Chapter 3

Workload Model

3.1. Overview

The core of any benchmark used in performance evaluation is the workload model. The goals of a good workload model are flexibility, generality, and simplicity [Ferrari72, Ferrari84]. Flexibility requires the model to allow for the selective tuning of individual workload features. Generality requires that a wide range of applications and programs be modeled accurately. Accuracy here refers to how closely the workload model is able to duplicate the performance of an application. And finally, simplicity requires that an application be defined easily in terms of the workload model. This chapter reviews I/O workload generators used in recent research, describes several I/O tracing studies, discusses in detail *Willy*[1], the workload generator used in this thesis, and relates how current I/O benchmarks can be accurately modeled by Willy.

3.2. I/O Workloads Used in Research

I/O workload generators outside the realm of benchmarking function primarily as aids in doing research on I/O architectures. These workloads fall into two classes: *open-system* workloads and *closed-system* workloads [Denning78].

Open-system workloads issue I/Os independently of I/O completions according to a fixed interarrival probability distribution. After one I/O is issued, the arrival generator waits a period of time, determined by the interarrival probability distribution, then repeats the process. Because new I/Os are issued independently of I/O completions, arbitrarily many I/Os could be in progress. When I/Os arrive faster than they finish, the number of I/Os in progress increases without limit, and the system is said to be in *saturation*.

The most common interarrival distribution is an exponential waiting time. An exponential waiting time between successive I/Os leads to I/Os arriving as what is called a Poisson process. Wilhelm uses this to derive a performance model for a disk system [Wilhelm77], while Geist and many others use this workload to analyze various disk-scheduling policies [Geist87, Oney75, Gotlieb73, Teorey72, Hofri80, Coffman72]. Arrival rate is usually the only workload parameter varied in these studies, though other parameters such as transfer size could also be varied.

In contrast to open-system workloads, closed-system ones issue I/Os by using a fixed number of I/O generators. Each generator issues an I/O, *waits until the I/O is completed*, pauses for another period of time, then repeats. Unlike open systems, new I/Os generated by a closed systems depend on the completion of the last request from that generator. Stone argues that this

[1] The name *Willy* is derived from the word *workload*.

method more closely approximates the way real computer systems operate [Stone72], since real systems usually have a fixed number of users who cannot issue their next I/O until their last I/O finishes. If the waiting time between an I/O completion and the next I/O is fixed at zero, then there will always be the same number of I/Os in the system; this is referred to as the *concurrency* or *load* on the system. A good example of a closed-system workload is Bodnarchuk and Bunt's NFS workload generator [Bodnarchuk91]. Their workload has many parameters, including the distribution of reads and writes and other NFS operations, read and write sizes, and which files to reference. The Solbourne Filesystem Benchmarks also use a closed-system workload model [Taylor90], the parameters for which include concurrency and request size. Chen and Lee also use a closed system workload in their studies on the disk array striping unit [Chen90, Lee91]; they add request size as another workload parameter.

3.3. I/O Tracing Studies

Another flavor of I/O workload is a trace. Several researchers have gathered I/O traces of various systems, and in this section, I review four important I/O trace studies covering three different computing environments: mainframe database systems, Unix engineering systems, and scientific supercomputing systems.

In 1976 to 1978, Smith and others analyzed traces gathered at a site running an Information Management System/360 (IMS) hierarchical database. The trace was gathered over a seven day period on a 200 MB database [Smith78]. Smith examined sequentiality in the traces to compute the benefits of prefetching. He found that less than 30% of all requests were sequential, much lower than studies done in engineering and scientific environments, though high enough to significantly improve performance through prefetching. Rodriguez-Rosell found that the IMS database generated accesses with little temporal locality [Rodriguez-Rosell76].

In 1985, Ousterhout *et al.* traced the I/Os done on three VAX 11/780 minicomputers at the University of California at Berkeley [Ousterhout85]. These computers were mainly used for software development, administration, and computer-aided design. The researchers found that over the 2-3 day tracing sample,

- I/O requirements were quite low; each active user required only 300-600 bytes per second of I/O, though bursts of I/O reached 100 KB per second.
- Most files were accessed sequentially in their entirety.
- Most files were short lived; half of all new files were deleted in 5 minutes.
- File caches would be effective in reducing disk accesses.

In 1991, Baker *et al.* performed a follow-up study to Ousterhout's tracing study [Baker91]. These researchers traced the experimental Sprite operating system cluster, which consisted of forty, 10-MIPS workstations at the University of California at Berkeley. These workstations were used in much the same way as the minicomputers in the original Berkeley study: software development, administration, and computer-aided design. The Sprite system, unlike the VAX 11/780, uses a shared, distributed file system spread over a network. In spite of 100 times more CPU power and a distributed file system, many notable trace characteristics were similar to the ones found in the original Berkeley study. For example, files were mostly accessed sequentially in their entirety, and many files were still short-lived. However, Baker *et al.* found that I/O requirements had increased an order of magnitude to 8 KB per second per active user and that large files had grown much larger to be multiple megabytes in size.

In 1991, Miller analyzed seven scientific applications running on a Cray Y-MP supercomputer at NASA Ames [Miller91]. These applications solved mostly computational fluid dynamic problems, such as climate modeling and aerodynamic turbulence. Miller reports a much more I/O intensive environment than did the Berkeley studies. Among the things he found were: data sizes were much larger, ranging between 10 to 200 MB; I/O sizes were larger, ranging from 16 to 400 KB; each application performed large amounts of I/O, from 60 MB to 20 GB; and I/O demands reached 70 MB per second. Like the Berkeley studies, however, the

NASA Ames applications tended to generate bursts of I/O and reference files sequentially.

3.4. Willy

In Chapter 2 two types of benchmarks were discussed: synthetic and application. We saw that application benchmarks often have difficulty stressing I/O systems. Application benchmarks are also harder to control than synthetic ones—if a change to a specific part of a workload needs to be made without perturbing the rest of the workload, synthetic benchmarks are much better suited than application workloads. For these reasons, I choose to use a parameterized synthetic benchmark to generate I/Os. This section describes in detail Willy, the workload generator used in this thesis, and how other benchmarks can be described in terms of Willy's parameters.

3.4.1. Workload Parameters

In this section, I describe three main categories of workload parameters: locality, request characteristics, and load.

3.4.1.1. Locality

Locality refers to the data location of each I/O. Due to caches, I/O performance is highly dependent on spatial and temporal locality [Denning70, Smith85]. As a result, the workload model must be able to generate workloads with various spatial and temporal localities. In describing locality, *working set* is used to refer to the data most commonly used by a program. The size of the working set is critical in determining how well caches work for an application. My workload model uses two parameters to characterize locality: *uniqueBytes* and *seqFrac*.

UniqueBytes refers to the total number of unique data bytes read or written in a workload—this is the total size of the data. Large values of uniqueBytes correspond to workloads with large working sets—any cache in the system will have to be larger to effectively cache the working set.

Rather than designate a fixed part of the data set as commonly used, Willy models a floating, commonly used data set. While creating a workload, Willy maintains a stack of the most recently used data. To decide which data to access, Willy chooses a depth in the stack and accesses the data at that depth. This method is based on the way most caches operate; most caches choose which data to keep and which to remove based on which was most recently used [Hill87]. This model is known as the LRU stack model [Rau79]. For this reason, workloads generated by Willy should closely mimic the cache behavior of real applications with the same values of uniqueBytes. For simplicity, the workload model fixes the average access depth in the LRU stack at half of uniqueBytes. Applications that significantly differ from this can compensate by increasing or decreasing uniqueBytes to match the average access depth. The distribution of the depth of access in the stack is modeled as a binomial distribution with a mean equal to half of uniqueBytes.

UniqueBytes only affects temporal locality; another parameter is needed to control spatial locality. The major type of spatial locality in I/O workloads is the presence of sequential accesses. Several of the tracing studies reviewed in Section 3.3 showed large fractions of accesses to be sequential [Ousterhout85, Baker91, Miller91]. *SeqFrac* controls the fraction of sequential accesses (seqFrac of 1.0 means all requests are sequential). Addresses for non-sequential accesses are chosen according to the discussion on hitDepth about the most recently used data stack. Addresses are chosen randomly for data being accessed for the first time (and hence do not appear in the data stack).

3.4.1.2. Request Characteristics

Two parameters define the characteristics of individual accesses. First, accesses can be either a read or a write. *ReadFrac* determines the fraction of requests that are reads. Each request is chosen to be a read or a write independently; runs of multiple reads or writes are not deliberately created.

The size of each request is determined by the parameter *sizeMean*. Sizes are distributed according to a binomial distribution with a mean of sizeMean. For the sake of simplicity, other types of distributions are not modeled.

3.4.1.3. Load

To control the load on the system, I choose the closed-system workload model mentioned in Section 3.2. A closed-system workload model involves two parameters: *processNum* and *cpuThink*. *ProcessNum* is the number of processes running simultaneously, also called concurrency [Chen90]. *CpuThink* is the time each process waits from the time an I/O finishes until it issues the next I/O. Because larger values of cpuThink only lessen the fraction of I/O in the workload, I fix cpuThink at zero; processes always issue new I/Os immediately after their last I/O completes.

3.4.2. Other Workload Issues

Besides these workload parameters, a workload model includes many other issues. In this section, I discuss these issues and the approach Willy takes for each.

The first issue is the performance metric that will be reported by Willy, and in this thesis, the main performance metric will be throughput. Average response time is easily calculated from average throughput in a closed system by Little's Law [Denning78]:

$$throughput = \frac{sizeMean}{cpuThink + responseTime} \times processNum$$

The next issue is the number of files accessed in the workload. Due to per-file locking, some file systems, such as Sprite [Rosenblum91] do not allow multiple processes to access a file simultaneously. To allow full use of the file system, my workload generator uses multiple files; the number of files is equal to the number of processes.

In order to keep the read or write property of an access independent of the address of the access, all writes are overwrites; they are not appended at the end of the file. This simplifies the workload model because the entire file is created before the workload starts, thus no new data needs to be allocated.

The next workload issue is the starting layout on disk. Ideally, researchers should start with a freshly made file system on an empty disk, perform a large, standard set of file creations, reads, writes, and file deletions, then run the benchmark. I leave this step for future research.

Address alignment often affects I/O performance. Most I/Os generated by real-world applications are aligned to the block size of the system. In keeping with that, addresses and sizes generated by this workload model are aligned to the block size, usually 4 KB or 8 KB.

The last workload issue addressed here is when to flush the file system cache, if at all. In this thesis, I first prime the file cache before measuring performance. To prime the file cache, I run the workload multiple times and discard the first run. This approach is commonly known as *warm start*.

3.5. Modeling Real Applications

In this section, I characterize the I/O patterns of real applications and benchmarks in terms of Willy's parameters. I discuss the benchmarks of Chapter 2 plus an I/O intensive sorting program and some I/O traces of scientific applications from [Miller91]. Table 3.1 shows the parameter values of these applications. Note the wide variation in the parameter values of these applications. Within the variation, however, there are several trends:

- The two file system benchmarks, Andrew and Sdet, touch small amounts data using small requests. Within these benchmarks, reads and writes occur in equal portions and sequentiality is moderate. High degrees of concurrency can be present, depending on how many simultaneous scripts are running.

Application	uniqueBytes (MB)	seqFrac	readFrac	sizeMean (KB)	processNum
Andrew	4.7	.77	.54	3	1
Sdet	8	.48	.56	2.4	4
TPC-B	47.9	.01	.51	4.4	1
ccm	11.6	.91	.51	32	1
gcm	229	1.00	.07	32	1
les	224	.07	.48	325	1
venus	55.2	.43	.65	456	1
sort	234	.99	.56	4.0	1
Minimum	1.8	.01	.07	2.4	1
Maximum	234	1.00	.65	456	4

Table 3.1: Workload characterization of benchmarks/applications. This table characterizes several benchmarks and applications in terms of this thesis' workload model. Ccm, gcm, les, and venus are three scientific applications run at NASA Ames Research Center [Miller91]. Ccm, gcm, and venus model atmospheric climates; les models aeronautical turbulence effects. The sort application sorted 4 files totaling 48 MB. Minimum and Maximum show the range of parameter values for these applications. Sdet was run with a concurrency of 5, which yields the maximum throughput on the DECstation 5000 of Table 2.1. More concurrently running scripts would increase the value of processNum, which is the maximum number of simultaneously occurring I/Os. TPC-B was run on the DECstation 5000/200, which delivered 13.2 transactions per second.

- The transaction processing benchmark TPC-B touches a large amount of data using small requests. Requests are mostly non-sequential.
- The scientific applications touch large amounts of data using large requests.

3.6. Representativeness of Willy

The most important question in developing a synthetic workload model is the question of representativeness [Ferrari84]. A synthetic workload should have enough parameters such that the performance of the synthetic workload is close to the performance of an application with the same set of parameter values. 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 a system in such a way that no synthetic workload (short of a full trace) could duplicate that I/O workload's performance. To show that my workload model captures the

Application	Throughput	Read Response Time	Write Response Time	Average Response Time
Sort	.20 MB/s	19.7 ms	1.6 ms	11.7 ms
Willy	.20 MB/s	20.0 ms	1.9 ms	11.0 ms
TPC-B	.13 MB/s	25.6 ms	1.3 ms	14.0 ms
Willy	.13 MB/s	22.1 ms	1.6 ms	12.3 ms

Table 3.2: Representativeness of Willy. This table shows how accurately Willy mimics the performance of two I/O-bound applications, Sort and TPC-B. All runs were done on a DECstation 5000 running Sprite. For each application, the synthetic workload used the parameter values given in Table 3.1.

important features of an I/O workload, this section compares the performance of I/O-bound applications to the performance of Willy with those applications' parameter values.

Out of the four application benchmarks run in this section (Andrew, TPC-B, Sdet, and Sort), only TPC-B and Sort are interesting to compare. Andrew and Sdet spend only a few percent of their running time performing I/O, making them uninteresting to compare. Another way to view this is the following: if an application spends 5% of its running time in I/O, then a workload model consisting of one parameter, cpu think time, would be able to model throughput within 95% accuracy, even without doing any I/O! The only application benchmarks which spend much of their time in I/O are TPC-B and Sort.

Table 3.2 compares Willy's performance on a DECstation 5000 with the performance of the applications being modeled. We see that both Sort and TPC-B can be modeled quite accurately using Willy with appropriate parameter values. 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.

In the next chapter, I develop a benchmark that explores the space of possible workloads, using Willy to measure the performance for each workload.

3.7. References

[Baker91] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff and J. K. Ousterhout, ‘‘Measurements of a Distributed File System’’, *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, October 1991.

[Bodnarchuk91] R. R. Bodnarchuk and R. B. Bunt, ‘‘A Synthetic Workload Model for a Distributed System File Server’’, *SIGMETRICS 1991*, May 1991.

[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.

[Coffman72] E. G. Coffman, L. A. Klimko and B. Ryan, ‘‘Analysis of Scanning Policies for Reducing Disk Seek Times’’, *SIAM J. Comput 1*, 3 (September 1972), 269-279.

[Denning70] P. J. Denning, ‘‘Virtual Memory’’, *ACM Computing Surveys 2* (September 1970), 153-190.

[Denning78] P. J. Denning and J. P. Buzen, ‘‘The Operational Analysis of Queueing Network Models’’, *ACM Compuing Surveys 10*, 3 (September 1978).

[Ferrari72] D. Ferrari, ‘‘Workload Characterization and Selection in Computer Performance Measurement’’, *IEEE Computer 5*, 4 (July/August 1972), 18-24.

[Ferrari84] D. Ferrari, ‘‘On the Foundations of Artificial Workload Design’’, *ACM SIGMETRICS 1984*, 1984, 8-14.

[Geist87] R. Geist and S. Daniel, ‘‘A Continuum of Disk Scheduling Algorithms’’, *ACM Transactions on Computer Systems 5*, 1 (February 1987), 77-92.

[Gotlieb73] C. C. Gotlieb and G. H. MacEwen, ‘‘Performance of Movable-Head Disk Storage Devices’’, *Journal of the Association for Computing Machinery (J. ACM) 20*, 4 (October 1973), 604-623.

[Hill87] M. D. Hill, ‘‘Aspects of Cache Memory and Instruction Buffer Performance’’, UCB/Computer Science Dpt. 87/381, November 1987. Ph.D. dissertation.

[Hofri80] M. Hofri, ‘‘Disk Scheduling: FCFS vs. SSTF Revisited’’, *Communications of the ACM 23*, 11 (November 1980), 645-653.

[Jones81] D. A. Jones, ‘‘Disk Workload Characterization from Event Trace Analyses’’, *Proceedings of the 8th International Symposium on Computer Performance Modelling, Measurement and Evaluation*, November 1981, 301-313. Performance 81.

[Lee91] E. K. Lee and R. H. Katz, ‘‘An Analytic Performance Model of Disk Arrays and its Applications’’, UCB/Computer Science Dpt. 91/660, University of California at Berkeley, 1991.

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

[Oney75] W. C. Oney, ‘‘Queueing Analysis of the Scan Policy for Moving-Head Disks’’, *Journal of the Association for Computing Machinery (J. ACM) 22*, 3 (July 1975), 397-412.

[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.

[Rau79] B. R. Rau, ‘‘Program Behavior and the Performance of Interleaved Memories’’, *IEEE Transactions on Computers C-28*, 3 (March 1979), 191-199.

[Rodriguez-Rosell76] J. Rodriguez-Rosell, ‘‘Empirical Data Reference Behavior in Data Base Systems’’, *IEEE Computer 9*, 11 (November 1976), 9-13.

[Rosenblum91] M. Rosenblum, Sprite File Locking, personal communication, October 29, 1991.

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

[Smith85] A. J. Smith, ‘‘Disk Cache-Miss Ratio Analysis and Design Considerations’’, *ACM Transactions on Computer Systems 3*, 3 (August 1985), 161-203.

[Stone72] D. L. Stone and R. Turner, ‘‘Disk Throughput Estimation’’, *Proceedings of the ACM Annual Conference*, August 1972, 704-711.

[Taylor90] D. Taylor, ‘‘Solbourne Filesystem Benchmarks--User Guide’’, Solbourne technical report, October 29, 1990.

[Teorey72] T. J. Teorey and T. B. Pinkerton, ‘‘A Comparative Analysis of Disk Scheduling Policies’’, *Communications of the ACM 15*, 3 (March 1972), 177-184.

[Wilhelm77] N. C. Wilhelm, ‘‘A General Model for the Performance of Disk Systems’’, *Journal of the Association for Computing Machinery (J. ACM) 24*, 1 (January 1977), 14-31.