The Architectural Implications of Pipeline and Batch Sharing in Scientific Workloads

Douglas Thain, John Bent, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny

Technical Report 1463
22 January 2002

Computer Sciences Department
University of Wisconsin, Madison

Abstract

We present a study of six batch-pipelined scientific workloads. Whereas other studies focus on the behavior of a single application, we characterize an emerging type of workload which consists of pipelines of sequential processes that use file storage for communication and also share significant data across a batch. This study includes measurements of the memory, CPU, and I/O requirements of individual components as well as analyses of I/O sharing within complete batches, as well as a discussion of the architectural ramifications of these new types of workloads.

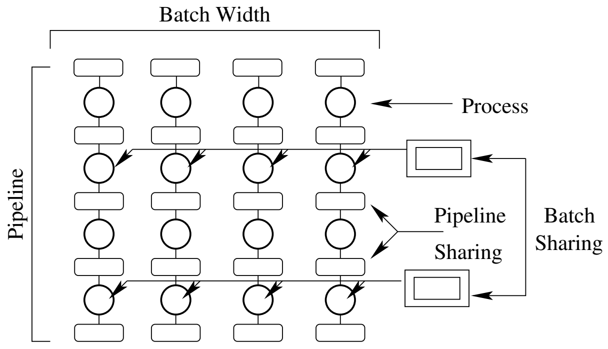


Figure 1. A Batch-Pipelined Workload

1. Introduction

For many years, researchers have understood the importance of studying workload characteristics in order to evaluate their impact on current and future systems architecture [6, 18, 20, 27, 31]. Most of these previous application studies have focused on the detailed behavior of single applications, whether sequential or parallel. For example, the caching behavior of the SPEC workloads has long been a topic of intense scrutiny [7, 12], and the communication characteristics of parallel applications has similarly been well documented [9, 37, 36].

However, applications are no longer used in isolation in production settings. Whether in disparate settings such as graphics rendering [17], video production [33], or computational science [13, 14], the desired end-result is often the product of a group of applications, each of which may be run hundreds or thousands of times with varied inputs.

We refer to such workloads as *batch-pipelined*, as illustrated within Figure 1. A batch-pipelined workload is composed of several independent pipelines; each pipeline contains sequential processes that communicate with the preceding and succeeding processes via private data files. Shared input files are used by all of the pipelines in various stages. As the figure suggests, a workload is generally submitted in large batches with all of the pipelines incidentally synchronized at the beginning. However, each pipeline is logically distinct and may correctly execute faster or slower than its siblings.

The key difference between studying the behavior of a single application and that of a batch-pipelined workload is that the *sharing behavior* of the batch-pipelined workload must be understood. For example, when many instances of the same application are run, the same executable and potentially many of the same input files are used. Thus, to realistically capture the full diversity of these production workloads, one must study the behavior of the entire pipeline and account for the effects of sharing.

In this paper, we present a study of six production scientific workloads. We collected these application pipelines from diverse fields of computational science, including astronomy, biology, geology, and physics; we believe the applications are representative of a broad class of important workloads. We first present a basic characterization of the computational, memory, and I/O demands of these workloads. We find that although individually, a single pipeline does not place a tremendous load on system resources, in combination the loads can be overwhelming. We focus particularly upon the I/O behavior of the workloads, because it is the primary source of sharing. We then characterize the sharing that occurs in the workloads, by breaking I/O activity into three categories: *endpoint*, which represents the input and final output, *pipeline-shared*, which is shared in a write-then-read fashion within a single pipeline, and *batch-shared*, which is comprised of input I/O is shared across pipelines. Through this characterization, we show that shared I/O is the dominant component of all I/O traffic.

Most importantly, we study the implications for systems architecture, both from a hardware and software perspective. We find that the architecture is strongly influenced by the different types of I/O traffic, and analogous to processors that differentiate instruction and data streams, systems must segregate I/O traffic in order to be able to successfully scale under these workloads. Further, as workloads such as these are likely to be run in wide-area peer-to-peer computing systems [11], we also study their behavior in such environments; extrapolating from current technology levels, we find that while cluster interconnect bandwidth will likely keep up with processor technology advancements, wide-area bandwidth will likely become the bottleneck, potentially limiting such collaborative computing efforts.

The rest of this paper is organized as follows. In Section 2, we describe the general characteristics of batch-pipelined workloads as well as our specific application pipelines. In Section 3, we describe our experimental method, and in Section 4, 5, and 6, we analyze the data and discuss architectural implications. We discuss related work in Section 7, and conclude in Section 8.

2. Applications

The applications that we characterize were chosen from a range of scientific disciplines. Our selection criteria were that the applications are attacking a major scientific objective, are composed of sequential applications, and require a scalable computing environment to accomplish high throughput. We focus mostly on six applications but in some measurements we include SETI@home [34] as a point of reference. With guidance from users, we chose workloads and input parameters to correspond to production use. For example, for CMS and AMANDA, we took the actual inputs used in current production runs. (More details on the exact inputs will be included in the final version) A summary of the applications is found in Figure 2.

Across these applications, we have observed the following characteristics behaviors:

An inverted hourglass storage profile. Small initial inputs are generally created by humans or initialization tools and expanded by early stages into large intermediate results. These intermediates are often reduced by later stages to small results to be interpreted by humans or incorporated into a database. Intermediate data, which often serves as checkpoint or cached values, may be ephemeral in nature.

Multi-level working sets. Users can easily identify large logical collections of data needed by an application, such as calibration tables and physical constants. However, in a given execution, applications tend to select a small working set of which users are not aware; this has significant consequences for data replication and caching techniques.

Significant data sharing. Although each application has a large configuration space, users submit large numbers of very similar jobs that access similar working sets. This property can be exploited for efficient wide-area distribution over modest communication links.

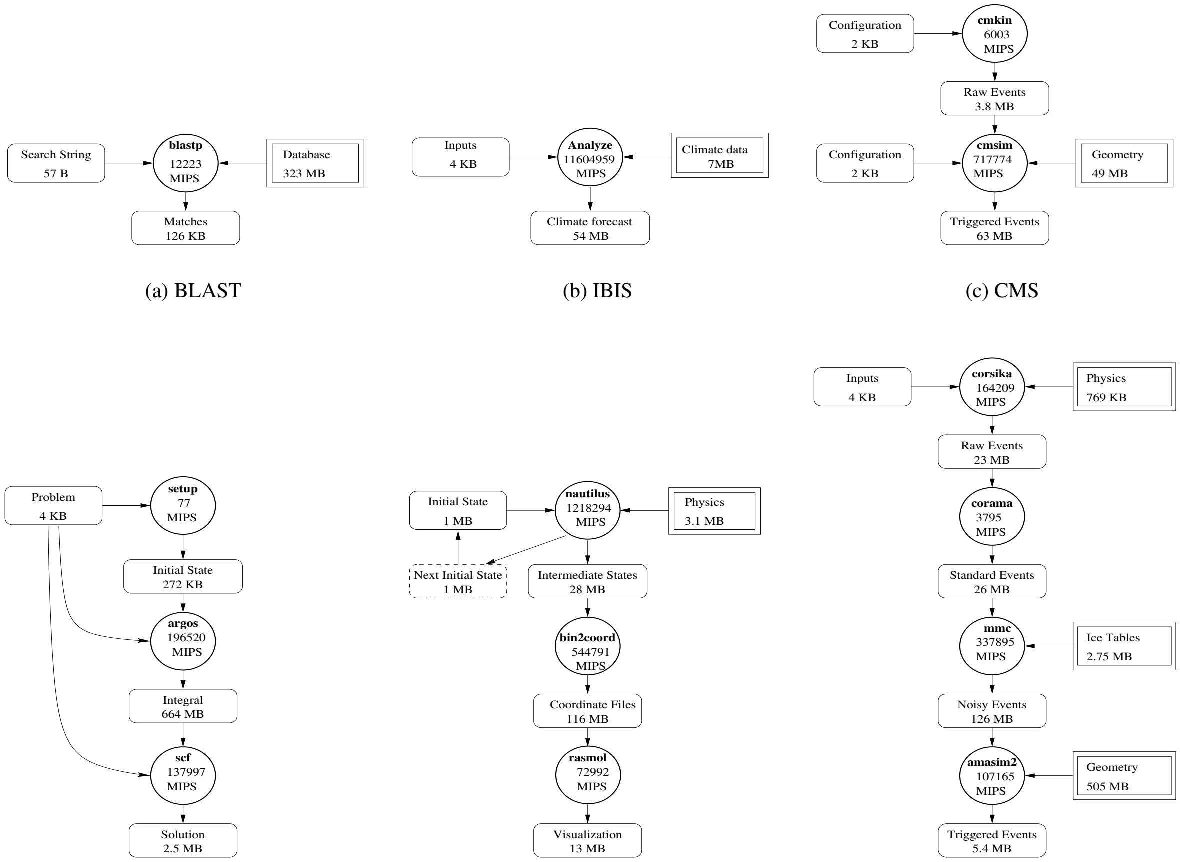
3. Method

For each application, we capture its CPU, memory and I/O behavior. The CPU and memory behavior is tracked with available hardware counters and statistics. To instrument I/O behavior, we insert a shared library that replaces the I/O routines in the standard library. For each explicit I/O event requested by the application, the library records an event marking the start and end of the operation, the instruction count, and other details about the I/O request. This technique can be applied to any application that is dynamically linked. Care is taken so as to avoid additional overheads due to tracing.

Access to memory-mapped files is traced with a user-level paging technique using the POSIX **mprotect** feature in a manner similar to that of Tempest [28]. Access to memory-mapped regions generates a user-level page fault (SIGSEGV) that may be handled and traced by the shared library. Only one application (BLAST) uses memory-mapped I/O. In the analysis that follows, page faults are considered equivalent to explicit read operations of one page size and non-sequential access to memory-mapped pages is recorded as an explicit seek operation.

4. Workload Analysis

An overview of the resources consumed by each application is given in Figure 3. These applications have a wide variance in run times on current hardware, ranging from a little more than a minute (BLAST) to a little more than a day (IBIS). Considered individually, these applications spend the majority of time consuming CPU rather than I/O, not requiring nearly the I/O capability as suggested by the Amdahl/Case rule of thumb (1 Mbit per second of I/O bandwidth per MIPS of CPU) [4]. Memory requirements and



(a) BLAST (b) IBIS (c) CMS

(d) HF (e) Nautilus (f) AMANDA

Figure 2. Application Schematics *These schematics summarize the structure of each application pipeline. Circles indicate individual processes, labeled with the name and user CPU time. Rounded boxes indicate data private to a pipeline. Double boxes indicate data shared between pipelines in a batch. Arrows indicate data flow. Labeled arrows indicate that actual I/O performed differs from the static file size.*

· **BLAST** *[3] searches genomic databases for matching proteins and nucleotides. Both queries and archived data may include errors or gaps, and acceptable match similarity is parameterized. Exhaustive search is often necessary. A single executable,* `blastp`*, reads a query sequence, searches through a shared database, and outputs matched sequences.*

· **IBIS** *[15] is a global-scale simulation of Earth systems. IBIS simulates effects of human activity on the global environment, i.e., global warming.* `ibis` *performs the simulation and emits a series of snapshots of the global state.*

· **CMS** *[13] is a high-energy physics experiment to begin operation in 2006. CMS testing software is a two-stage pipeline; the first stage,* `cmkin`*, given a random seed, generates and models the behavior of particles accelerated by the ring. The output is a set of events that are fed to* `cmsim`*, which simulates the response of the detector. The final output represents events that exceed the triggering threshold of the detector.*

· **Nautilus** *[35] is a simulation of molecular dynamics. An input configuration describes molecules within a three-dimensional space. Newton's equation is solved for each particle. Incremental snapshots are taken to periodically capture particle coordinates. The final snapshot is often passed back to the program as an initial configuration for another simulation. Eventually, all snapshots are converted into a standard format using* `bin2coord` *and consolidated into images using* `rasmol`.

· **Messkit Hartree-Fock (HF)** *[8] is a simulation of the non-relativistic interactions between atomic nuclei and electrons, allowing the computation of properties such as bond strengths and reaction energies. Three distinct executables comprise the calculation:* `setup` *initializes data files from input parameters,* `argos` *computes and writes integrals corresponding to the atomic configuration, and* `scf` *iteratively solves the self-consistent field equations.*

· **AMANDA** *[14] is an astrophysics experiment designed to observe cosmic events such as gamma-ray bursts by collecting the resulting neutrinos through their interaction with the Earth's mass. The first stage of the calibration software,* `corsika`*, simulates the production of neutrinos and the primary interaction which creates showers of muons.* `corama` *translates the output into a standard high-energy physics format.* `mmc` *propagates the muons through the earth and ice while introducing noise from atmospheric sources. Finally,* `amasim2` *simulates the response of the detector to incident muons.*

Application		Real Time (s)	Millions of Instructions Integer	Float	Burst	Memory (MB) Text	Data	Share	I/O Traffic MB	Ops	I/O Rates MB/MIPS	MB/s
setiathome	seti	41587.1	1953084.8	1523932.2	4.6	0.1	15.7	1.1	75.8	417260	0.00002	0.00
blast	blastp	264.2	12223.5	0.2	0.1	2.9	323.8	2.0	330.1	88671	0.02701	1.25
ibis	ibis	88024.3	7215213.8	4389746.8	104.7	0.7	24.0	1.4	336.1	110802	0.00003	0.00
cms	cmkin	55.4	5260.4	743.8	6.1	19.4	5.0	2.6	7.5	988	0.00125	0.14
	cmsim	15595.0	492995.8	225679.6	0.4	8.7	70.4	4.3	3798.7	1915559	0.00529	0.24
	total	15650.4	498256.1	226423.4	0.4	19.4	70.4	4.3	3806.2	1916546	0.00525	0.24
hf	setup	0.2	76.6	0.4	0.0	0.5	4.0	1.3	9.1	2953	0.11870	56.43
	argos	597.6	179766.5	26760.7	0.8	0.9	2.5	1.4	663.8	254713	0.00321	1.11
	scf	19.8	132670.1	5327.6	0.2	0.5	10.3	1.3	3983.4	765562	0.02887	201.06
	total	617.6	312513.2	32088.6	0.3	0.9	10.3	1.4	4656.3	1023228	0.01351	7.54
nautilus	nautilus	14047.6	767099.3	451195.0	18.6	0.3	146.6	1.2	270.6	65523	0.00022	0.02
	bin2coord	395.9	263954.4	280837.2	4.2	0.0	2.2	1.4	403.3	129727	0.00074	1.02
	rasmol	158.6	69612.8	3380.0	1.9	0.4	4.9	1.7	128.7	38431	0.00176	0.81
	total	14602.2	1100666.5	735412.2	7.9	0.4	146.6	1.7	802.7	233681	0.00044	0.05
amanda	corsika	2187.5	160066.5	4203.6	26.4	2.4	6.8	1.4	24.0	6225	0.00015	0.01
	corama	41.9	3758.4	37.9	0.3	0.5	3.2	1.1	49.4	12693	0.01300	1.18
	mmc	954.8	330189.1	7706.5	0.3	0.4	22.0	4.9	154.4	1141633	0.00046	0.16
	amasim2	3601.7	84783.8	20382.7	143.7	22.0	256.6	1.6	550.3	733	0.00523	0.15
	total	6785.9	578797.8	32330.7	0.5	22.0	256.6	4.9	778.0	1161275	0.00127	0.11

Figure 3. Resources Consumed

Application		Total I/O Files	Traffic	Unique	Static	Reads Files	Traffic	Unique	Static	Writes Files	Traffic	Unique	Static
setiathome	seti	14	75.77	3.02	3.02	12	71.62	0.72	1.04	11	4.15	2.36	2.68
blast	blastp	11	330.11	323.59	586.21	10	329.99	323.46	586.09	1	0.12	0.12	0.12
ibis	ibis	136	336.08	73.64	73.64	132	140.08	73.48	73.48	118	196.00	66.66	66.66
cms	cmkin	4	7.49	3.88	3.88	2	0.00	0.00	0.00	2	7.49	3.88	3.88
	cmsim	16	3798.74	116.00	126.18	11	3735.24	52.86	63.05	5	63.50	63.13	63.13
	total	17	3806.22	119.88	130.06	11	3735.24	52.86	63.05	6	70.98	67.01	67.01
hf	setup	5	9.13	0.40	0.40	3	5.44	0.26	0.26	3	3.69	0.39	0.40
	argos	5	663.76	663.75	663.97	2	0.04	0.03	0.26	4	663.73	663.74	663.97
	scf	11	3983.40	664.61	664.61	9	3979.33	663.79	664.60	8	4.07	2.50	2.69
	total	11	4656.30	666.54	666.54	9	3984.81	663.80	664.60	9	671.49	666.53	666.53
nautilus	nautilus	17	270.64	32.90	32.90	7	4.25	4.25	4.25	10	266.40	28.66	28.66
	bin2coord	247	403.27	273.87	273.87	123	152.78	152.66	152.66	241	250.49	249.39	249.39
	rasmol	242	128.75	128.76	128.76	124	115.87	115.88	115.88	120	12.88	12.88	12.88
	total	501	802.66	435.48	435.48	252	272.90	272.74	272.74	369	529.76	290.94	290.94
amanda	corsika	8	23.96	23.96	23.96	5	0.76	0.75	0.75	3	23.21	23.21	23.21
	corama	6	49.37	49.37	49.37	3	23.17	23.17	23.17	3	26.20	26.20	26.20
	mmc	11	154.36	154.36	154.36	9	28.92	28.92	28.92	2	125.43	125.43	125.43
	amasim2	29	550.35	550.40	635.78	27	545.04	545.09	630.47	3	5.31	5.31	5.31
	total	46	778.04	778.09	863.42	40	597.89	597.96	683.32	7	180.14	180.11	180.11

Figure 4. I/O Volume

Appl.	Open (%)	Dup (%)	Close (%)	Read (%)	Write (%)	Seek (%)	Stat (%)	Other (%)
seti	64595 (15.5)	0 (0.0)	64596 (15.5)	64266 (15.4)	32872 (7.9)	63154 (15.1)	127742 (30.6)	15 (0.0)
blastp	18 (0.0)	11 (0.0)	18 (0.0)	84547 (95.3)	1556 (1.8)	2478 (2.8)	37 (0.0)	5 (0.0)
ibis	1044 (0.9)	0 (0.0)	1044 (0.9)	26866 (24.2)	28985 (26.2)	51527 (46.5)	1208 (1.1)	122 (0.1)
cmkin	2 (0.2)	0 (0.0)	2 (0.2)	2 (0.2)	492 (49.8)	479 (48.5)	8 (0.8)	2 (0.2)
cmsim	17 (0.0)	0 (0.0)	16 (0.0)	952859 (49.7)	18468 (1.0)	944125 (49.3)	47 (0.0)	24 (0.0)
total	19 (0.0)	0 (0.0)	18 (0.0)	952861 (49.7)	18960 (1.0)	944604 (49.3)	55 (0.0)	26 (0.0)
setup	6 (0.2)	0 (0.0)	6 (0.2)	1061 (35.9)	735 (24.9)	1118 (37.9)	19 (0.6)	6 (0.2)
argos	3 (0.0)	0 (0.0)	3 (0.0)	8 (0.0)	127569 (50.1)	127106 (49.9)	18 (0.0)	4 (0.0)
scf	34 (0.0)	0 (0.0)	34 (0.0)	509642 (66.6)	922 (0.1)	254781 (33.3)	121 (0.0)	18 (0.0)
total	43 (0.0)	0 (0.0)	43 (0.0)	510711 (49.9)	129226 (12.6)	383005 (37.4)	158 (0.0)	28 (0.0)
nautilus	497 (0.8)	0 (0.0)	488 (0.7)	1095 (1.7)	62573 (95.5)	188 (0.3)	678 (1.0)	1 (0.0)
bin2coord	1190 (0.9)	6977 (5.4)	12238 (9.4)	33623 (25.9)	65109 (50.2)	3 (0.0)	407 (0.3)	10141 (7.8)
rasmol	359 (0.9)	22 (0.1)	517 (1.3)	29956 (77.9)	3457 (9.0)	1 (0.0)	252 (0.7)	3850 (10.0)
total	2046 (0.9)	6999 (3.0)	13243 (5.7)	64674 (27.7)	131139 (56.1)	192 (0.1)	1337 (0.6)	13992 (6.0)
corsika	13 (0.2)	0 (0.0)	13 (0.2)	199 (3.2)	5943 (95.5)	8 (0.1)	36 (0.6)	10 (0.2)
corama	4 (0.0)	0 (0.0)	4 (0.0)	5936 (46.8)	6728 (53.0)	2 (0.0)	12 (0.1)	4 (0.0)
mmc	8 (0.0)	0 (0.0)	9 (0.0)	29906 (2.6)	1111686 (97.4)	0 (0.0)	7 (0.0)	7 (0.0)
amasim2	30 (4.1)	0 (0.0)	28 (3.8)	577 (78.7)	24 (3.3)	4 (0.5)	57 (7.8)	10 (1.4)
total	55 (0.0)	0 (0.0)	54 (0.0)	36618 (3.2)	1124381 (96.8)	14 (0.0)	112 (0.0)	31 (0.0)

Figure 5. I/O Instruction Mix

		Endpoint I/O (MB)				Pipeline I/O (MB)				Batch I/O (MB)			
Application		Files	Traffic	Unique	Static	Files	Traffic	Unique	Static	Files	Traffic	Unique	Static
setiathome	seti	2	0.34	0.34	0.34	12	75.43	2.68	2.68	0	0.00	0.00	0.00
blast	blastp	2	0.12	0.12	0.12	0	0.00	0.00	0.00	9	329.99	323.46	586.09
ibis	ibis	20	179.92	53.97	53.97	99	148.27	12.69	12.69	17	7.89	6.98	6.98
cms	cmkin	2	0.07	0.07	0.07	1	7.42	3.81	3.81	1	0.00	0.00	0.00
	cmsim	6	63.50	63.13	63.13	1	5.56	3.81	3.81	9	3729.67	49.04	59.24
	total	6	63.56	63.20	63.20	2	12.99	7.62	7.62	9	3729.67	49.04	59.24
hf	setup	3	0.14	0.14	0.14	2	8.99	0.26	0.26	0	0.00	0.00	0.00
	argos	3	1.81	1.81	1.81	2	661.95	661.93	662.17	0	0.00	0.00	0.00
	scf	3	0.01	0.01	0.01	7	3983.39	664.59	664.59	1	0.00	0.00	0.00
	total	3	1.96	1.94	1.94	7	4654.34	664.59	664.59	1	0.00	0.00	0.00
nautilus	nautilus	6	1.18	1.10	1.10	9	266.32	28.66	28.66	2	3.14	3.14	3.14
	bin2coord	1	0.00	0.00	0.00	241	403.25	273.85	273.85	5	0.02	0.01	0.01
	rasmol	119	12.88	12.88	12.88	120	115.79	115.79	115.79	3	0.08	0.09	0.09
	total	124	14.06	13.99	13.99	369	785.37	418.25	418.25	8	3.24	3.24	3.24
amanda	corsika	2	0.04	0.04	0.04	3	23.17	23.17	23.17	3	0.75	0.75	0.75
	corama	3	0.00	0.00	0.00	3	49.37	49.37	49.37	0	0.00	0.00	0.00
	mmc	0	0.00	0.00	0.00	6	151.63	151.63	151.63	5	2.73	2.73	2.73
	amasim2	5	5.31	5.31	5.31	2	40.00	40.00	125.43	22	505.04	505.04	505.04
	total	6	5.22	5.21	5.21	11	264.31	264.29	349.69	29	508.52	508.52	508.52

Figure 6. I/O Roles

program sizes are all quite modest in comparison to total I/O volume.

Figure 4 details the I/O volume produced by each application. We make several observations. Although these applications are conceived as a pipeline of multiple stages, they are not connected by simple data streams. Rather, each makes complex read/write use of the file system, as indicated by the number of files each accesses. SETI, CMS, HF, and to a lesser degree, BLAST, all read input data multiples times. Over-writing of output data is also found in all pipelines with the exception of AMANDA. Output overwriting is usually done to update application-level checkpoints in place. We are somewhat alarmed to observe that such checkpoints are unsafely written directly over existing data, rather than written to a new file and atomically replaced with `rename`. Several pipelines are distributed with large collections of data that may be of use to many runs. However, any typical run only accesses a small portion common to similar runs. For example, the static size of the BLAST dataset exceeds the unique amount read by the application by 45%.

The distribution of "I/O instructions" is given in Figure 5. Two features stand out. First, a high degree of random access is present throughout, with the exception of AMANDA. This results from the nature of the data files accessed by the programs, generally with complex, self-referencing, internal structure, and contradicts many file system studies which indicate the dominance of sequential I/O [5]. Second, a very large number of `opens` are issued relative to the number of files actually accessed. Typically designed on standalone workstations, these applications are not optimized for the realities of distributed computing, where opening a file for access can be many times more expensive than issuing a read or write. The `Other` column sums a number of generally uncommon operations such as `ioctl` and `access`. The high numbers in this column reflect the fact that `bin2coord` and `rasmol` are driven by shell scripts which perform many `readdir` operations.

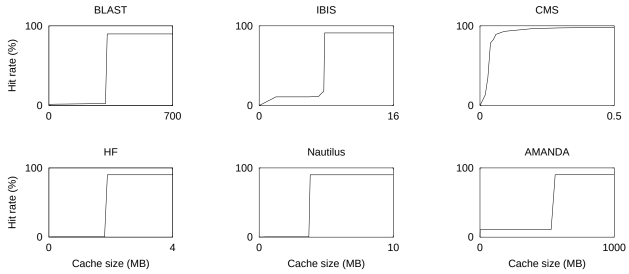


Figure 7. Batch Working Set. *These figures show the hitrate of the batch I/O traffic as a function of cache size with a batch width of 10. Notice that most of these working sets are very small and even the largest as seen in AMANDA is only slightly more than half of a gigabyte. Further notice that maximum hit rates are achieved with cache sizes that are relatively small compared to the total batch I/O traffic.*

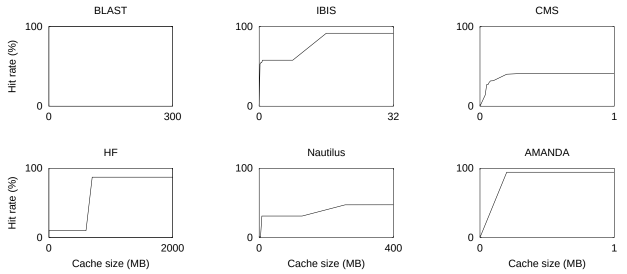
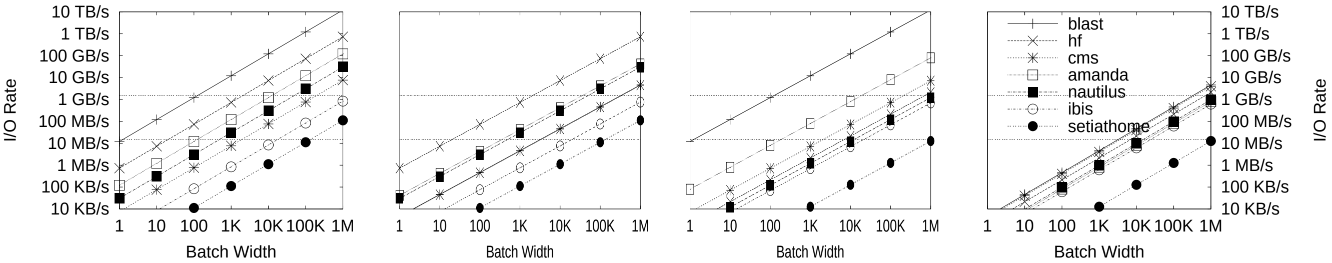


Figure 8. Pipeline Working Set. *These figures show the hitrate of the private I/O traffic as a function of cache size. AMANDA is anomolous here because it has no pipelined I/O and thus needs no pipeline cache at all. For most of the other applications, notice that a small cache size relative to the total private I/O of the application can achieve maximal hit rates. The exception is Nautilus which has an early plateau but is not maximized until its cache is almost the size of the total traffic.*

5. Sharing Analysis

To characterize the different types of sharing in batch-pipelined workloads, we have divided the I/O traffic into three roles. *Endpoint* traffic consists of the initial inputs and final outputs that are unique to each pipeline. They must be read from and written to the central site regardless of the system design. *Pipeline* traffic consists of intermediate data passed between pipeline stages. *Batch* traffic consists of input data that are identical across all pipelines.

Through our understanding of each application, we identified every file accessed as either endpoint, pipeline, or batch, and computed the traffic performed in each category, as shown in Figure 6. We immediately see that comparatively little traffic is needed at the endpoints; the bulk is either pipeline or batch, depending on the application. To examine the sharing potential of each workload, we separated the pipeline and batch I/O traces and replayed them over cache simulators of various sizes. These results are shown in Figures 7 and 8. What is evident from these results is that the working sets (both batch and pipeline) of each of these applications is very low relative to their total I/O demands. This is an encouraging result as it shows that maximum hit rates can be achieved with small caches and in most cases this maximal hit rate will exceed 90%. Note that the size of the executable files is included implicitly in these calculations.



All Data **Endpoint and Pipeline** **Endpoint and Batch** **Endpoint Only**

Figure 9. Scalability of I/O Roles

6. Architectural Implications

Each of these workloads are potentially infinite. In these problem domains, the ability to harness more computing power enables higher resolution, more parameters, and lower statistical uncertainties. Current users of these applications wish to scale up throughput by running hundreds or thousands simultaneously. At this scale, applications normally considered CPU-bound become I/O bound when considered in aggregate.

To give some idea of the growing envelope of current scientific computing, consider that in the spring of 2002, the CMS pipeline was used to simulate 5 million events divided into 20,000 pipelined jobs, consuming 6 CPU-years and producing a terabyte of output. This batch was only a small fraction attempted as a test run before full production begins in 2007. Successive yearly workloads are planned to grow. All the necessary code and data are published in authoritative form by the experiment's central site. Likewise, all simulation outputs must eventually be moved back for archival storage.

In this section, we will explore the general properties of computing and storage systems that may be built to satisfy these workloads. We will not explore detailed algorithms for data management, but instead consider the balance between the necessary resources.

6.1. Endpoint Scalability

We assume that each of these applications, like CMS, relies on a central site for the authenticity and archival of input and output data. Thus, ultimate scalability is limited by competition for this shared resource. However, we have demonstrated that actual endpoint I/O traffic is a relatively small fraction of the total for all of these applications. If we are able to eliminate all non-endpoint traffic from the endpoint server through techniques such as caching and replication then we may see significant gains in scalability.

Of course, traffic elimination must be carried out carefully. Pipeline-shared traffic may only be eliminated if it is truly of no use to the end user. Such intermediate data might be necessary to return for debugging or even for archival if the ability to reproduce it is questionable. Batch-shared may only be eliminated within the constraints of maintaining the consistency and authenticity of potentially changing input data. Traffic elimination cannot be done blindly without some consideration of how the data are actually used outside the computing system.

That said, we may consider the limits of a system for executing such workloads based on its ability to eliminate shared traffic. Figure 9 shows how each of the selected applications would scale in four systems each eliminating some category of traffic. We assume the presence of a buffering structure sufficient to completely overlap all CPU and I/O; figures assume a 2000 MIPS CPU and show MB per second of CPU time. Two horizontal lines show milestones in I/O bandwidth. The lower, at 15 MB/s, represents a capable commodity hard disk. The upper, at 1500 MB/s, represents a very aggressive storage server and network.

The leftmost graph shows the scalability of a system that carries all traffic to the endpoint server. In this discipline, a high end storage device is needed for systems of very modest size, and is even overwhelmed by two applications near n=100. Only IBIS and SETI would be able to scale to n=100,000. If batch-shared traffic is eliminated, we will make significant improvements in CMS and Nautilus, as shown in the second graph. On the other hand, if pipeline-shared traffic is eliminated, we observe significant gains for SETI, HF, and Nautilus, as shown in the third. If only endpoint I/O is performed, then we reach the limit shown in the rightmost graph. All of the applications shown could scale over 1000 workers with modest storage, and over 100,000 with high-end storage. SETI alone could potentially scale to 1 million CPUs, an indicator of its specialized design for wide-area deployment.

6.2 Hardware Architecture

To reach these scalability limits, we must provide a hardware architecture that localizes the effects of each of the I/O

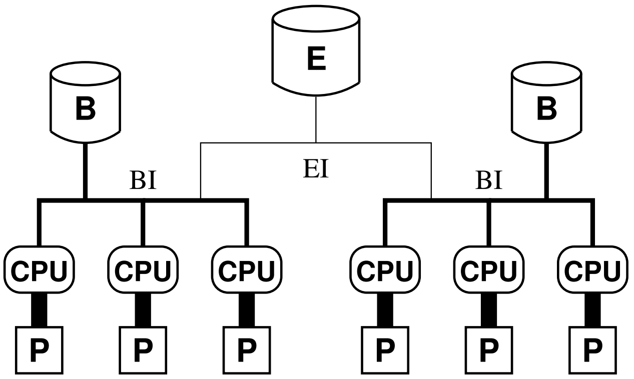


Figure 10. A General I/O Architecture

Each storage device in this architecture corresponds to the three roles of I/O: pipeline-shared (P), batch-shared (B), and endpoint (E). Each CPU owns a private pipeline-shared device, is connected to the batch-shared storage by a batch interconnect (BI) and then to the endpoint storage by an endpoint interconnect (EI). We consider a wide range of technologies – from I/O backplanes to wide-area networks – as candidates for both interconnects.

roles. We propose that there is a natural division of hardware resources into the three roles of endpoint, pipeline-shared, batch-shared. If sufficiently powerful software can accurately classify traffic, then scalable systems can be provisioned with the right amount of hardware to create a balanced system for the target workload. This provisioning is not necessarily that obtained by networking commodity desktop workstations into uniform clusters.

A general architecture is shown in Figure 10. There are three types of storage corresponding to the three roles of I/O traffic. An endpoint (E) server is primarily responsible for serving unique input data and archiving output data. When necessary, it must also provide authentic copies of batch-shared data, although this role must be minimized. As we have mentioned, this service is necessarily the bottleneck of the system; a high-end network storage device may be necessary. To minimize load on the archive, batch-shared data are stored on batch-sharing (B) services in the network. These provide access to the read-only data common to all pipeline executions. Finally, each CPU is equipped with a device for temporary storage of pipeline-shared (P) data. For the application sizes presented here, P may simply be main memory. Such memory is logically distinct from other CPUs and needs no facilities for coherence or other communication without the direction of the CPU. We will consider a range of technologies as candidates for the batch-sharing interconnect (BI) and the endpoint interconnect (EI).

Figure 11 shows application performance and scalability of six candidate configurations. All are composed of 1024 CPUs, connected by an aggressive 1500 MB/s endpoint interconnect, and contain some varying number and capacity of batch interconnects. Two types of CPU are shown: a 2 BIPS CPU, typical for commodity systems in 2002, and a 32 BIPS CPU, predicted as typical in 2008 according to Moore's Law; although a simplistic measure of processor performance, instructions-per-second serves us well in these coarse-grained estimations. Each CPU is coupled with three possible batch-sharing interconnects: 12.5 MB/s, representing Fast Ethernet, 125 MB/s, representing Gigabit Ethernet or a commodity I/O backplane, and 1250 MB/s, representing 10 Gigabit Ethernet or a high-performance I/O backplane. For each application and configuration, we select the number of CPUs per batch interconnect that max-

		CPUs/		% Utiliz.		
Pipeline	**Config**	**BI**	**Speedup**	**CPU**	**BI**	**EI**
setiathome	CPU:	1024	1.00	100	1	0
blast	2 BIPS	1	1.00	23	99	0
ibis	BI:	1024	1.00	99	86	0
cms	12 MB/s	40	1.00	100	99	11
hf		1024	1.00	100	92	0
nautilus		666	1.00	100	99	1
amanda		7	1.00	100	94	1
setiathome	CPU:	1024	1.00	100	0	0
blast	2 BIPS	2	4.25	100	84	1
ibis	BI:	1024	1.00	99	8	0
cms	125 MB/s	403	1.00	100	99	11
hf		1024	1.00	100	9	0
nautilus		1024	1.00	100	15	1
amanda		74	1.00	100	99	1
setiathome	CPU:	1024	1.00	100	0	0
blast	2 BIPS	23	4.25	100	97	1
ibis	BI:	1024	1.00	99	0	0
cms	1250 MB/s	1024	1.00	100	25	11
hf		1024	1.00	100	0	0
nautilus		1024	1.00	100	1	1
amanda		742	1.00	100	99	1
setiathome	CPU:	1024	16.00	100	25	0
blast	32 BIPS	1	1.00	1	99	0
ibis	BI:	74	16.00	99	99	10
cms	12 MB/s	1	7.70	48	19	91
hf		69	16.00	100	99	12
nautilus		41	16.00	100	98	16
amanda		1	6.91	43	92	8
setiathome	CPU:	1024	16.00	100	2	0
blast	32 BIPS	1	9.72	14	96	3
ibis	BI:	743	16.00	99	99	10
cms	125 MB/s	1	8.32	52	2	99
hf		692	16.00	100	99	12
nautilus		416	16.00	100	99	16
amanda		3	16.00	100	64	18
setiathome	CPU:	1024	16.00	100	0	0
blast	32 BIPS	1	67.96	100	67	22
ibis	BI:	1024	16.00	99	13	10
cms	1250 MB/s	1	8.39	52	0	99
hf		1024	16.00	100	14	12
nautilus		1024	16.00	100	24	16
amanda		38	16.00	100	81	18

Figure 11. Configurations

imizes system throughput measured in jobs completed per second. This throughput is then normalized against that of the slowest system, yielding a speedup ratio. The final three columns of Figure 11 also show the percent utilization of each resource, indicating which is the bottleneck.

The first configuration is approximately the state of commodity computing clusters in 2002. Given an appropriate cluster size this technology may be made well-balanced for four of seven applications. It is far over-provisioned for SETI, and underprovisioned for BLAST. Moving down the table, upgrading the BI to 125 MB/s offers BLAST a speedup of 4.25 times, even with only two CPUs per BI. Other applications see no benefit. Of these applications, BLAST may be the only one well-suited to commodity SMPs with small numbers of processors.

The fourth configuration, using 32 BIPS processors and a 12.5 MB/s batch interconnect, might be the state of commodity computing clusters in 2008. Here, with smaller but still reasonable cluster sizes, IBIS, HF, and Nautilus may still be made well-balanced. AMANDA is underprovisioned, but may be satisfied by advancing to faster networks. A single instance of BLAST will still consume the majority of a 1250 MB/s network, so distributing this computation will require I/O development to match CPU growth. However, CMS is in deeper trouble. It is not constrained by the local area network, but with 1024 processors, its global throughput is limited by the bandwidth of even a 1500 MB/s storage device.

Clearly, no single configuration will satisfy all applications. Any general-purpose system will be drastically over- or under- provisioned for a given application. However, careful allocation of resources to different classes of jobs may still yield a balanced system. For example, 1024 2 BIPS CPUs connected by a 125 MB/s network could easily be shared by two instances of BLAST and 1022 instances of Nautilus, utilizing the CPU and BI and 100 and 99 percent, respectively. If such systems are to be used as commodity engines for scientific work, then network resources must rise to become an allocable resource in combination with CPU time and disk space.

6.3 Software Architecture

These workloads, whether executed on conventional or custom architectures, have unusual demands of system software. They require something like a distributed file system, that provides access to shared input data and output space via a consistent naming scheme within constraints of security, persistence, and performance. Traditional file systems do not serve these applications because their naming and consistency requirements are targeted to interactive cooperating users. These applications require a data management system that has specialized requirements for workload analysis, failure recovery, and resource management.

Scalable solutions to both pipeline and batch sharing problems require that an application's I/O be classified into each of the three roles with some degree of accuracy. Custom applications such as SETI have succeeded in wide scalability by virtue of manual I/O division: all endpoint I/O happens via explicit network communication. Yet, we can hardly expect that all valuable applications will be rewritten for a distributed environment. Ideally, such I/O roles would be detected automatically by the system, but we might reasonably ask the user to provide hints of I/O roles to the system without modifying applications directly.

A number of file systems take account of the conventional wisdom that quickly-deleted data is a significant source of traffic in general-purpose workload. However, this recognition has limited application due to the requirements of reliability and consistency in interactive systems. For example, NFS permits a 30-60 second delay between application writes and data movement to the server. Were this delay made to be minutes or hours in order to accommodate pipeline sharing, the reduction in unnecessary writes would be accompanied by a much increased danger of data loss during a crash and some very unusual consistency semantics. The session semantics of AFS are even worse: closing a file is a blocking operation that forces the write-back of dirty data. Not only would all vertically shared data be written back at each of the (numerous) close operations, but the CPU would be held idle between pipelines, offering no possibility of CPU-I/O overlap.

General-purpose file systems operate under the assumption that most data must eventually flow back to the archival site. These workloads require the opposite assumption: most created data should remain *where it is created* until an explicit operation by the writer, the system, or perhaps the user forces it into archival storage. This improves overlap and eliminates unnecessary writes, but runs the danger that I/O operations waiting to be written back may fail, due to permissions, disconnection, or any of the many other sources of error in a distributed file system. This is acceptable in a scientific batch computing system, as long as such a failed I/O can be detected, matched with the process that issued it, and force a re-execution of the job. Such semantics would not be appropriate in an interactive workload, where a user expects that I/O completion is immediate.

Input sharing has been accomplished in traditional file systems through the use of client-side caches. Such caches have improved both scalability and performance by sharing recently-read data between processes However, cache misses are still resolved at the central server. This will not scale when many thousands of compute nodes begin executing the same program at once and all request the same input files to fill cold caches. To achieve scalability in batch sharing, a compute node must attempt to leverage the potentially warm caches of its peers. Such techniques will seek to maximize scalability, even at the price of reduced single-client performance.

7. Related Work

The CPU, memory, communication, and I/O characteristics of applications have been studied for many years by the research community. These can be roughly categorized by the type of workloads that they consider: general-purpose workloads containing many applications, sequential applications examined in isolation, or parallel applications in isolation. We summarize the work in each of these categories, focusing on those that have examined file system activity.

File system activity has been examined for a range of general-purpose workloads. Many of the studies that have greatly influenced file system design over the last 20 years focused on academic and research workloads [31, 24, 5, 29]. These studies have found that most files have very short lifetimes, access patterns exhibit a high degree of locality, and read-write sharing is rare. However, missing from these broad studies of traffic is any linkage to the applications that generate the traffic.

More similar to our work are those studies that have focused on the behavior of individual applications. The memory and I/O behavior of sequential applications received great attention during the early development of virtual-memory and file cache mechanisms and policies in traditional operating systems [22]. For example, Denning's working set model [10], initially applied to memory-access patterns, has also been examined as a model for sequential I/O behavior [19]. As with file systems, studies of commercial workloads [6, 18] have become more common in recent years. However, in this domain, the interaction or pipeline behavior of sequential applications has not been examined. While we believe it may also be interesting to study the detailed memory-system behavior of our applications, we do not believe the opportunities for sharing are fundamentally different than other studies.

Parallel applications are in many ways the most similar to pipelined batch applications. The CPU, memory, communication, and I/O behavior of parallel and vector applications have been quantified in a number of studies [9, 37, 36], but the most relevant studies consider the impact of explicit I/O [30, 32, 23, 8, 1]. Our study embellishes these works by studying the sharing behavior of an important new class of workload.

Many of these studies demonstrate the drastic differences in I/O behavior for parallel applications compared to general-purpose workloads. For example, parallel scientific workloads often have high, bursty I/O rates [21, 25] and relatively constant behavior across different runs and input parameters [25]; further, parallel workloads tend to be dominated by the storage and retrieval costs of large files, particularly check-point files[21, 16]; finally, quick deletion is uncommon [16].

As many have observed, improvements in processor and memory speed have far outstripped improvements in I/O performance. Models of I/O system behavior [2, 26] have relied on general rules such as Amdahl's law to guide system design. A more detailed measurement of the interaction between I/O-intensive processes will serve to guide future models of systems and workloads.

8. Conclusions

Applications are not run in isolation. In production settings, scripting and workflow tools are used to glue together series of applications into pipelines; a particular pipeline may be run many thousands of times over varied inputs to achieve the goals of the users. We term such workloads batch-pipelined, as batches of pipelines are run at a given instant.

In this paper, we characterize an important class of batch-pipelined workloads, within the realm of computational science. Beyond typical characterizations of processing, memory, and I/O demands, we bring forth the sharing characteristics of the workloads, and demonstrate their importance on scalability.

We also study the architectural impacts of this workload class, from both a hardware and software perspective. The key to scalability of these workloads is I/O classification; by segregating I/O traffic by type, and aggressively exploiting sharing characteristics, scalability can be improved by many orders of magnitude.

As shared computing infrastructures and application outsourcing become commonplace, the ability of scalable systems to handle the demands of batch-pipelined workloads will become increasingly important. Studying the interaction of many such batch-pipelined workloads on a single platform is thus an interesting topic for future research.

9 Acknowledgements

We gratefully acknowledge all of the people who helped us to install, operate, and understand these applications and their behavior, including Paolo Desiati, Zach Miller, Dierk Polzin, Daniel Reed, and Alan de Smet. Douglas Thain is supported by a Lawrence Landweber NCR Fellowship in Distributed Systems and the Wisconsin Alumni Research Foundation.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. K. Hollingsworth, J. Saltz, and A. Sussman.

Tuning the performance of I/O intensive parallel applications. In *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems (IOPADS)*, pages 15–27, Philadelphia, Pennsylvania, 1996.

[2] J. Akella and D. P. Siewiorek. Modeling and measurement of the impact of input/output on system performance. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 390–399, 1991.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, , and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. In *Nucleic Acids Research*, pages 3389–3402, 1997.

[4] G. Amdahl. Storage and I/O Parameters and System Potential. In *IEEE Computer Group Conference*, pages 371–72, June 1970.

[5] M. G. Baker, J. H. Hartmann, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP)*, July 1991.

[6] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 3–14, 1998.

[7] J. F. Cantin and M. D. Hill. Cache Performance for Selected SPEC CPU2000 Benchmarks. Computer Architecture News (CAN), September 2001.

[8] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed. Input/output characteristics of scalable parallel applications. In *Proceedings of the IEEE/ACM Conference on Supercomputing*, San Diego, California, 1995.

[9] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural Requirements of Parallel Scientific Applications with Explicit Communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, San Diego, California, May 17–19, 1993. ACM SIGARCH and IEEE Computer Society TCCA. *Computer Architecture News,* 21(2), May 1993.

[10] P. J. Denning. The working set model for program behavior. In *Proceedings of the ACM symposium on Operating System Principles (SOSP)*, 1967.

[11] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[12] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, , and A. J. Smith. Cache Performance of the SPEC92 Benchmark Suite. *IEEE Micro*, August 1993.

[13] K. Holtman. CMS data grid system overview and requirements. CMS Note 2001/037, CERN, July 2001.

[14] P. Hulith. The AMANDA experiment. In *Proceedings of the XVII International Conference on Neutrino Physics and Astrophysics*, Helsinki, Finland, June 1996.

[15] e. a. J.A. Foley. An integrated biosphere model of land surface processes, terrestrial carbon balance, and vegetation dynamics. *Global Biogeochemical Cycles*, 10(4):603–628, 1996.

[16] S. Kuo, M. Winslett, Y. Cho, J. Lee, and Y.Chen. Efficient input and output for scientific simulations. In *Proceedings of I/O in Parallel and Distributed Systems (IOPADS)*, pages 33–44, 1999.

[17] T. L. Lancaster. The Renderman Web Site. http://www.renderman.org/, 2002.

[18] D. C. Lee, P. J. Crowley, J.-L. Bear, T. E. Anderson, and B. N. Bershad. Execution characteristics of desktop applications on windows NT. In *ISCA '98*, pages 27–38, 1998.

[19] S. Majumdar and R. B. Bunt. Measurement and analysis of locality phases in file referencing behavior. In *SIGMETRICS '86*, pages 180–192, May 1986.

[20] A. M. G. Maynard, C. Donnelly, and B. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 1994.

[21] E. L. Miller and R. H. Katz. Input/output behavior of supercomputing applications. In *Proceedings of the ACM/IEEE conference on Supercomputing*, pages 567–576, 1991.

[22] J. M. Murphy and R. B. Bunt. Characterising program behavior with phases and transitions. In *Proceedings of the International Conference on Measurement and Modelling of Computer Systems (SIGMETRICS)*, pages 226–234, May 1988.

[23] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, 1996.

[24] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Procedings of the 10th ACM Symposium on Operating Systems Principles (SOSP)*, pages 15–24, December 1985.

[25] B. K. Pasquale and G. C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of the ACM/IEEE conference on Supercomputing*, pages 388–397, November 1993.

[26] A. L. N. Reddy. A study of I/O system organizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 308–317, 1992.

[27] A. L. N. Reddy and P. Banerjee. A study of I/O behavior of perfect benchmarks on a multiprocessor. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 312–321, 1990.

[28] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and typhoon: User-level shared memory. In *Proc. of the 21th Annual Intl. Symp. on Computer Architecture (ISCA)*, pages 325–337, 1994.

[29] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *USENIX Annual Technical Conference*, 2000.

[30] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante. The impact of I/O on program behavior and parallel scheduling. In *Proceedings of the Joint International Conference on Measurement and Modelling of Computer Systems (SIGMETRICS)*, pages 56–65, 1998.

[31] M. Satyanarayanan. A study of file sizes and functional lifetimes. In *Proceedings of the 8th Symposium on Operating Systems Principles (SOSP)*, pages 96–108, 1981.

[32] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed. I/O requirements of scientific applications: An evolutionary view. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applica-*

tions, pages 576–594. IEEE Computer Society Press, New York, NY, 2001.

[33] S. Soderbergh. Mac, Lies, and Videotape. www.apple.com/hotnews/articles/2002/04/fullfrontal/, 2002.

[34] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proceedings of the 5th International Conference on Bioastronomy*, 1997.

[35] A. K. Sum and J. J. de Pablo. Nautilus: Molecular Simulations code. Technical report, University of Wisconsin - Madison, Dept. of Chemical Engineering, 2002.

[36] F. Wong, R. P. Martin, R. H. Arpaci-Dusseau, D. Wu, and D. E. Culler. Architectural Requirements and Scalability of the NAS Parallel Benchmarks. In *Supercomputing '99*, Portland, Oregon, Nov. 1999.

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Shingh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, June 22–24, 1995. ACM SIGARCH and IEEE Computer Society TCCA. *Computer Architecture News,* 23(2), May 1994.