Chapter 2

Previous Benchmarks

2.1. I/O Benchmarks

Chapter 1 highlighted some trends and developments in disks, disk arrays, file caches, solid state disks, tapes, and file systems which create new challenges for the evaluation of storage systems. As a result, benchmarks used in the evaluation process must evolve to comprehensively stress these new I/O systems. For example, because disk arrays are able to service many I/Os at the same time, benchmarks need to issue many simultaneous I/Os if they hope to stress a disk array. Caches create distinct performance regions based on the capacity used by a program, so benchmarks likewise should measure these different performance regions.

In this chapter, I list standards that I will use to critique I/O benchmarks. I then review, run, and evaluate I/O benchmarks in use today.

2.2. The Ideal I/O Benchmark

In this thesis, I define I/O benchmarks as measuring the data I/O performance seen by a program issuing reads and writes. Specifically, I am *not* using I/O benchmarks to measure the performance of file system commands, such as deleting files, making directories, or opening and closing files. While these are perfectly valid and important metrics, they are more a measure of the operating system and processor speed than they are of the storage components.

It is unfortunate that most people use trivial benchmarks when purchasing and evaluating I/O systems. These include, for example, the time it takes to write 1 MB to disk, the average disk access time, or the raw disk transfer rate. These metrics are similar to the CPU clock rate in processor performance evaluation in that they are better than nothing but do not translate easily into performance that an end user can see. To correct this blind acceptance of trivial benchmarks, I list six desirable characteristics of I/O benchmarks in this section.

First, a benchmark should help system designers and users understand why a system performs as it does. This is because computer architects and operating system programmers need to have benchmarks to evaluate design changes and isolate reasons for poor performance. In addition, users should be also able to use benchmarks to understand the optimal ways of using their machines. For instance, if a user wanted to have his application fit within the file cache, the ideal I/O benchmark should be able to provide information on the size of a machine's file cache. This criterion may require reporting results for several different workloads, which would enable 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, the performance of an I/O benchmark should be limited by the I/O devices. The most intuitive test of being I/O-limited is the following: if the speedup resulting from taking out all I/Os from an application is greater than the speedup resulting from taking out all CPU operations (leaving only I/O), then the application is I/O-limited. Unfortunately, this test is quite hard to perform in practice—almost any non-trivial application will not function when its I/O is eliminated. Instead, I test for how I/O-limited an application is by measuring the fraction of time it spends in doing reads and writes.

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 quickly become obsolete as machines evolve. For instance, *IOStone* tries to exercise a system's entire 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, of course, this is no longer true. Another example of an I/O benchmark's need to scale is provided by disk arrays. As mentioned above, disk arrays allow multiple I/Os to be in progress simultaneously. But, because most current I/O benchmarks do not scale the number of processes issuing I/O, they 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 broad range of audiences.

Fourth, a good I/O benchmark should allow fair comparisons across machines and should include two aspects. First, a fair comparison across machines should be able to be made for I/O workloads identical to the benchmark. However, since users rarely have the same workload as a standard benchmark, the results from a benchmark should predict performance for workloads that differ from itself.

Fifth, the ideal I/O benchmark should be relevant to a wide range of applications. It is certainly easier to target a benchmark to a specific audience, and benchmarks that do a good job representing their target applications are invaluable for those applications. But, benchmarks usable by many audiences would be better still.

Finally, in order for results to be meaningful, benchmarks should be tightly specified. Results should be reproducible by general users; optimizations that are allowed and disallowed should be explicitly stated; the machine environment on which the benchmarking takes place should be well-defined and reported (CPU type and speed, operating system, compiler, network, disk, other load on the system); the starting state of the system (file cache state, data layout on disk) should be well-defined and consistent, and so on.

In summary, the six characteristics of the ideal I/O benchmark are as follows: it should help the evaluator understand system performance; its performance 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.

2.3. System Platforms

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 disk (Wren) RAID 0	HP 1350SX
I/O Bus	SCSI-I	SCSI-I	Fast SCSI-II
Memory Size	28 MB	32 MB	32 MB
Mem. Bus Peak Speed	80 MB/s	100 MB/s	264 MB/s
Mem. Bus Sustained Speed	25-30 MB/s	???	125 MB/s
Operating System	SunOS 4.1	Sprite LFS	HP/UX 8.07

Table 2.1: System platforms. This table shows the three systems on which benchmarks were run. The DECstation [DECstation90] uses a three disk RAID disk array [Patterson88] with a 16 KB striping unit [Chen90] 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 [HP730].

In this chapter, I run standard I/O benchmarks on three systems. Additional systems are tested in Chapters 4 and 5 (see Table 4.1 on page 64). All systems in this chapter are high-performance workstations with differing I/O systems; Table 2.1 summarizes their characteristics. Note that these computers were introduced in different years—this study is not meant to be a competitive market analysis of the competing products.

In order to better understand these benchmarks, I modified their software slightly. For example, I compiled in special I/O routines that traced I/O activity. To accomplish this, I used publicly available code for as many programs as possible. In general, I used GNU (Gnu's Not Unix) code developed by the Free Software Foundation. To make results directly comparable between machines for benchmarks that used the compiler, I took the same step as Ousterhout [Ousterhout90] in having the GNU C compiler generate code for an experimental CPU called SPUR [Hill86].

2.4. Overview of Current I/O Benchmarks

In this section, I describe, critique, and run five common benchmarks used in I/O system evaluation: Andrew, TPC-B, Sdet, Bonnie, and IOStone. Table 2.2 contains information about how to obtain these benchmarks. I categorize them into two classes: application benchmarks and synthetic benchmarks.

Benchmark	Contact	E-mail address
SPEC SDM	National Computer Graphics Corp.	spec-ncga@cup.portal.com
TPC	Shanley Public Relations	shanley@cup.portal.com
Bonnie	Tim Bray	tbray@watsol.Waterloo.EDU
IOStone	Jeffrey Becker	becker@iris.ucdavis.EDU
LADDIS	Bruce Keith	spec-preladdis-beta-test@riscee.pko.dec.com

Table 2.2: List of contacts for various benchmarks.

2.4.1. Application Benchmarks

Application benchmarks use standard programs, such as compilers, utilities, editors, and databases, in various combinations to produce a workload. Each benchmark targets a single application area, such as transaction processing or system development. Although these benchmarks usually do a good job of accurately representing their target application area, as will be seen, they are often not I/O limited.

2.4.1.1. Andrew

Andrew was designed at Carnegie Mellon University to be a file system benchmark for comparing the Andrew File System against other file systems [Howard88]. It was originally meant to act only as a convenient yardstick for measuring file systems and not necessarily as a representative workload for benchmarking. Despite this intent, however, it has become a widely used, de facto benchmarking standard [Ousterhout90].

Andrew is meant to represent the workload generated by a typical set of software system developers. It copies a file directory hierarchy, examines and reads the new copy, then compiles the copy. The file directory contains 70 files totaling 0.2 MB. Carnegie Mellon's experience in 1987 suggests the load generated roughly equals that generated by five users.

System	Copy Phase		Compile Phase		Total	
	Time	% I/O	Time	% I/O	Time	% I/O
SPARCstation 1+	82 sec.	4%	137 sec.	7%	219 sec.	6%
DECstation 5000	20 sec.	10%	67 sec.	3%	87 sec.	4%
HP 730	17 sec.	27%	28 sec.	6%	45 sec.	13%

Table 2.3: Results from the Andrew benchmark. This table shows the results obtained from running the Andrew Benchmark. Andrew is divided into two sections: the *copy phase*, consisting of the copy, examination, and reading stages; and the *compile phase*. In each column, the percentage of time spent performing reads and writes is listed. Each of these numbers represents the average of three runs; each run starts with an empty file cache. Note the small percentage of time spent in I/O.

In Table 2.3, I list results derived from Andrew on the three system platforms. As in [Ousterhout90], I divide Andrew into two sections: the *copy phase*, consisting of the copy, examination, and reading stages; and the *compile phase*. Note that on all machines Andrew spends only 6%-13% actually doing data reads and writes. Also note that the HP 730, which has the fastest CPU, spends a higher fraction of time in I/O than the others, a result that supports my contention that systems with faster and faster CPUs will become more and more I/O limited.

2.4.1.2. TPC-B

TPC-B measures transaction processing performance for a simple, database update [TPCB90]. The first version of this benchmark, TP1, first appeared in 1985 [Anon85] and quickly became the de facto standard in benchmarking for transaction processing systems.

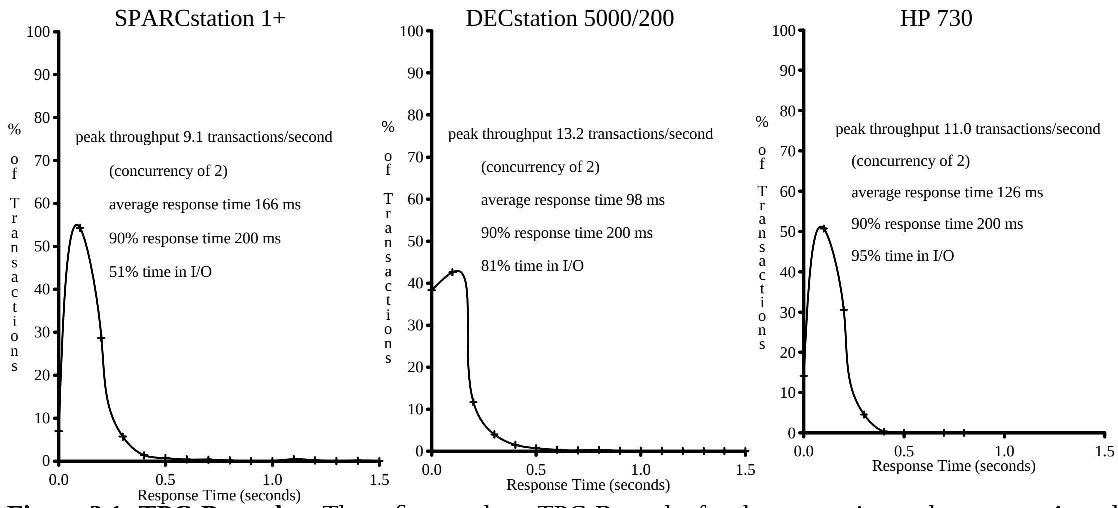

Figure 2.1: TPC-B results. These figures show TPC-B results for three experimental systems. As a database program, Seltzer's simple transaction processing library LIBTP [Seltzer92] were used. Due to a deadlocking bug in the software package, however, concurrencies higher than two were not run. This is reflected by response times much faster than those required by TPC-B.

TPC-A[1] [TPCA89] and TPC-B [TPCB90] are more tightly specified versions of TP1 and have replaced it as the standard transaction processing benchmark. As a transaction processing benchmark, TPC-B not only measures the machine supporting the database but also the database software.

TPC-B repeatedly performs *Debit-Credit* transactions, each of which simulates a typical change to an account on a bank's database, which consists of customer accounts, bank branches, and tellers. Using a random customer request, a Debit-Credit transaction reads and updates the necessary account, branch, and teller balances. Requests are generated by a number of simulated customers, each requesting transactions as quickly as possible.

TPC-B's main metric is maximum throughput measured in transactions-per-second, qualified by a response time threshold demanding that 90% of all transactions be completed within two seconds. TPC-B also reports the price of the system and the required storage. The number of accounts, branches, and tellers specified by TPC-B is proportional to throughput—for each additional transaction-per-second of performance reported, the test system's database must add 10 MBs of account information. Using a database of this size, TPC-B reports a graph of throughput versus the average number of outstanding requests as well as a histogram of response times for the maximum throughput. In Figure 2.1, I show TPC-B's response time characteristics on the three systems, using Seltzer's simple, transaction supporting package LIBTP [Seltzer92].

---

[1]The main difference between TPC-A and TPC-B is the presence of real terminals. TPC-A demands that the test be done with actual terminals providing input at an average rate of one request every 10 seconds. TPC-B generates requests with internal drivers running as fast as possible. This thesis discusses only TPC-B.

2.4.1.3. Sdet

The System Performance Evaluation Cooperative (SPEC) was founded in 1988 to establish independent standard benchmarks [Scott90]. Their first set of benchmarks, SPEC Release 1, primarily measures CPU performance. Their second set of benchmarks, System Development Multi-tasking (SDM) Suite, measures overall system performance for software development and research environments. SDM consists of two benchmarks, Sdet [Gaede81, Gaede82] and Kenbus1 [McDonell87]. Sdet and Kenbus1 are quite similar in benchmarking methodology; their main difference is the specific mix of user commands. Because Sdet does much more I/O than Kenbus1, I discuss only Sdet.

Sdet's workload consists of a number of *scripts* running concurrently. Each script contains a list of user commands in random order. These commands are taken from a typical software-development environment and include editing, text formatting, compiling, file creating

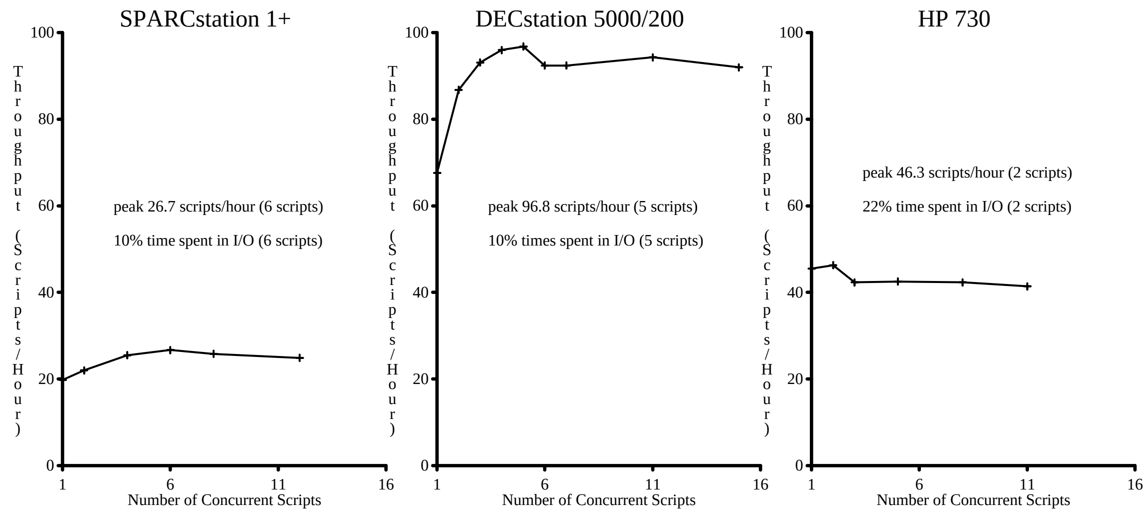

Figure 2.2: Sdet results. These figures show results from the SPEC SDM benchmark Sdet. Sdet varies the number of scripts running simultaneously but spends little time in I/O.

and deleting, as well as miscellaneous other UNIX utilities [SPEC91]. Sdet increases the number of scripts running concurrently until it reaches the system's maximum throughput, which is measured as the script-completion rate (scripts per hour). Sdet reports this maximum rate, along with a graph of throughput versus script concurrency (Figure 2.2). Once again, only a small percentage of time, 10%-22%, is spent in I/O.

2.4.2. Synthetic Benchmarks

Unlike application benchmarks that use standard programs, which in turn issue I/O, synthetic benchmarks exercise an I/O system by directly issuing read and write commands. By issuing reads and writes directly, synthetic benchmarks are able to generate more I/O-intensive workloads. However, synthetic benchmarks often yield less convincing results because they, unlike application benchmarks, do not perform useful work. I review four synthetic benchmarks here. Two of the most popular benchmarks are Bonnie and IOStone. I also create a third one to demonstrate a typical scientific I/O workload. Finally, I review an emerging benchmark for network file systems called LADDIS.

2.4.2.1. Bonnie

Bonnie measures I/O performance on a single file for a variety of simple workloads. One workload reads sequentially the entire file one character at a time while another writes the file

Workload	SPARCstation 1+		DECstation 5000		HP 730	
	KB/s	% I/O	KB/s	% I/O	KB/s	% I/O
Sequential Char Write	229	1%	300	23%	1285	2%
Sequential Block Write	558	63%	663	86%	1777	84%
Sequential Block Rewrite	230	74%	297	86%	604	87%
Sequential Char Read	193	2%	253	40%	995	15%
Sequential Block Read	625	63%	781	82%	2023	87%
Random Block Read	31 IOs/sec	78%	32 IOs/sec	86%	39 IOs/sec	95%

Table 2.4: Results from Bonnie. This table shows results from running Bonnie. A 100 MB file was used for all runs.

one character at a time. Other workloads exercise block-sized sequential reads, writes, or reads followed by writes (rewrites). The final workload uses three processes to simultaneously issue random I/Os. The size of the file is set by the evaluator and should be several times larger than the system's file cache, thus preventing the entire file from fitting in the cache. For each workload, Bonnie reports throughput, measured in KB per second or I/Os per second, and CPU utilization. I show results for the three systems in Table 2.4. Most of Bonnie's workloads are I/O-limited, though the character reads and writes are CPU-limited.

2.4.2.2. IOStone

IOStone is a synthetic I/O benchmark [Park90] based on system traces of Unix minicomputers and workstations [Ousterhout85, Hu86] and IBM mainframes [Smith78, Smith81]. Using 400 files totaling 1 MB, IOStone reads and writes data in patterns which approximate the locality found in [Ousterhout85]. One process performs all the accesses—no I/O parallelism is present. IOStone reports a single throughput result, measured in IOStones per second (Table 2.5). Though the file caches of the HP 730 and the SPARCstation 1+ are large enough to contain all of the data of IOStone, HP/UX and SunOS limit the number of files present in the file cache to approximately 300. Since IOStone touches 400 files, many references access the disk. IOStone runs much faster on the DECstation 5000, because Sprite imposes no such limit, and all accesses can be satisfied from the file cache.

System	IOStones/Second	Percent Time Spent in I/O
SPARCstation 1+	11,002	61%
DECstation 5000	50,905	26%
HP 730	20,409	81%

Table 2.5: Results from IOStone. This table shows results from running IOStone. HP/UX and SunOS limit the number of files resident in the file cache, so many references need to access the disk. Because Sprite allows all IOStone data files to reside in the file cache, its performance is 2-4 times better, and much less I/O limited, than the other systems.

2.4.2.3. Sample Scientific Workload

Andrew, SDM, IOStone, and Bonnie all target system development or workstation environments. Other application areas, such as scientific or supercomputing code, have substantially different workload characteristics [Miller91]. Typical scientific applications generally touch much more data and use much larger request sizes than workstation applications. To illustrate I/O performance for a supercomputing environment, I define two simple workloads: a large file read workload, which reads a 100 MB file in 128 KB units, and a large file write workload, which writes a 100 MB file in 128 KB units. In Table 2.6, I show results for the three system platforms.

2.4.2.4. LADDIS

Network file systems provide file service to a set of *client* computers, connected by a network. The computer providing this file service is called the *server*. One popular protocol for network file service is Sun Microsystem's NFS [Sandberg85]. In 1989, Shein, Callahan, and Woodbury created NFSStone, a synthetic benchmark to measure NFS performance [Shein89]. NFSStone generated a series of NFS file requests from a single client to stress and measure server performance. These operations included reads, writes, and various other file operations such as examining a file. The exact mix of operations was patterned after a study done by Sun [Sandberg85]; the sizes of the files were patterned after the study done in [Ousterhout85].

System	Large File Reads		Large File Writes	
	Throughput	% I/O	Throughput	% I/O
SPARCstation 1+	0.64 MB/s	99%	0.64 MB/s	98%
DECstation 5000	0.82 MB/s	92%	1.28 MB/s	98%
HP 730	1.98 MB/s	100%	1.86 MB/s	98%

Table 2.6: Results from two sample scientific workloads. This table shows the results obtained from running two simple workloads typical of scientific applications. *Large File Reads* consists of sequentially reading a 100 MB file in 128 KB units. *Large File Writes* consists of sequentially writing a 100 MB file in 128 KB units.

Later, Legato Systems refined NFSStone and called it NHFSStone to avoid copyright infringement. Nevertheless, NFSStone and NHFSStone had several problems: one client could not always fully stress a file server; different versions of the benchmarks abounded; file and block sizes were not realistic; and only SunOS clients could run them.

In 1990, seven companies, joined forces to create a NFS benchmark capable of stressing even the most powerful file servers. The result was *LADDIS*, named after the seven companies (Legato, Auspex, Digital Equipment Corporation, Data General, Interphase, and Sun). It will be incorporated as part of the SPEC SFS (System Level File Server) Suite of benchmarks. LADDIS is based on NHFSStone but, unlike NHFSStone, runs on multiple, possibly heterogeneous, clients and networks [Nelson92, Levitt92]. Like NHFSStone, LADDIS is a synthetic benchmark with a set mix of operations. As of this writing, LADDIS is still under development. Its current form includes default percentages for the mix of operations, although it allows the evaluator to modify that mix. In its final release, SPEC will likely standardize the operation mix. Currently, LADDIS can vary many parameters. For example, besides the percentage of each operation in the workload, LADDIS gives the evaluator the freedom 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, and the block size of I/O requests. Reads are weighted to produce 85% block-sized requests and 15% partial-block requests; Writes are weighted to produce 50% of each. Moreover, LADDIS touches many files, and each operation's data files are chosen randomly.

The reporting philosophy of LADDIS is quite similar to that of TPC-B. The preferred metric is a throughput (NFS operations per second) versus response time graph. As a more compact form, users may report the maximum throughput possible subject to a response time constraint of 50 ms. Like TPC-B, LADDIS scales according to the reported throughput—for every 100 NFS operations/second of reported throughput, capacity must increase by at least 1 GB—although it does not yet specify how to scale the rest of the workload parameters.

LADDIS is expected to be released by SPEC in 1993.

2.5. Critique of Current Benchmarks

In applying my list of benchmark goals from Section 2.2 to current I/O benchmarks, it can be seen that there is much room for improvement. I show a qualitative evaluation of today's I/O benchmarks in Figure 2.3 and make the following observations:

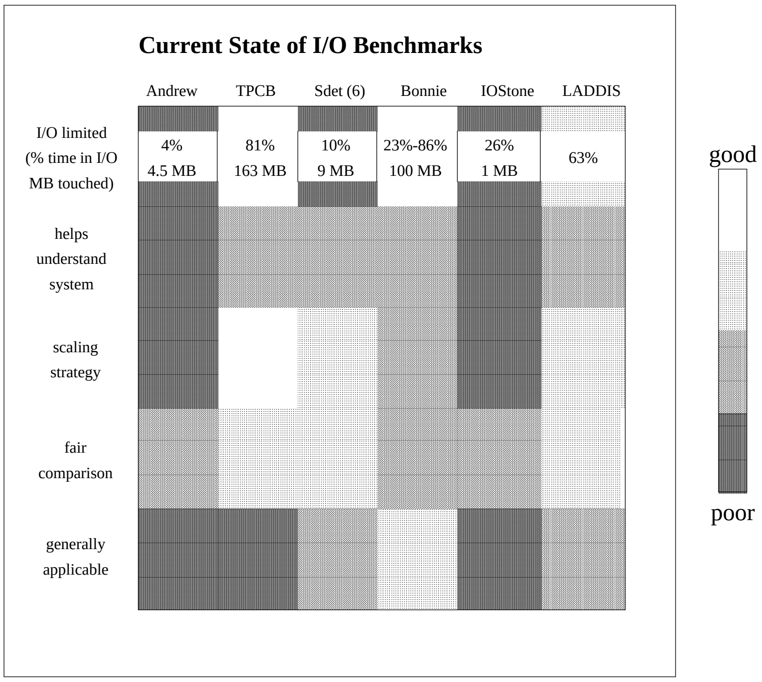

Figure 2.3: Current state of I/O benchmarks. This figure shows a qualitative evaluation of benchmarks used today to evaluate I/O systems. It can be seen that several are not I/O bound and that most do not provide understanding of the system, lack a well-defined scaling strategy, and are not generally applicable. The percent time spent in I/O was measured on the DECstation 5000/200 of Table 2.1. LADDIS was not available for execution as of this writing, but a pre-release 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%).

- *Many I/O benchmarks are not I/O limited*. On the DECstation 5000/200, Andrew, Sdet[2], and IOStone spend a quarter or less of their time doing I/O. Furthermore, many of the benchmarks touch very little data; IOStone touches only 1 MB of user data; Andrew touches only 4.5 MB. The best of the group is Bonnie, but even it has some tests that were CPU-bound.
- *Today's I/O benchmarks do not help in understanding system performance*. Andrew and IOStone give only a single, bottom-line performance result. TPC-B and Sdet fare somewhat better because they help the user understand system response under various loads, while Bonnie begins to help the user understand performance by running six workloads. These workloads show the differences in performance 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's occurring in parallel.
- *Many of today's I/O benchmarks have no scaling strategy*. For example, several made no provision for adjusting the workload to stress machines with larger file caches, for example. Without a well-defined scaling strategy, I/O benchmarks quickly grow obsolete. Several exceptions are noteworthy. 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 throughput. Sdet also has a superior scaling strategy, which varies the number of simultaneously active scripts until the peak performance is achieved. Automatically scaling certain aspects of a workload represents 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

[2] This refers to Sdet running at a peak throughput concurrency level of 6.

scale a few workload parameters, such as disk space or number of clients, but will leave other parameters inflexible.

- *Today's I/O benchmarks make fair comparisons for workloads identical to the benchmark, but do not help in drawing conclusions about the relative performance of machines for other workloads*. Ideally, the results from a benchmark could be applied to a wider range of workloads.
- *Today's I/O benchmarks focus on a narrow range of application range. For example, TPC-B is intended solely for benchmarking debit-credit, transaction processing systems.*

The generally poor state of I/O benchmarks suggests an urgent need for new benchmarks.

2.6. References

[Anon85] Anon and *et al.*, ''A Measure of Transaction Processing Power'', *Datamation, 31*, 7 (April 1, 1985), 112-118.

[Bechtolsheim90] A. V. Bechtolsheim and E. H. Frank, ''Sun's SPARCstation 1: A Workstation for the 1990s'', *Procedures of the IEEE Computer Society International Conference (COMPCON)*, Spring 1990, 184-188.

[Chen90] P. M. Chen and D. A. Patterson, ''Maximizing Performance in a Striped Disk Array'', *Proceedings of the 1990 ACM SIGARCH Conference on Computer Architecture*, Seattle WA, May 1990, 322-331.

[DECstation90] *DECstation 5000 Model 200 Technical Overview*, Digital Equipment Corporation, 1990.

[Gaede81] S. Gaede, ''Tools for Research in Computer Workload Characterization'', *Experimental Computer Performance and Evaluation*, 1981. D. Ferrari, M. Spadoni, eds..

[Gaede82] S. Gaede, ''A Scaling Technique for Comparing Interactive System Capacities'', *13th International Conference on Management and Performance Evaluation of Computer Systems*, 1982, 62-67. CMG 1982.

[HP730] *HP Apollo Series 700 Model 730 PA-RISC Workstation*, Hewlett-Packard, 1992.

[Hill86] M. D. Hill, S. J. Eggers, J. R. Larus, G. S. Taylor, G. Adams, B. K. Bose, G. A. Gibson, P. M. Hansen, J. Keller, S. I. Kong, C. G. Lee, D. Lee, J. M. Pendleton, S. A. Ritchie, D. A. Wood, B. G. Zorn, P. N. Hilfinger, D. Hodges, R. H. Katz, J. K. Ousterhout and D. A. Patterson, ‘‘Design Decisions in SPUR’’, *IEEE Computer 19*, 11 (November 1986).

[Horning91] R. Horning, L. Johnson, L. Thayer, D. Li, V. Meier, C. Dowdell and D. Roberts, ‘‘System Design for a Low Cost PA-RISC Desktop Workstation’’, *Procedures of the IEEE Computer Society International Conference (COMPCON)*, Spring 1991, 208-213.

[Howard88] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham and M. J. West, ‘‘Scale and Performance in a Distributed File System’’, *ACM Transactions on Computer Systems 6*, 1 (February 1988), 51-81.

[Hu86] I. Hu, ‘‘Measuring File Access Patterns in UNIX’’, *Performance Evaluation Review 14*, 2 (1986), 15-20. ACM SIGMETRICS 1986.

[Levitt92] J. Levitt, ‘‘Better Benchmarks are Brewing’’, *Unix Today!*, January 20, 1992.

[McDonell87] K. J. McDonell, ‘‘Taking performance evaluation out of the stone age’’, *Proceedings of the Summer Usenix Technical Conference*, Phoenix, Arizona, June 1987, 407-417.

[Miller91] E. L. Miller and R. H. Katz, ‘‘Input/Output Behavior of Supercomputing Applications’’, *Proceedings of Supercomputing ’91*, 1991, 567-576.

[Nelson92] B. Nelson, B. Lyon, M. Wittle and B. Keith, ‘‘LADDIS--A Multi-Vendor & Vendor-Neutral NFS Benchmark’’, *UniForum Conference*, January 1992.

[Nielsen91] M. J. K. Nielsen, ‘‘DECstation 5000 Model 200’’, *Procedures of the IEEE Computer Society International Conference (COMPCON)*, Spring 1991, 220-225.

[Ousterhout85] J. K. Ousterhout, H. Da Costa and *et al.*, ‘‘A Trace-Driven Analysis of the UNIX 4.2 BSD File System’’, *Operating Systems Review 19*, 5 (December 1985), 15-24. Proceedings of the 10th Symp. on Operating System Principles.

[Ousterhout90] J. K. Ousterhout, ‘‘Why aren’t operating systems getting faster as fast as hardware?’’, *Proceedings USENIX Summer Conference*, June 11-15, 1990, 247-256.

[Park90] A. Park and J. C. Becker, ‘‘IOStone: A synthetic file system benchmark’’, *Computer Architecture News 18*, 2 (June 1990), 45-52.

[Patterson88] D. A. Patterson, G. Gibson and R. H. Katz, ‘‘A Case for Redundant Arrays of Inexpensive Disks (RAID)’’, *International Conference on Management of Data (SIGMOD)*, June 1988, 109-116.

[SPEC91] *SPEC SDM Release 1.0 Manual*, System Performance Evaluation Cooperative, 1991.

[Sandberg85] R. Sandberg, D. Goldbert, S. Kleiman, D. Walsh and B. Lyon, ‘‘Design and Implementation of the Sun Network Filesystem’’, *Summer 1985 Usenix Conference*, 1985.

[Scott90] V. Scott, ‘‘Is Standardization of Benchmarks Feasible?’’, *Proceedings of the BUSCON Conference*, Long Beach, CA, Feb 14, 1990, 139-147.

[Seltzer92] M. Seltzer and M. Olson, ‘‘LIBTP: Portable, Modular Transactions for UNIX’’, *USENIX 1992?*, January 1992.

[Shein89] B. Shein, M. Callahan and P. Woodbuy, ‘‘NFSStone--A Network File Server Performance Benchmark’’, *Proceedings of the USENIX Summer Technical Conference 1989*, , 269-275.

[Smith78] A. J. Smith, ‘‘Sequentiality and Prefetching in Database Systems’’, *ACM Transactions on Database Systems 3*, 3 (1978), 223-247.

[Smith81] A. J. Smith, ‘‘Analysis of Long Term File Reference Patterns for Application to File Migration Algorithms’’, *IEEE Transactions on Software Engineering SE-7*, No. 4 (1981), 403-417.

[TPCA89] *TPC Benchmark A Standard Specification*, Transaction Processing Performance Council, November 10, 1989.

[TPCB90] *TPC Benchmark B Standard Specification*, Transaction Processing Performance Council, August. 23, 1990.