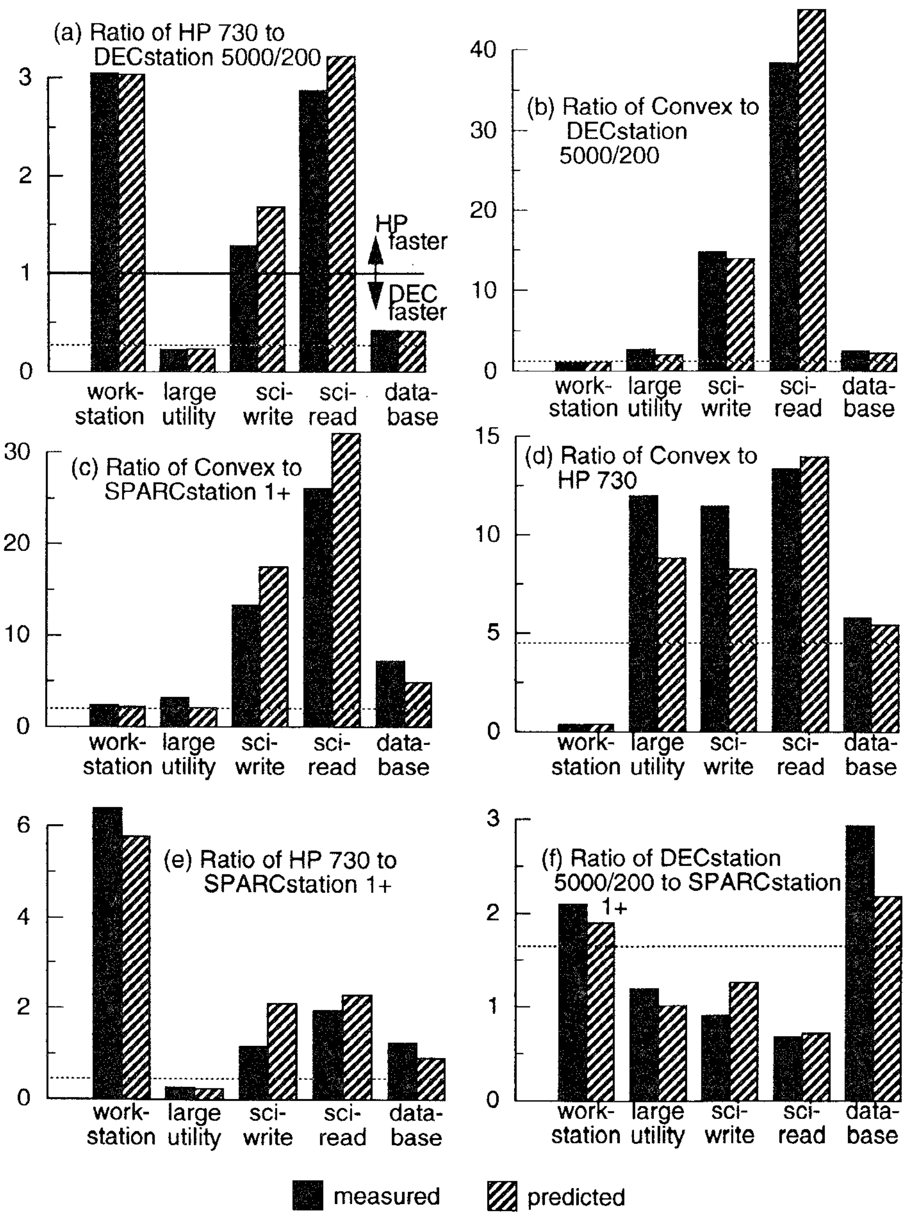


Fig. 17. Measured versus predicted ratio. This figures shows how accurately systems can be compared using performance prediction. For comparison, the performance ratio given by Andrew is shown as a dashed line. Predicting performance using the output of the self-scaling benchmark captures this variability in the ratio of two systems' performance in a way that no single-point benchmark can.

APPENDIX

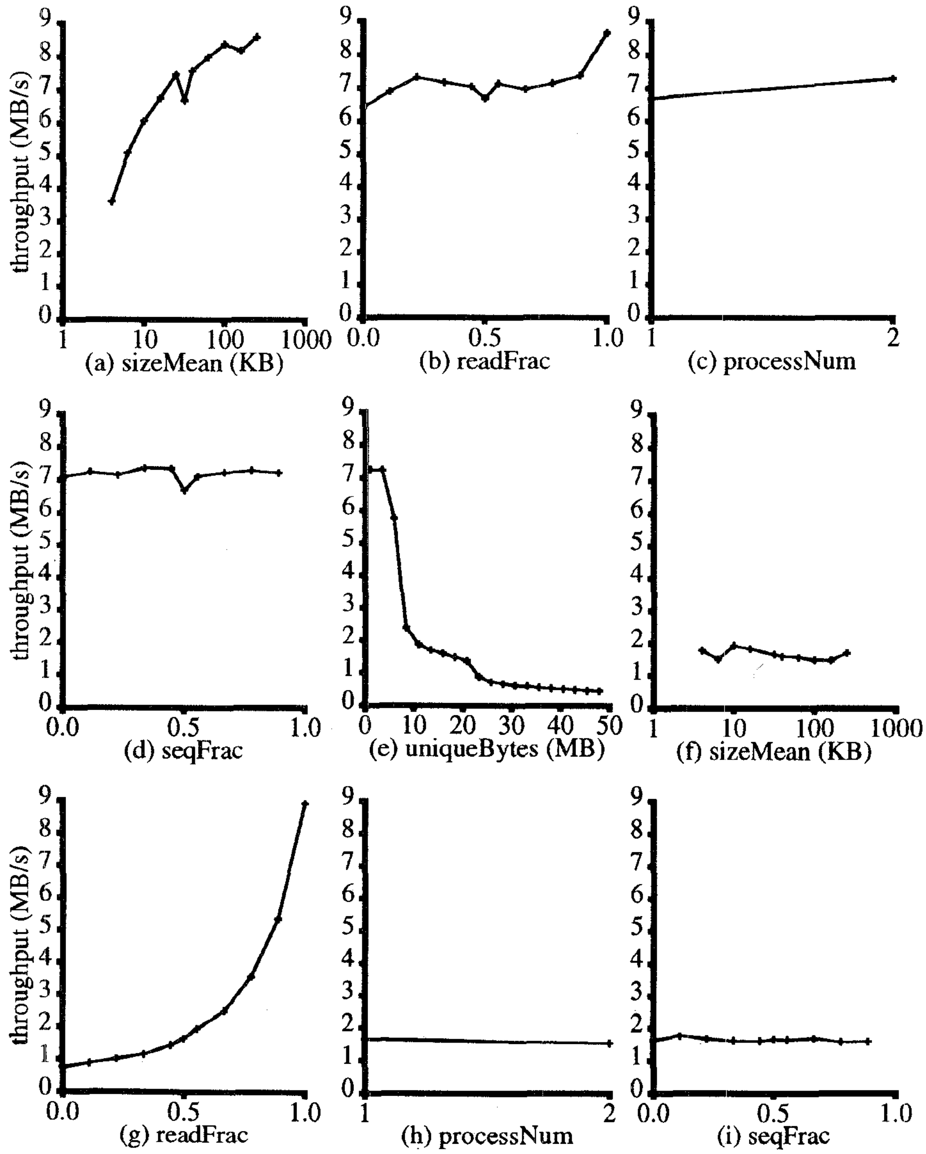


Fig. 18. Self-scaling benchmark for DECstation 5000/200—Part 1. This figure shows results from the revised, self-scaling benchmark of the DECstation 5000/200. Graph (e) shows three plateaus in uniqueBytes, due to the different effective file cache sizes for reads and writes. Graphs for the third plateau are shown in Figure 19. The focal point for uniqueBytes is 2 MB in graphs (a)–(d) and 15 MB in graphs (f)–(i). For all graphs, the focal points for the other parameters is sizeMean = 40 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. Note how reads can be faster than writes (g), slower than writes (Figure 19b), or the same speed (b).

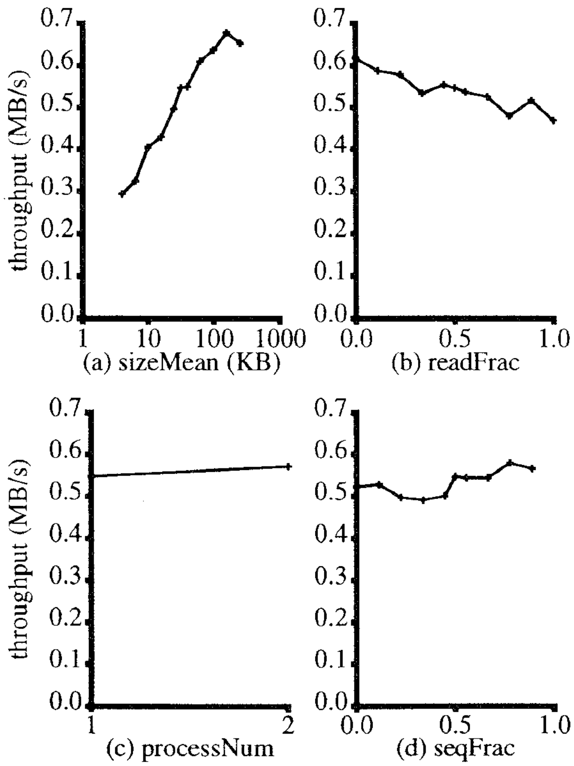


Fig. 19. Self-scaling benchmark for DECstation 5000/200—Part 2. This figure continues the results from the revised, self-scaling benchmark of the DECstation 5000/200. The focal point for these graphs is uniqueBytes = 36 MB, sizeMean = 40 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5.

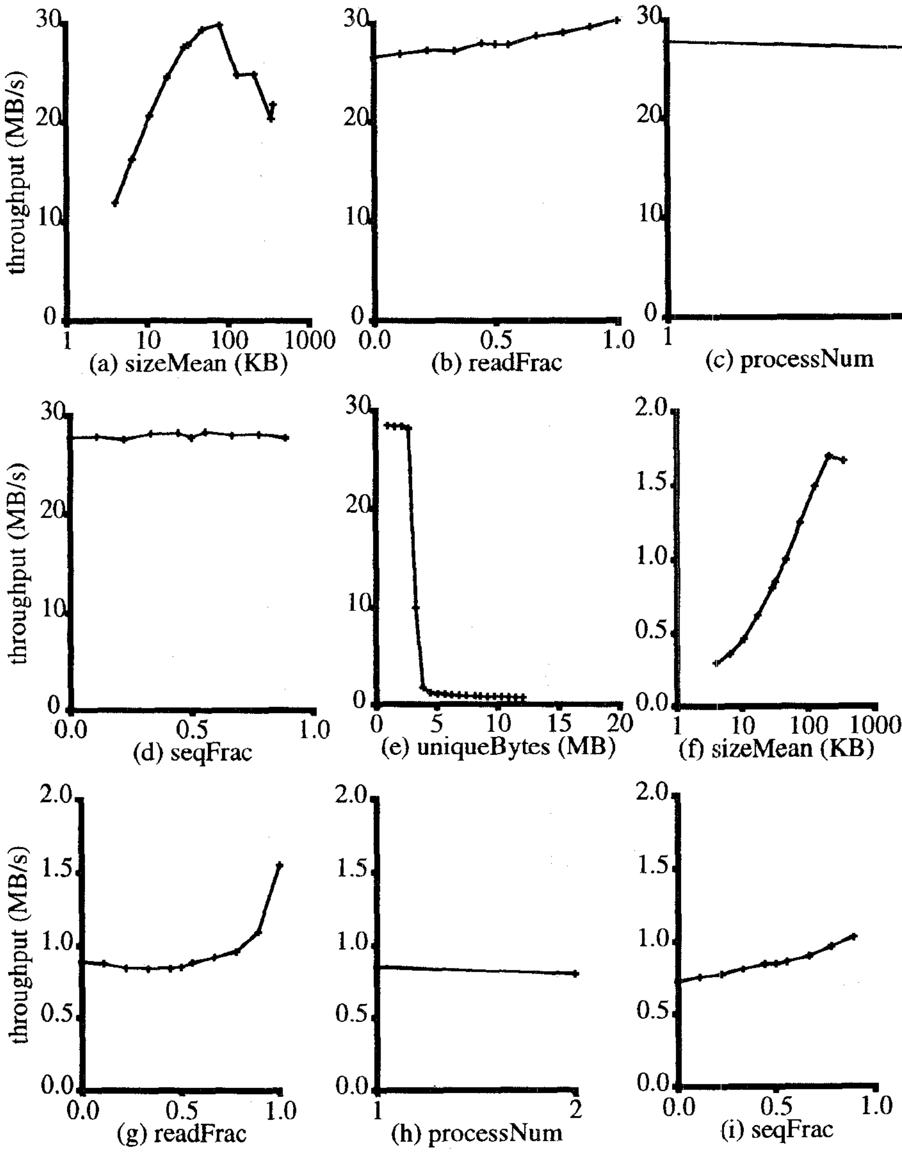


Fig. 20. Self-scaling benchmark of HP 730. In this figure, we show results from the revised self-scaling benchmark of the HP 730. The focal point for uniqueBytes is 1.8 MB in graphs (a)–(d) and 8 MB in graphs (f)–(i). For all graphs, the focal points for the other parameters are sizeMean = 32 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. From graph (e), we see that the file cache is much smaller than the main memory size would lead us to believe (3 MB instead of 32 MB). Also note the extremely high performance, up to 30 MB/s, when uniqueBytes is small enough to fit in the file cache.

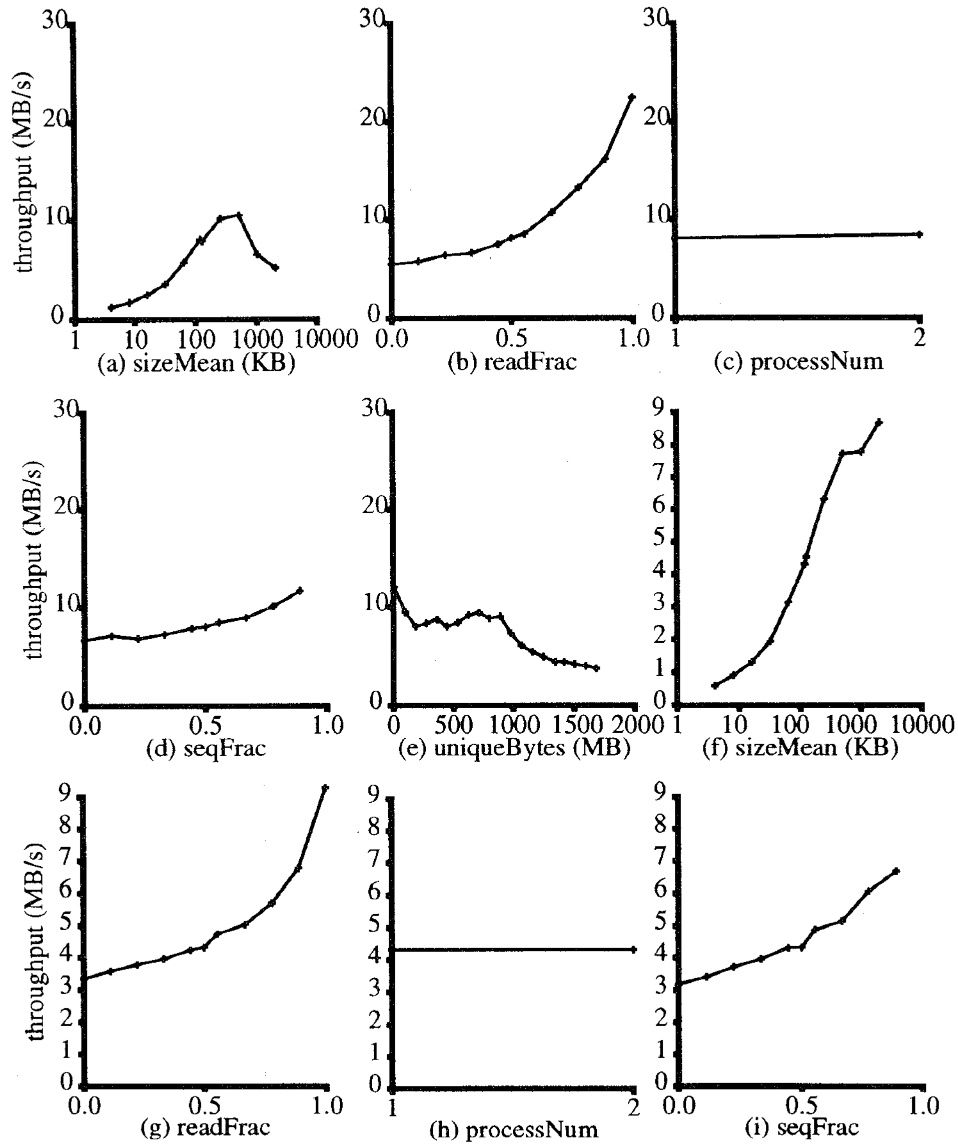


Fig. 21. Self-scaling benchmark of Convex C240. In this figure, we show results from the revised self-scaling benchmark of the Convex C240. The focal point for uniqueBytes is 450 MB in graphs (a)–(d) and 1376 MB in graphs (f)–(i). For all graphs, the focal points for the other parameters is sizeMean = 120 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. Note how large the file cache is (800 MB), reflecting how much main memory the system has (1 GB). Also note how large sizeMean grows before reaching its maximum performance (graphs (a) and (f)).

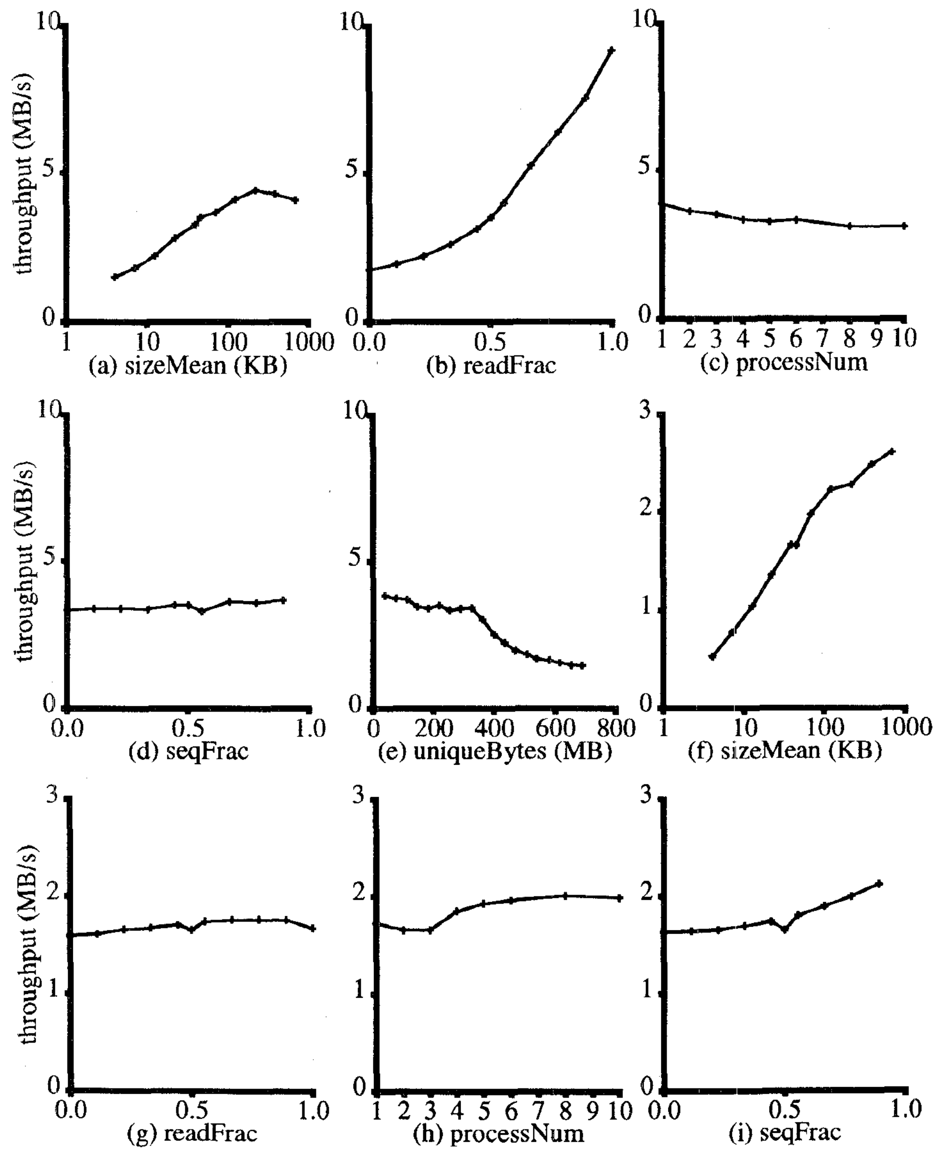


Fig. 22. Self-scaling benchmark of Solbourne. In this figure, we show results from the revised self-scaling benchmark of the Solbourne. The focal point for uniqueBytes is 180 MB in graphs (a)–(d) and 540 MB in graphs (f)–(i). For all graphs, the focal points for the other parameters is sizeMean = 45 KB, readFrac = 0.5, processNum = 1, seqFrac = 0.5. Note how much slower writes are than reads when using the file cache (graph (b)). Apparently, the cache write-back policy on the Solbourne causes writes to perform at disk speeds even when accesses can be satisfied by the file cache.

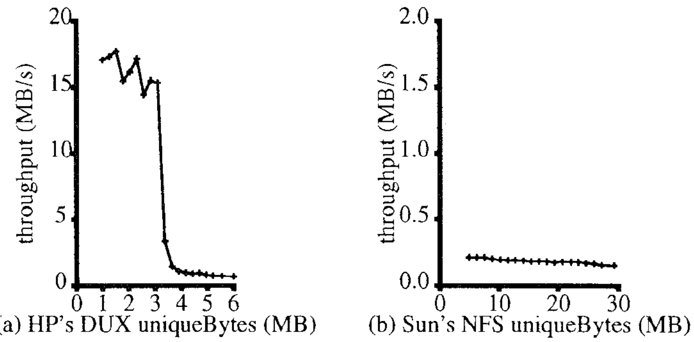


Fig. 23. Self-scaling results for two client-server configurations. This figure shows the results from a self-scaling benchmark of two client-server configurations. The configuration used in graph (a) is an HP 720 client workstation accessing an HP 730 (same as Figure 2) file server running DUX (HP's Distributed Unix protocol). The configuration used in graph (b) is a SPARCstation 1+ client workstation accessing a separate SPARCstation 1+ file server running Sun's NFS protocol. Both configurations use an ethernet network to connect the client and server. The main reason the HP configuration gets much higher performance is that the DUX protocol allows clients to cache both reads and writes, allowing I/Os to proceed at memory speeds rather than disk or network speeds.

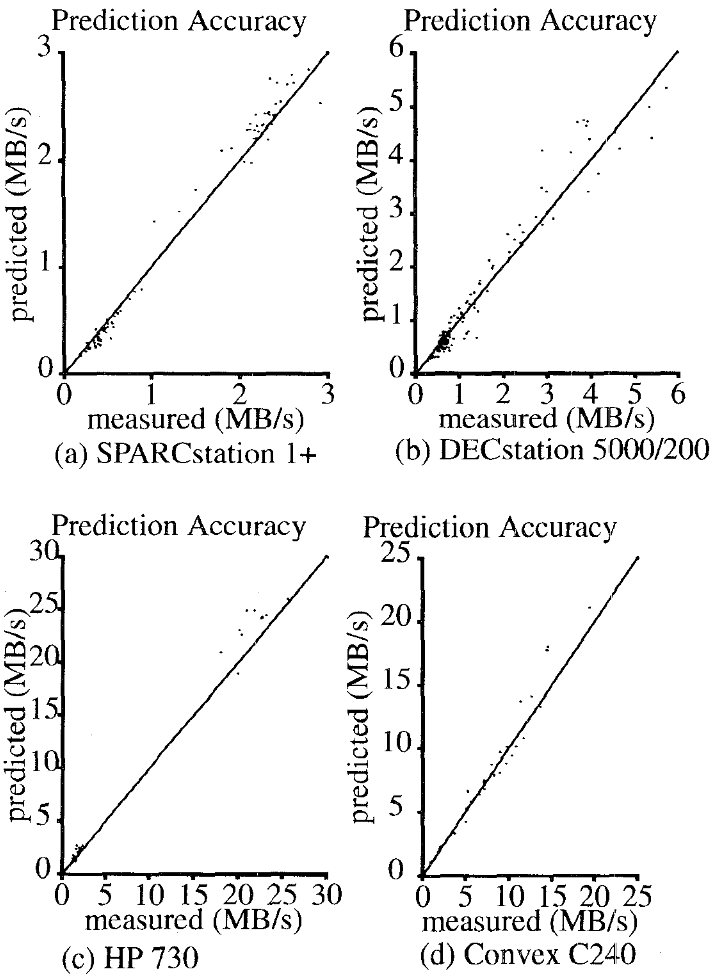


Fig. 24. Enhanced prediction accuracy. This figure shows the enhanced prediction accuracy when the points of high error are taken out. For the SPARCstation 1+, workloads with uniqueBytes between 20 and 27 MB are taken out, and median error drops from 10% to 7%. For the DECstation 5000/200, workloads with uniqueBytes between 5 and 10 MB are taken out, and median error drops from 12% to 10%. For the HP 730, workloads with uniqueBytes between 3 and 4 MB are taken out, and median error drops from 13% to 8%. For the Convex C240, workloads with very large mean sizes (greater than 300 KB) are taken out, and median error drops from 14% to 8%.

ACKNOWLEDGMENTS

We thank Garth Gibson, John Hartman, Randy Katz, Edward Lee, Mendel Rosenblum, Rafael Saavedra-Barrera, John Wilkes, and the anonymous reviewers for their insights and suggestions.

REFERENCES

ANON ET AL. 1985. A measure of transaction processing power. *Datamation 31*, 7 (Apr.), 112–118.

BECHTOLSHEIM, A. V., AND FRANK, E. H. 1990. Sun's SPARCstation 1: A Workstation for the 1990s. In *Procedures of the IEEE Computer Society International Conference (COMPCON)*. ACM, New York, 184–188.

BERRY, M., CHEN, D., KOSS, P., KUCK, D., LO, S., PANG, Y., POINTER, L., ROLOFF, R., SAMEH, A., CLEMENTI, E., CHIN, S., SCHNEIDER, D., FOX, G., MESSINA, P., WALKER, D., HSIUNG, C., SCHWARZMEIER, J., LUE, K., ORSZAG, S., SEIDL, F., JOHNSON, O., GOODRUM, R., AND MARTIN, J. 1989. The perfect club benchmarks: Effective performance evaluation of supercomputers. *Int. J. Supercomput. Appl.* (Fall).

BRAY, T. 1990. Bonnie source code. Netnews posting USENET.

CHEN, P. M. 1992. Input-output performance evaluation: Self-scaling benchmarks, predicted performance. Ph.D. dissertation, UCB/Computer Science Dept. 92/714, Univ. of California.

CHEN, P. M., AND PATTERSON, D. A. 1990. Maximizing performance in a striped disk array. In *Proceedings of the 1990 International Symposium on Computer Architecture* (Seattle, May). IEEE/ACM, New York, 322–331.

DEC. 1990. *DECstation 5000 Model 200 Technical Overview*. Digital Equipment Corp., Palo Alto, Calif.

FERRARI, D. 1984. On the foundations of artificial workload design. In *Proceedings of the 1984 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM, New York, 8–14.

GAEDE, S. 1982. A scaling technique for comparing interactive system capacities. In the *13th International Conference on Management and Performance Evaluation of Computer Systems*. Computer Measurement Group, 62–67.

GAEDE, S. 1981. Tools for research in computer workload characterization. *Experimental Computer Performance and Evaluation*, D. Ferrari, and M. Spadoni, Eds.

HEWLETT-PACKARD. 1992. *HP Apollo Series 700 Model 730 PA-RISC Workstation*, Hewlett-Packard, Palo Alto, Calif.

HORNING, R., JOHNSON, L., THAYER, L., LI, D., MEIER, V., DOWDELL, C., AND ROBERTS, D. 1991. System design for a low cost PA-RISC desktop workstation. In *Procedures of the IEEE Computer Society International Conference (COMPCON)*. IEEE, New York, 208–213.

HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1 (Feb.), 51–81.

OUSTERHOUT, J. K., AND DOUGLIS, F. 1989. Beating the I/O bottleneck: A case for log-structured file systems. *SIGOPS 23*, 1 (Jan.), 11–28.

OUSTERHOUT, J. K., CHERENSON, A., DOUGLIS, F., AND NELSON, M. 1988. The Sprite network operating system. *IEEE Comput. 21*, 2 (Feb.), 23–36.

PARK, A., AND BECKER, J. C. 1990. IOStone: A synthetic file system benchmark. *Comput. Arch. News 18*, 2 (June), 45–52.

PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. 1988. A case for redundant arrays of inexpensive disks (RAID). In the *ACM International Conference on Management of Data (SIGMOD)*. ACM, New York, 109–116.

ROSENBLUM, M., AND OUSTERHOUT, J. K. 1991. The design and implementation of a log-structured file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*. ACM, New York.

SPEC. 1991a. *SPEC SDM Release 1.0 Manual*. System Performance Evaluation Cooperative, Fairfax, Va.

SPEC. 1991b. *SPEC SDM Release 1.0 Technical Fact Sheet*. Franson and Hagerty Associates, Fairfax, Va.

SAAVEDRA-BARRERA, R. H., SMITH, A. J., AND MIYA, E. 1989. Machine characterization based on an abstract high-level language machine. *IEEE Trans. Comput. 38*, 12 (Dec.), 1659–1679.

SCOTT, V. 1990. Is standardization of benchmarks feasible? In *Proceedings of the BUSCON Conference3* (Long Beach, Calif.). Conference Management Corp., 139–147.

TPPC. 1990. *TPC Benchmark B Standard Specification.* Transaction Processing Performance Council, Freemont, Calif.

TPPC. 1989. *TPC Benchmark A Standard Specification.* Transaction Processing Performance Council, Freemont, Calif.

Received May 1993; revised October 1993; accepted October 1993