Pipeline and Batch Sharing in Grid Workloads

Douglas Thain, John Bent, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny
Computer Sciences Department, University of Wisconsin, Madison

Abstract

We present a study of six batch-pipelined scientific workloads that are candidates for execution on computational grids. Whereas other studies focus on the behavior of single applications, this study characterizes workloads composed 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. We conclude with a discussion of the ramifications of these workloads for end-to-end scalability and overall system design.

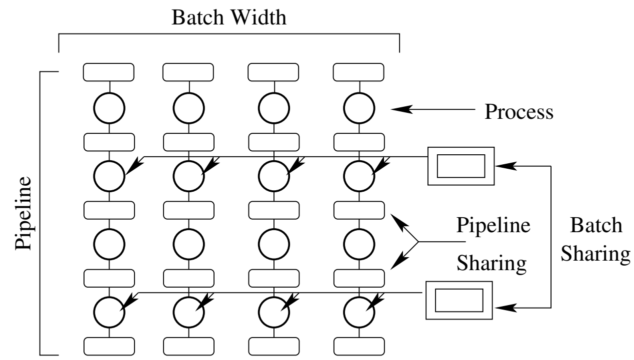


Figure 1. A Batch-Pipelined Workload

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 [5, 15, 17]. 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 [6] , and the communication characteristics of parallel applications has similarly been well documented [8, 33, 32].

However, applications are not used in isolation in production settings. Particularly in computational science, 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. Such applications are executed in a high throughput computing system such as Condor [16] and may be managed by high-level workflow software such as Chimera [10].

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 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 design. We find that wide-area network bandwidth is a serious scalability problem for these applications, unless attempts are made to eliminate shared I/O. Successful systems for these workloads must segregate the three types of I/O traffic in order to be able to scale successfully. We submit that pipeline-shared data is at least as significant a problem as batch-shared data, and elucidate why traditional file systems are not appropriate for these workloads.

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 Sections 4 and 5, we analyze the data and discuss the implications. We discuss related work in Section 6, and conclude in Section 7.

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 [26] as a point of reference. With guidance from users, we chose workloads and input parameters to correspond to production use. Descriptions of the applications are found in Figure 2.

Some of the applications have a variable granularity. Both CMS and AMANDA process a variable number of small, independently generated events. For these applications, we chose pipeline sizes of 250 events (CMS) and 100,000 showers (AMANDA), corresponding to typical production use. In both cases, the CPU and I/O resources consumed by a pipeline scale linearly with the number of events. IBIS has multiple datasets of differing resolutions in which the granularity of the resolution reflects the size of the dataset. In these experiments, we used a medium sized dataset. IBIS and Nautilus perform single simulations of variable length, while SETI, BLAST, and HF operate on a work unit of fixed size.

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

A diamond-shaped 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. For example, analysis of Condor logs shows that the usual batch size is over a thousand for AMANDA, CMS and BLAST. This property can be exploited for efficient wide-area distribution over modest communication links.

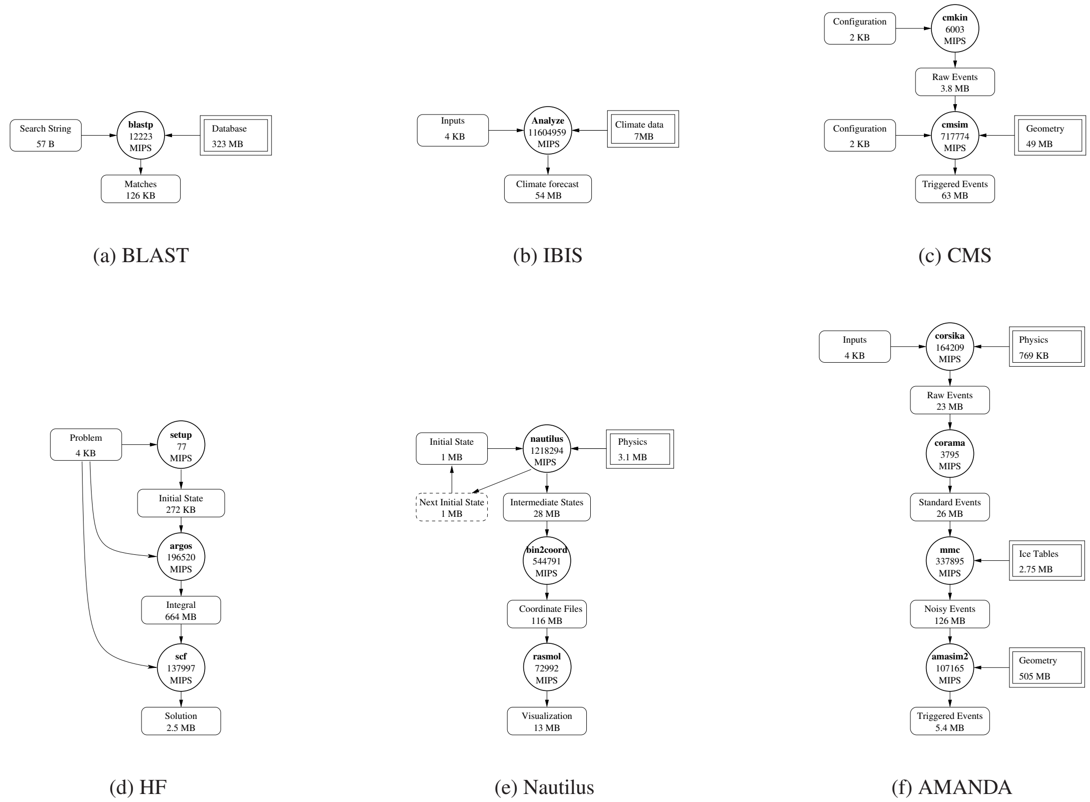
3. Method

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 make use of a shared-library interposition agent [29] 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. 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. Memory requirements and program sizes are all quite modest in comparison to total I/O volume.



(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 instruction counts. Rounded boxes indicate data private to a pipeline. Double boxes indicate data shared between pipelines in a batch. Arrows indicate data flow.*

· **BLAST** *[2] 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 matches.*

· **IBIS** *[9] 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** *[12] 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.*

· **Messkit Hartree-Fock (HF)** *[7] 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.*

· **Nautilus** *[27] 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`*.*

· **AMANDA** *[13] 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			Memory (MB)			I/O Traffic		
			Integer	Float	Burst	Text	Data	Share	MB	Ops	MB/s
seti	seti	41587.1	1953084.8	1523932.2	4.6	0.1	15.7	1.1	75.8	417260	0.00
blast	blastp	264.2	12223.5	0.2	0.1	2.9	323.8	2.0	330.1	88671	1.25
ibis	ibis	88024.3	7215213.8	4389746.8	104.7	0.7	24.0	1.4	336.1	110802	0.00
cms	cmkin	55.4	5260.4	743.8	6.1	19.4	5.0	2.6	7.5	988	0.14
	cmsim	15595.0	492995.8	225679.6	0.4	8.7	70.4	4.3	3798.7	1915559	0.24
	total	15650.4	498256.1	226423.4	0.4	19.4	70.4	4.3	3806.2	1916546	0.24
hf	setup	0.2	76.6	0.4	0.0	0.5	4.0	1.3	9.1	2953	56.43
	argos	597.6	179766.5	26760.7	0.8	0.9	2.5	1.4	663.8	254713	1.11
	scf	19.8	132670.1	5327.6	0.2	0.5	10.3	1.3	3983.4	765562	201.06
	total	617.6	312513.2	32088.6	0.3	0.9	10.3	1.4	4656.3	1023228	7.54
nautilus	nautilus	14047.6	767099.3	451195.0	18.6	0.3	146.6	1.2	270.6	65523	0.02
	bin2coord	395.9	263954.4	280837.2	4.2	0.0	2.2	1.4	403.3	129727	1.02
	rasmol	158.6	69612.8	3380.0	1.9	0.4	4.9	1.7	128.7	38431	0.81
	total	14602.2	1100666.5	735412.2	7.9	0.4	146.6	1.7	802.7	233681	0.05
amanda	corsika	2187.5	160066.5	4203.6	26.4	2.4	6.8	1.4	24.0	6225	0.01
	corama	41.9	3758.4	37.9	0.3	0.5	3.2	1.1	49.4	12693	1.18
	mmc	954.8	330189.1	7706.5	0.3	0.4	22.0	4.9	154.4	1141633	0.16
	amasim2	3601.7	84783.8	20382.7	143.7	22.0	256.6	1.6	550.3	733	0.15
	total	6785.9	578797.8	32330.7	0.5	22.0	256.6	4.9	778.0	1161275	0.11

Figure 3. Resources Consumed. *Shown here are the total amounts of resources consumed. In this and in subsequent tables, shading is used to differentiate between different application pipelines. Real time refers to the total wall-clock time of each of the applications when run without instrumentation overhead. Burst is the average number of instructions executed between I/O operations. Instruction counts were obtained using the performance monitoring counters (PMCs) available on x86-class processors. Notice that, with the exception of HF, all of the application pipelines have very modest bandwidth requirements.*

Application		Total I/O				Reads				Writes			
		Files	Traffic	Unique	Static	Files	Traffic	Unique	Static	Files	Traffic	Unique	Static
seti	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. *Shown here are the total amounts of I/O performed. Traffic is the number of bytes that flow into and out of the process. Unique I/O considers only unique byte ranges within this total traffic. Notice that HF and CMS both perform large proportions of reread traffic indicating that caching is particularly important for them. Static I/O refers to the total size of all of the files accessed and may be less than unique I/O if applications read only portions of the files. Notice for example that BLAST reads less than 60% of the total data in the files that it accesses. This suggests that systems which prestage data sets may sometimes be performing unnecessary work.*

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. *Shown here are the total number of the different type of I/O instructions executed by each of the applications. The* `Seek` *column includes non-sequential access to memory-mapped pages and ignores all* `lseek` *operations which do not actually change the file offset. 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. Notice that many of the applications have high degrees of random access as shown by the ratio of* `seeks` *to* `reads` *and* `writes`*. This contradicts previous file system studies which indicate the dominance of sequential I/O [4].*

	Endpoint I/O (MB)				Pipeline I/O (MB)				Batch I/O (MB)			
Appl.	Files	Traffic	Unique	Static	Files	Traffic	Unique	Static	Files	Traffic	Unique	Static
seti	2	0.34	0.34	0.34	12	75.43	2.68	2.68	0	0.00	0.00	0.00
blastp	2	0.12	0.12	0.12	0	0.00	0.00	0.00	9	329.99	323.46	586.09
ibis	20	179.92	53.97	53.97	99	148.27	12.69	12.69	17	7.89	6.98	6.98
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
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	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
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. *Shown here are the total amounts of each type of I/O performed. Endpoint traffic consists of the initial inputs and final outputs that are unique to each application. Pipeline traffic is intermediate data passed between pipeline stages or even intermediate data passed between different phases of a single stage. Batch traffic is input data that are shared across different instances of the pipeline. Traffic is the number of bytes that flow into and out of the process. Unique I/O considers only unique byte ranges within this total traffic. Static I/O refers to the total size of all of the files accessed and may be less than unique I/O if applications read only portions of the files. Notice that all of the applications, with the exception of IBIS, have very little endpoint traffic relative to their total traffic. This indicates that the scalability of systems that run these applications will depend on their ability to differentiate between these different types of I/O.*

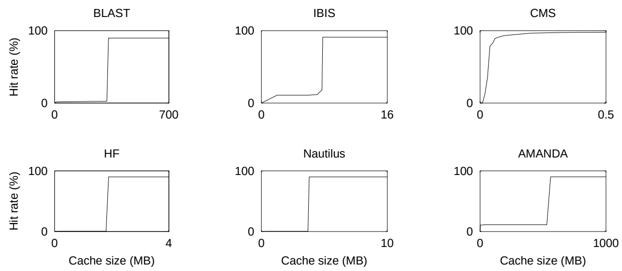


Figure 7. Batch Cache Simulation

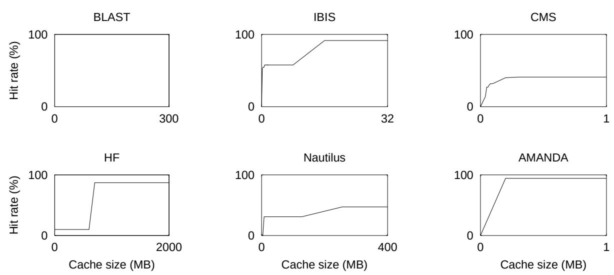


Figure 8. Pipeline Cache Simulation

Figure 4 details the I/O volume produced by each pipeline stage. 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. Overwriting of output data is also found in all pipelines with the exception of AMANDA. Output over-writing 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 by renaming it.) 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 operations is given in Figure 5. Notice that many of these applications have a high degree of random access, as shown by the ratio of `seeks` to `reads` and `writes` 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 [4].

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 or even intermediate data passed between different phases of a single stage. *Batch* traffic is 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.

Examining both Figures 5 and 6, we note that 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.

Figures 7 and 8 show the working set sizes of batch-shared and pipeline-shared data in each workload. These values are computed from simulations performed on the trace data with a batch width of 10 and a varying LRU cache size with 4KB blocks. Executable files are implicitly included as batch-shared data. In general, for both types of sharing, the necessary cache sizes are small with respect to both the I/O volume and the sizes of typical main memories today. There are some outliers. AMANDA has a large amount of batch shared data (over half a GB) that is read only once, and thus a cache is not effective until very large sizes. However, AMANDA also has a very high pipeline hit rate at small cache sizes due to a large number of single-byte I/O requests. Due to its high degree of re-reading and output overwriting, CMS needs only very small cache sizes to effectively maximize its hit rates. BLAST has no pipeline data. IBIS, though one stage, has pipeline data in the form of checkpoints written and read multiple times.

Figure 9 shows how these applications relate to Amdahl's long standing system balance ratios [3], recently amended by Gray [11]. These workloads have CPU-IO ratios (measured in MIPS/MBPS) far exceeding Amdahl's ideal value of eight, indicating reliance on computation rather than I/O. The ratio of memory to CPU speed, known as alpha, is near or below Amdahl's value of one, and with the exception of the last component of AMANDA, never comes close to Gray's value of four. This also indicates reliance on computation rather than memory. Finally, the ratio of CPU instructions to I/O instructions is several orders of magnitude larger than 50,000. With respect to a single instance of each pipeline, a commodity computing node engineered to Amdahl's metrics is considerably over-provisioned with I/O bandwidth and memory capacity.

Appl.	CPU/IO (MIPS/MBPS)	MEM/CPU (MB/MIPS)	CPU/IO (instr/op)
seti	45888	0.15	8737 K
blastp	37	26.77	144 K
ibis	34530	0.20	109823 K
cmkin	801	0.26	6372 K
cmsim	189	1.86	393 K
total	190	2.09	396 K
setup	8	0.06	27 K
argos	311	0.02	850 K
scf	34	0.30	189 K
total	74	0.16	353 K
nautilus	4501	1.71	19496 K
bin2coord	1350	0.00	4403 K
rasmol	566	0.02	1991 K
total	2287	1.20	8238 K
corsika	6854	0.14	27670 K
corama	76	0.06	313 K
mmc	2189	0.10	310 K
amasim2	191	12.48	150443 K
total	785	3.77	551 K
Amdahl	**8**	**1.00**	**50 K**
Gray	**8**	**1 - 4**	**>50 K**

Figure 9. Amdahl's Ratios

5. System 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 provisioning of the necessary resources.

5.1. Endpoint Scalability

Regardless of the capacity of individual computing nodes, the ultimate scalability of these workloads is limited by competition for shared resources. We assume that each workload relies on a central site for the authenticity and archival of input and output data. 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 10 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.

It is valuable to consider the limits of workload scalability as CPU and I/O hardware improve in performance over time. The limits of space prevent us from doing so here, but a detailed discussion may be found in a technical report. [28]

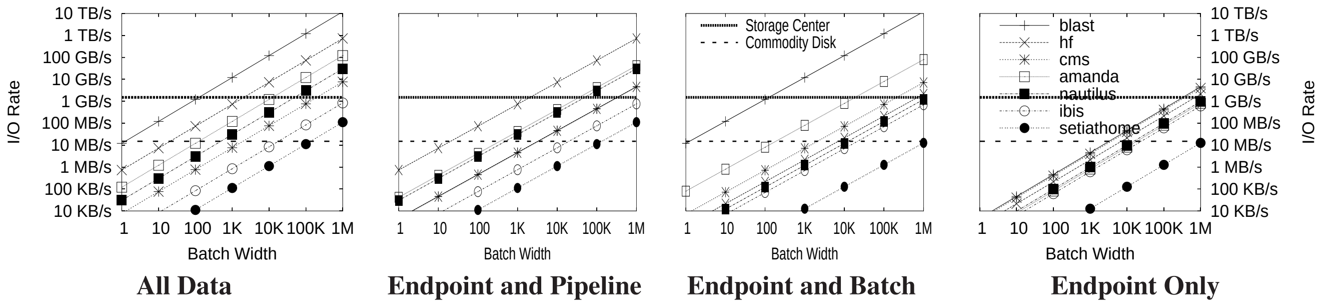


Figure 10. Scalability of I/O Roles. *These graphs show how the scalability of these applications can be improved by up to several orders of magnitude when batch-shared and pipeline-shared I/O are not performed at the endpoint server. The two horizontal lines show milestones in I/O bandwidth. The upper, at 1500 MB/s, represents a high-end storage center and the lower, at 15 MB/s, represents a current commodity disk.*

5.2 Software Architecture

In order to scale to large sizes, software architectures for these workloads must strive to eliminate batch-shared and pipeline-shared data from endpoint interactions wherever possible, within the 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.

By itself, the issue of batch input sharing has received significant attention in the grid computing community. Deployed systems such as SRB [20] and GDMP [24] manage widely-distributed and well-known batch shared data. Techniques for discovering [31] and replicating [21] batch-shared data have been proposed.

Without diminishing the importance of batch sharing, we submit that the issue of pipeline sharing is a very different problem that has been relatively neglected. As Figure 10 shows, the localization of both types of I/O is necessary to achieve high scalability. The treatment of pipeline-shared data must necessarily be different than that of batch shared data, because it will have one writer and few (or one) reader before it is discarded. Pipeline-shared outputs will require some facility for discovery by the reader of the data, but need not be advertised to the same degree as batch-shared data. The loss of a pipeline-shared output may require the re-execution of a previous computation stage.

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 re-written for a distributed environment. Ideally, such I/O roles would be detected automatically. Such an approach is taken by the TREC [30] system, which deduces program dependencies from I/O behavior. We might also 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 increases 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 batch 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.

We suggest that this problem can be attacked by a coupling with a workflow manager, such as Condor's DAGMan or Globus' Chimera [10], which tracks the dependencies in general graphs of jobs. In both of these systems, I/O activity is presumed to be a reliable (and centralized) side effect of execution. However, if the creation and positioning of pipeline-shared data were integrated into the workflow, such data could be efficiently shared while still maintaining the possibility of error recovery.

6. 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 [25, 18, 4, 22]. 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 in commercial workloads [5, 15]. 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 in 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 [8, 33, 32]; a few of which consider the impact of explicit I/O [23, 7, 1]. Our study complements 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 [17] and relatively constant behavior across different runs and input parameters [19]; further, parallel workloads tend to be dominated by the storage and retrieval costs of large files, particularly check-point files [17]; finally, quick deletion is uncommon [14].

7. 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 a collection of scientific batch-pipelined workloads. Beyond typical characterizations of processing, memory, and I/O demands, we bring forth the sharing characteristics of the workloads, and demonstrate their importance to scalability. The key to managing 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.

8 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, Amadeu Sum, Juan de Pablo, Jon Foley, Dierk Polzin, Daniel Reed, and Alan de Smet. We would like to also thank Brian Forney, Muthian Sivathanu, Florentina Popovici and Timothy Denehy for helpful discussion and comments on the paper.

This work is sponsored by NSF CCR-0092840, CCR-0133456, CCR-0098274, NGS-0103670, ITR-0086044, an IBM Faculty Award, and the Wisconsin Alumni Research Foundation. Douglas Thain is supported by a Lawrence Landweber NCR Fellowship in Distributed Systems.

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] 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.

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] I. Foster, J. Voeckler, M. Wilde, and Y. Zhou. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.

[11] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proceedings of the Sixteenth IEEE International Conference on Data Engineering (ICDE)*, pages 3–12, 2000.

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

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

[14] 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.

[15] 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 *Proceedings of the 25th International Symposium on Computer Architecture (ISCA)*, pages 27–38, 1998.

[16] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[17] 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.

[18] 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.

[19] 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.

[20] A. Rajasekar, M. Wan, and R. Moore. MySRB and SRB - components of a data grid. In *In Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, July 2002.

[21] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, July 2002.

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

[23] 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.

[24] A. Samar and H. Stockinger. Grid Data Management Pilot. In Proceedings of *IASTED International Conference on Applied Informatics (AI2001)*, February 2001.

[25] 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.

[26] 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.

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

[28] D. Thain, J. Bent, R. Arpaci-Dusseau, A. Arpaci-Dusseau, and M. Livny. The architectural implications of pipeline and batch sharing in scientific workloads. Technical Report CS-TR-1463, University of Wisconsin, Computer Sciences Department, January 2003.

[29] D. Thain and M. Livny. Multiple bypass: Interposition agents for distributed computing. *Journal of Cluster Computing*, 4:39–47, 2001.

[30] A. Vahdat and T. Anderson. Transparent result caching. Technical Report CSD-97-974, Computer Science Division, University of California-Berkeley, 1997.

[31] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, May 2001.

[32] 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.

[33] 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.