Understanding and Improving Computational Science Storage Access through Continuous Characterization

Phil Carns
Kevin Harms
Bill Allcock
Charles Bacon
Sam Lang
Rob Latham
Rob Ross

Mathematics and Computer Science Division
Argonne Leadership Computing Facility
Argonne National Laboratory

Motivation

- Leadership computing systems are used for a variety of scientific applications
- Understanding production I/O behavior on these systems is important for several reasons

- For scientists:
  - Is my application performing well?
  - Can I get more bang for my buck?
- For administrators:
  - How are the storage resources being used?
  - What applications and resources need to be tuned?
- For researchers and system planners:
  - What are the trends as applications scale up?
  - How do we design the next system?
  - What research avenues are the most promising?

Our goal: to observe I/O patterns of the majority of applications running on our leadership platform, without perturbing their execution, with enough detail to gain insight and aid in performance debugging.

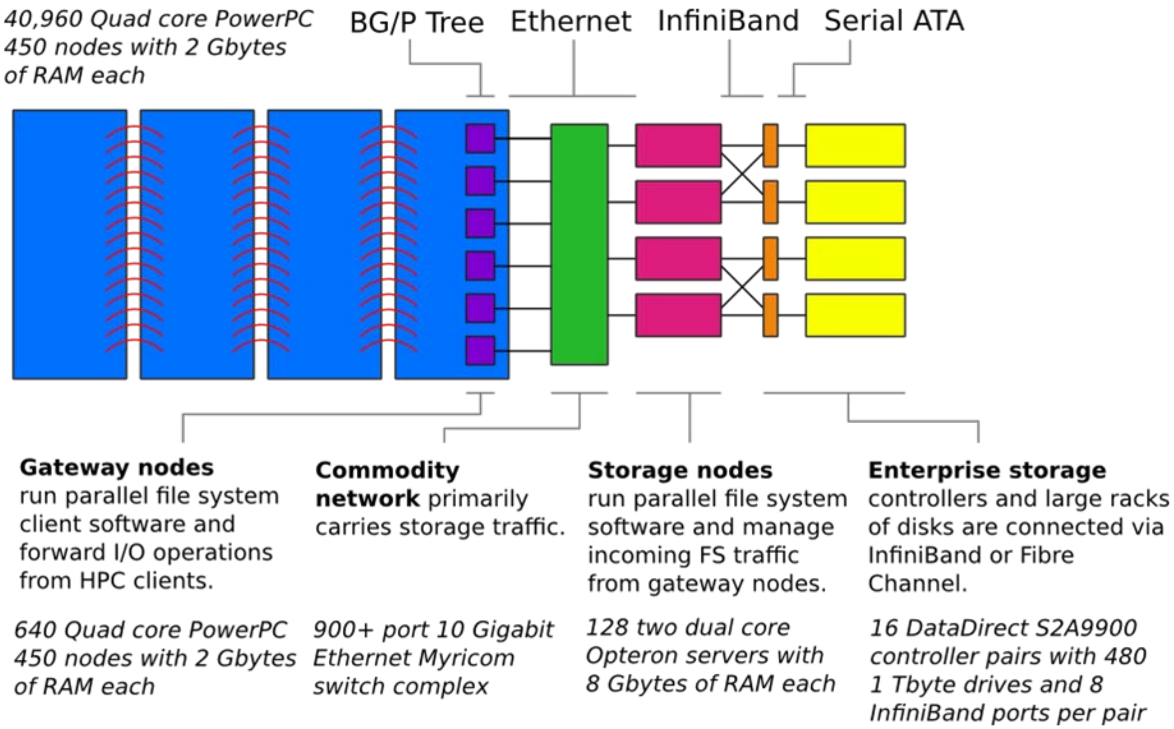
The challenge of collecting data at scale:

How do we observe a leadership storage system in its natural habitat?

Target system:

Intrepid IBM BlueGene/P

Argonne National Laboratory



Gateway nodes run parallel file system client software and forward I/O operations from HPC clients.

640 Quad core PowerPC 450 nodes with 2 Gbytes of RAM each

Commodity network primarily carries storage traffic.

900+ port 10 Gigabit Ethernet Myricom switch complex

Storage nodes run parallel file system software and manage incoming FS traffic from gateway nodes.

128 two dual core Opteron servers with 8 Gbytes of RAM each

Enterprise storage controllers and large racks of disks are connected via InfiniBand or Fibre Channel.

16 DataDirect S2A9900 controller pairs with 480 1 Tbyte drives and 8 InfiniBand ports per pair

- Open science, capability workload
- Sensitive to performance overhead in production
- Too many applications to instrument manually

Technical hurdles

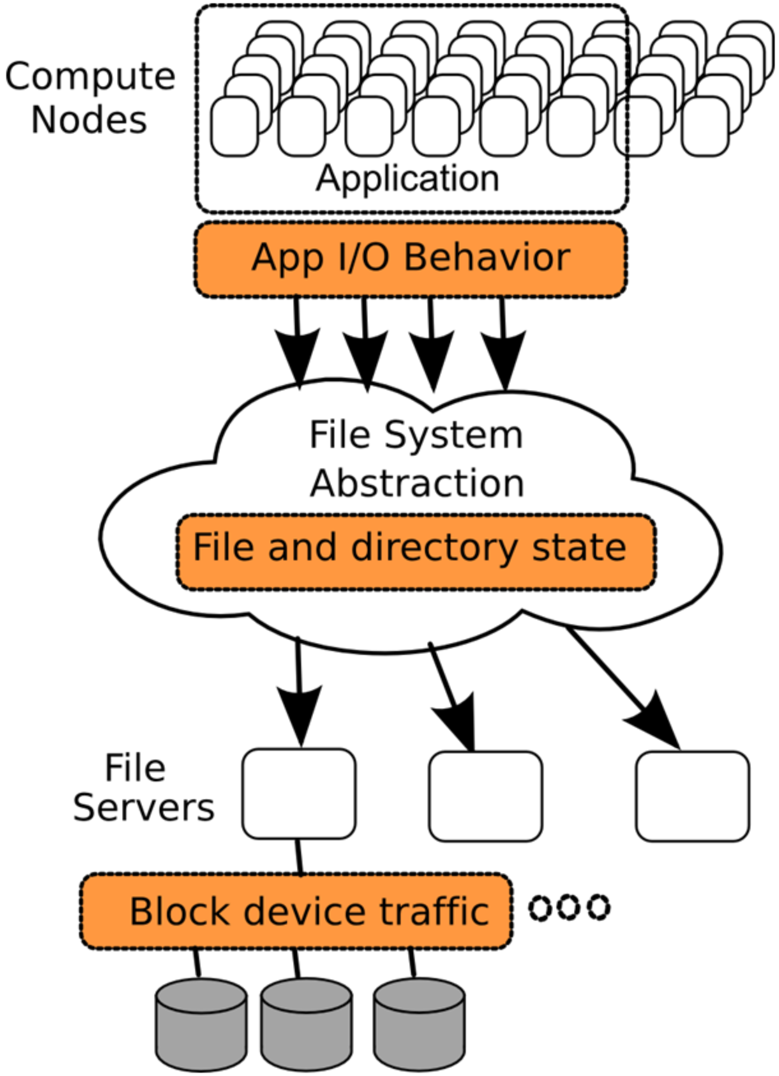
- How do we observe applications?
  - Application or protocol level tracing
    - Performance overhead and volume of data: untenable for system-wide use
  - Benchmarks
    - Isolated examples don't reflect system diversity
  - Statistical sampling
    - May miss critical features

- What APIs or protocols should we instrument?
  - MPI-IO? POSIX? HLL? File System? I/O Forwarding?

- Deluge of information available from multiple system components
  - Inconsistent format
  - What is the overhead?

- How do we correlate application behavior with system activity?

Three views of I/O



Tools used for instrumentation:

- Darshan: instruments and characterizes I/O function calls at the application level (ANL)
- Fsstats: collects static information about aggregate file attributes such as file size and access time (Shobhit Dayal, CMU)
- Iostat: part of the Sysstat tool suite, can be used to report block device utilization statistics (Sebastien Godard)

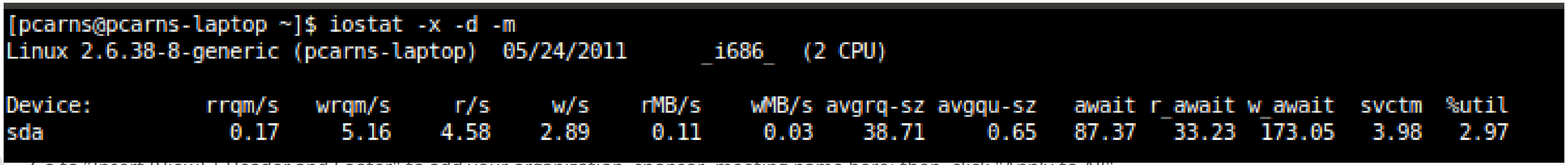
Darshan

- Darshan records counters, histograms, and strategically chosen timestamps related to I/O activity (*not a complete trace of each operation*)
- POSIX, POSIX stream, MPI-IO, and limited HDF5 and PNetCDF functions
- Access patterns, access sizes, I/O time, alignment, datatypes, etc.

- Link-time wrappers inserted via modifications to the default MPI compiler scripts
- Minimal overhead during execution
- Reduction, compression, and storage is performed at MPI_Finalize() time
- "**Application level**" is important: we observe the application's intentions, rather than the system software's interpretation of those intentions
- Inspired by the 1990s Charisma project, Kotz and Nieuwejaar

Fsstats and iostat

- Fsstats:
  - http://www.pdsi-scidac.org/fsstats/
  - Walks a specified directory tree
  - Creates aggregate histograms and usage summaries
  - We developed a small wrapper to run fsstats in parallel across a collection of user directories and merge the results
- Iostat:
  - http://sebastien.godard.pagesperso-orange.fr/
  - Reports data from /proc about utilization on each block device
  - Can be run continuously to report information over regular intervals
  - We developed wrappers to run iostat on each file server and filter results only include GPFS and PVFS block devices. Logs were post-processed to create aggregate summaries.



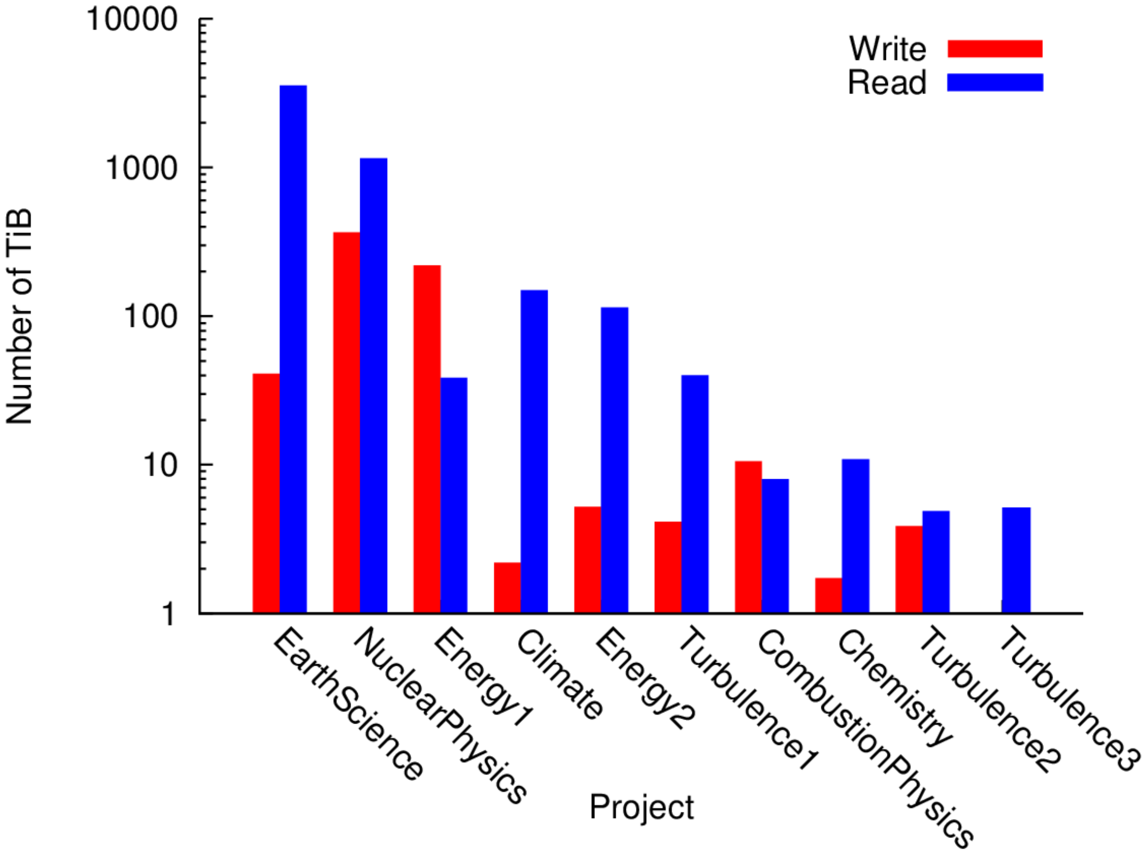
```
[pcarns@pcarns-laptop ~]$ iostat -x -d -m
Linux 2.6.38-8-generic (pcarns-laptop)  05/24/2011      _i686_  (2 CPU)

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.17     5.16    4.58    2.89     0.11     0.03    38.71     0.65   87.37   33.23  173.05   3.98   2.97
```

Studying a production storage system

- In the paper we analyzed a two month window of data collected from January to March of 2010
- This presentation will cover a subset of the findings:
  - What can we learn about trends in I/O patterns and overall system usage?
  - What level of detail can we obtain for specific applications, and how are the most I/O intensive applications performing?
  - How can this data assist in identifying applications with tuning needs?
  - How can this data be used to influence future storage research?
- Application instrumentation:
  - 6,500 jobs (25% of all core hours) were instrumented
  - Examples from 38 distinct science and engineering project allocations
- File system contents (GPFS and PVFS):
  - Roughly 191 million files, sampled at beginning and end of study
- Block device traffic (16 DDN 9900s):
  - Continuous sampling at 60 second intervals, 8 petabytes of total traffic

Which instrumented applications were the most data-intensive?

Amount of data accessed by projects instrumented with Darshan



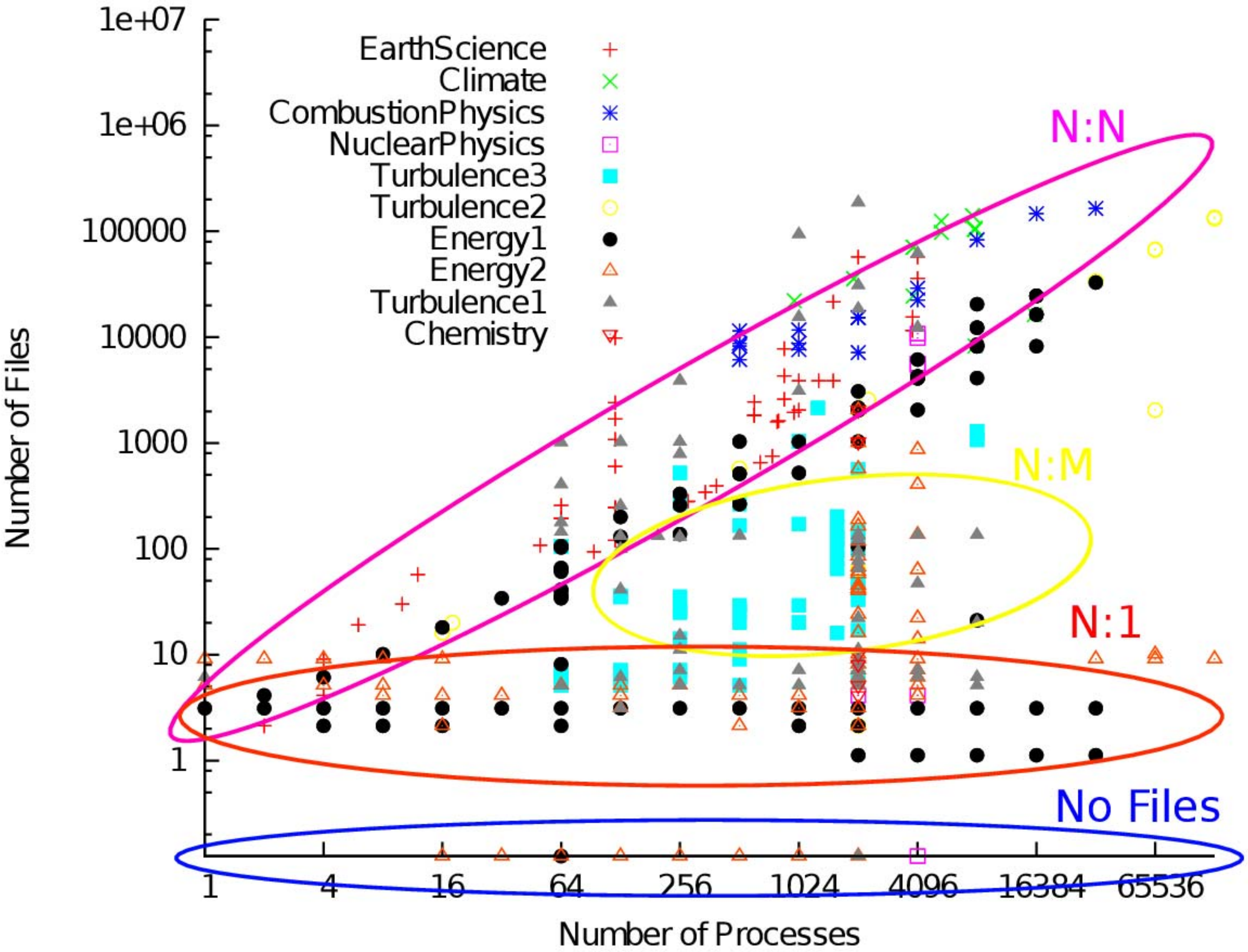
- Contradicts assumptions about write-dominated workloads in HPC
- Data usage varies wildly across scientific domains

What I/O strategies do production applications really use?

- Almost as many different I/O strategies as there are applications….

- APIs:
  - Mostly POSIX or MPI-IO, some use of HDF5 and PNetCDF
- Number of files:
  - N:N, N:1, N:0, N:(N/x) and N:(N*x) mapping of processes to files
  - N:(N/x) includes several examples of manual aggregation: subsets of processes performing I/O on behalf of others without using MPI-IO functionality
- I/O operations:
  - Anywhere from 0.01% to 95% of I/O time is spent performing metadata operations rather than actually moving data
  - Access sizes ranging from 1 byte to 1 GByte

I/O strategy example: Files per process

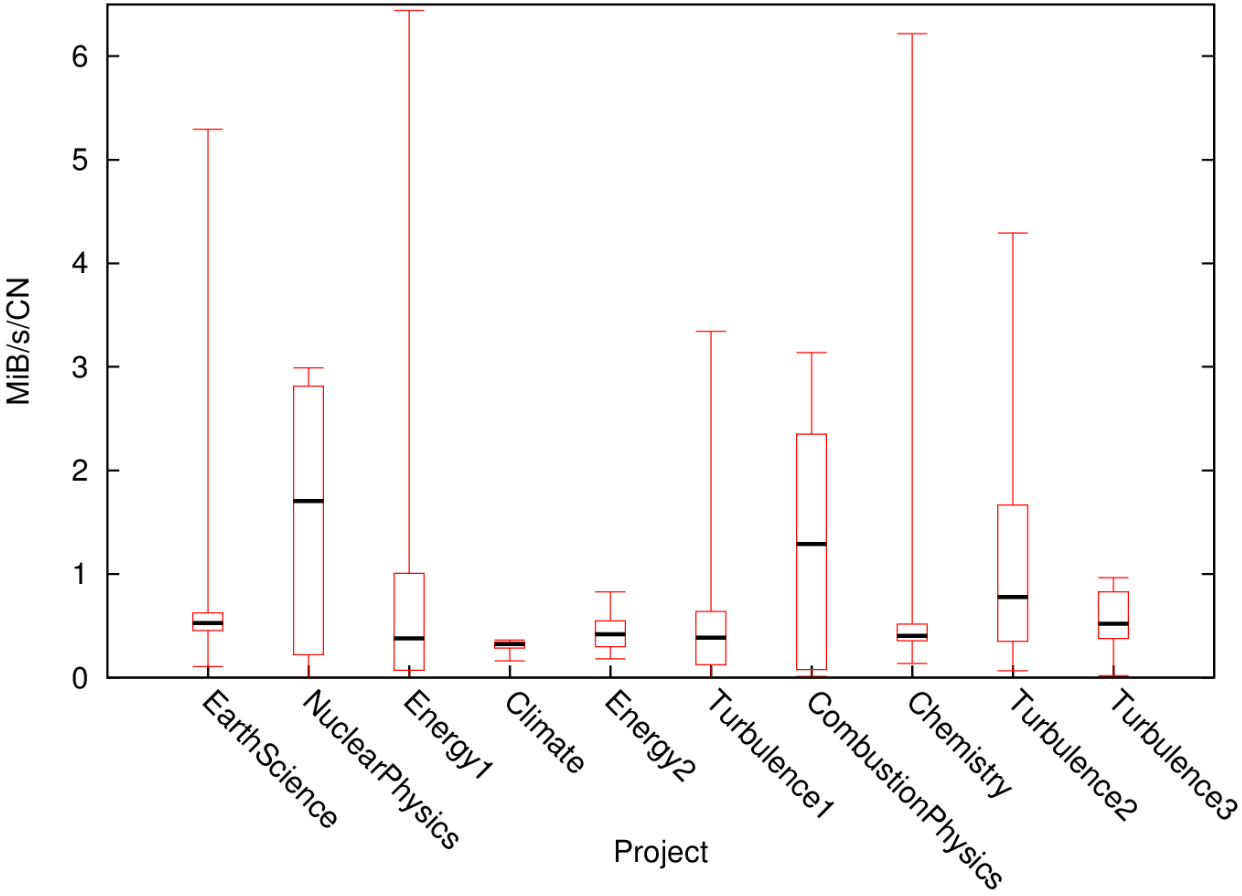


How successful are the various I/O strategies?

- We developed a normalized metric to compare I/O performance across jobs without explicit instrumentation

- The accuracy is limited, but it helps to provide some initial indication of which applications are most interesting

- Data is filtered to only include jobs with 1024 processes or more and at least 512 MiB of data



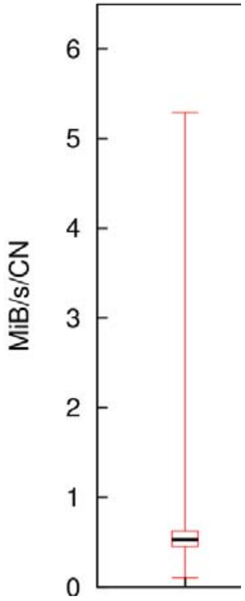
$$MiB/s/CN = \left(\frac{\sum_{rank=0}^{n-1} (bytes_r + bytes_w)}{max_{rank=0}^{n-1} (t_{md} + t_r + t_w)} \right) / N_{cn}$$

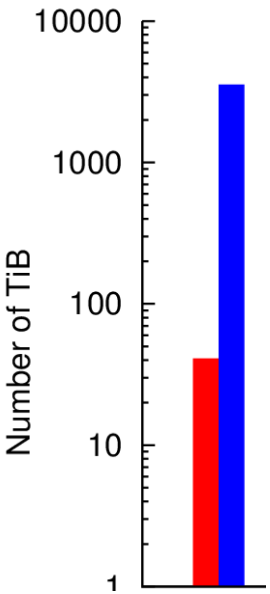
Project case study: EarthScience

- Most jobs used 4096 cores
- Performance is relatively low, despite some positive characteristics
  - relatively large access size, ranging from 100 KiB to 4 MiB

- File usage:
  - Over 1 million files accessed by some script jobs
  - One read or write per file
  - Contributed 88% of new files during study, but only 15% of storage capacity
- 95% of I/O time was spent performing metadata activity

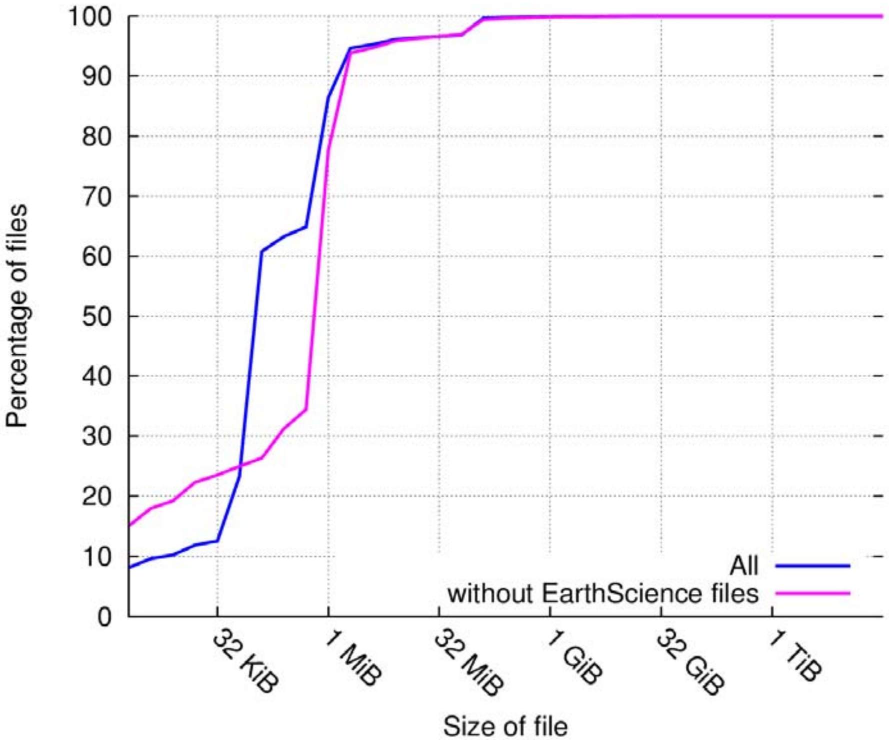
- This is essentially a data-intensive analysis code, should work well but the file organization leads to staggering metadata overhead on Intrepid

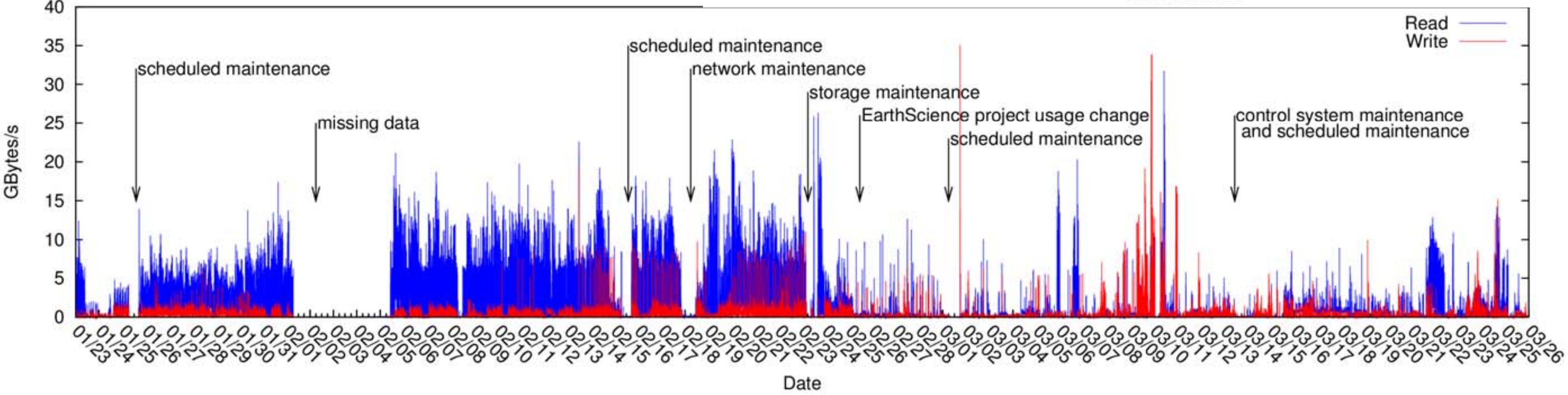




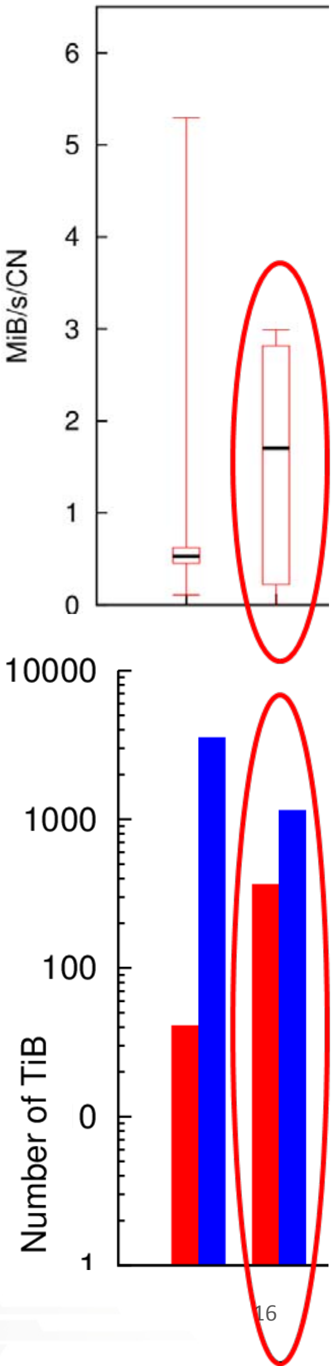
EarthScience impact

- 96 million files (out of 191 million total), mostly under 128 KiB in size, skewed analysis of file sizes
- Reads went from 78% to 50% of all traffic when this project reduced its activity in February





Project case study: NuclearPhysics

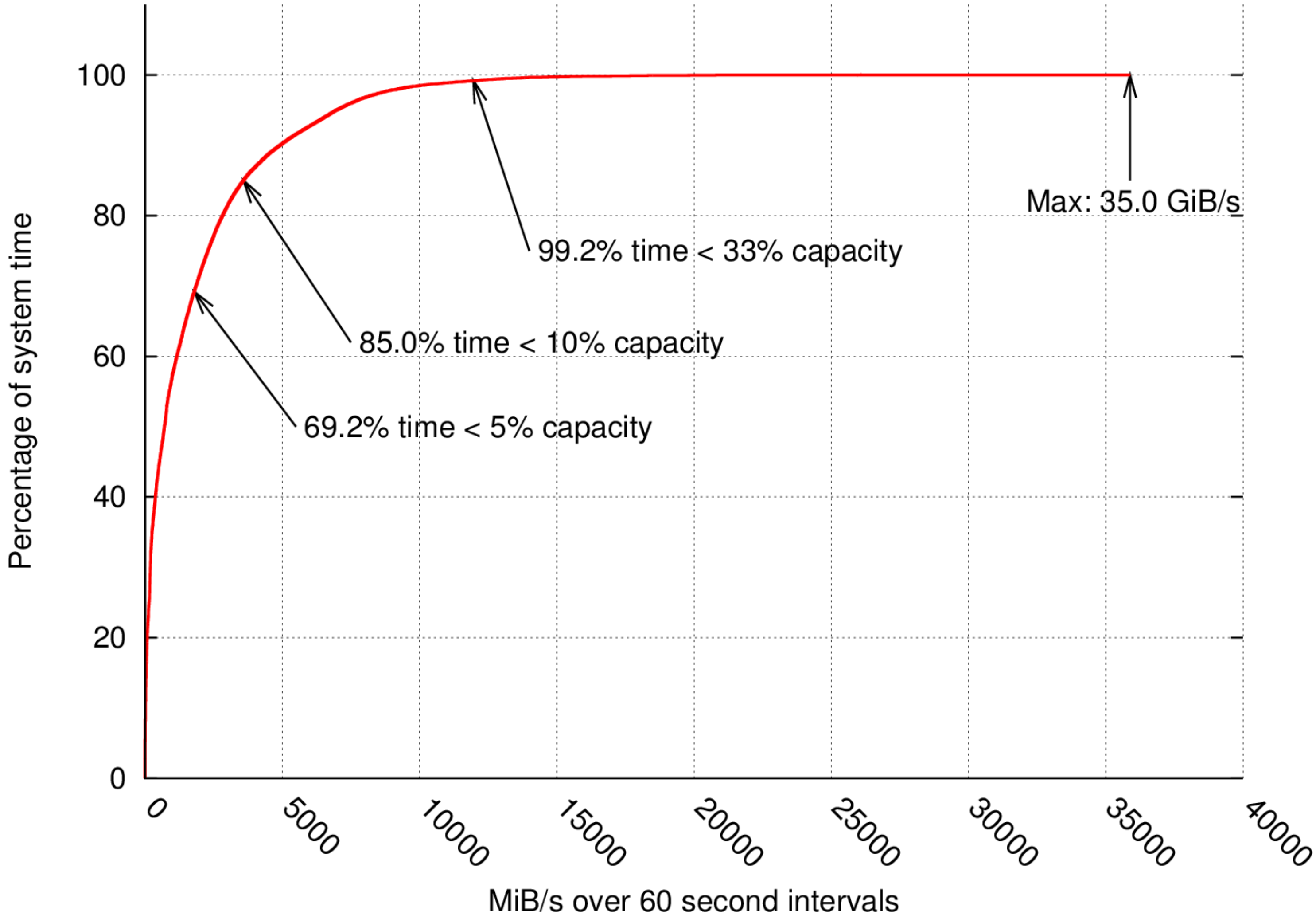


Broad range of performance:

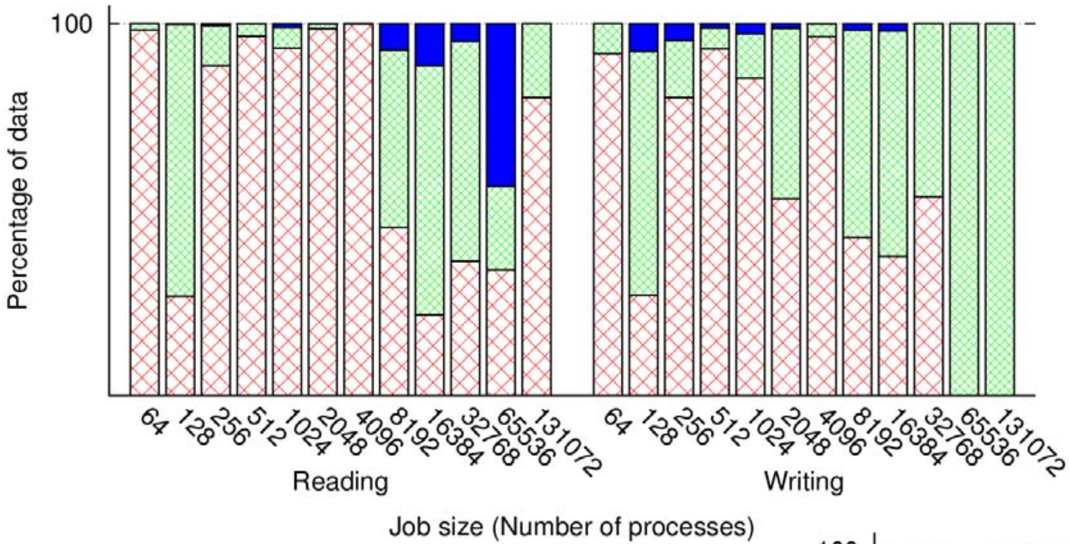
- but very little variability among jobs with consistent command line and job size
- The gap comes from use of two distinctly different executables:
- 1st executable:
  - example of manual aggregation: 512 of 4096 processes perform I/O
  - Read 1 TiB and write 500 GiB per job
  - Large access size, balanced I/O, pretty good performance
- 2nd executuable:
  - Rank-0 I/O for 2048 or 4096 processes
  - Poor relative performance
  - Is this a problem?
  - *Not yet*: only 1% of run time is spent performing I/O

Trends: burstiness

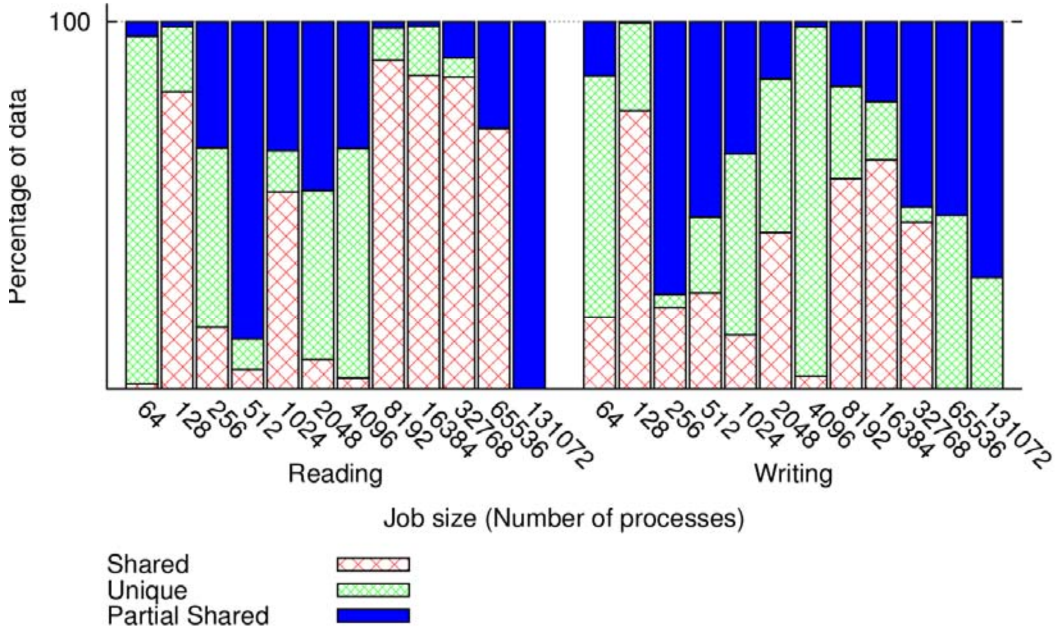
Cumulative distribution of aggregate throughput on 1 minute intervals



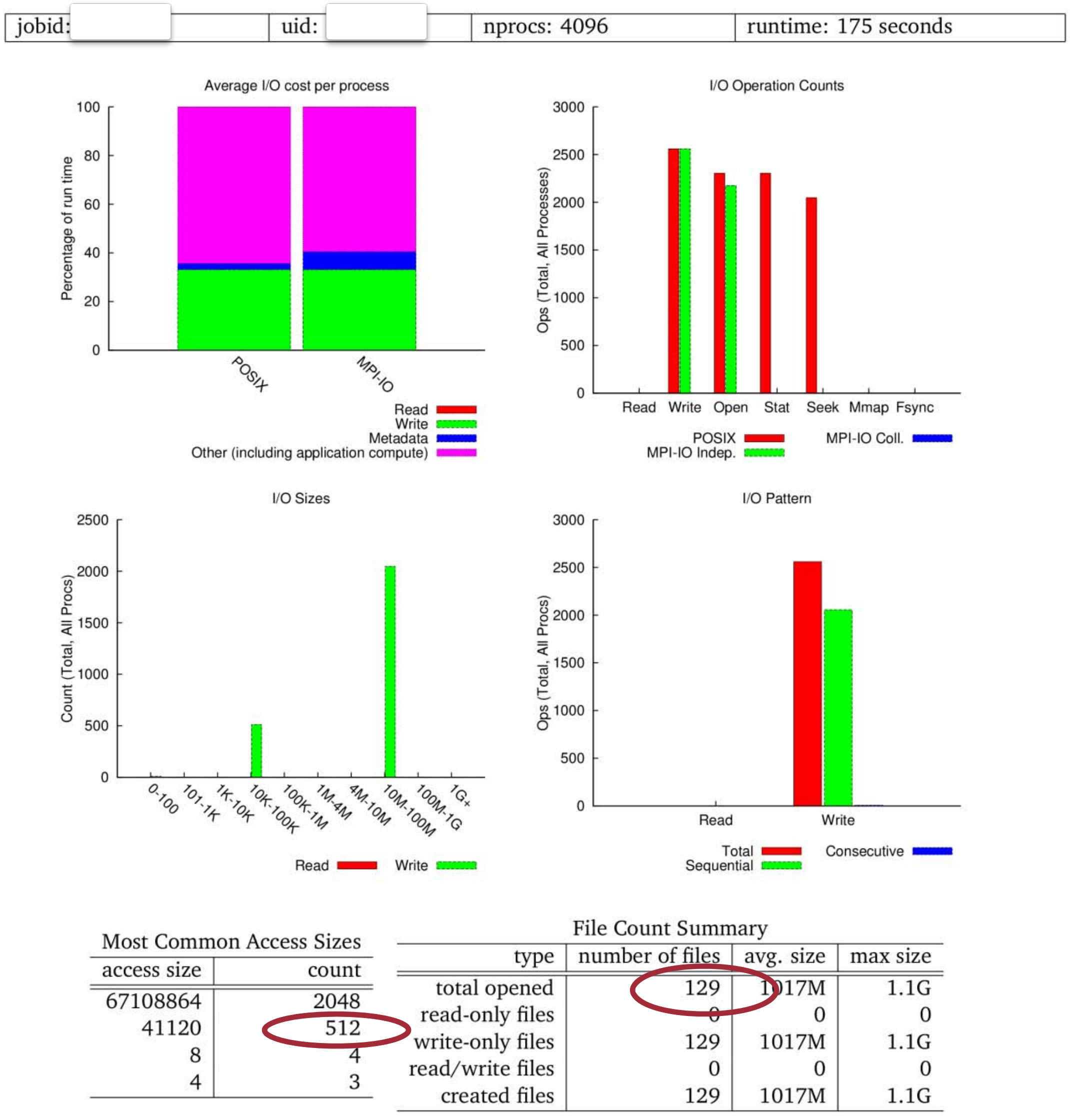
Trends in application behavior vs. scale



- Increased use of MPI-IO at larger scales
- Increased sharing of files among subsets of processes at larger scales



Examples of debugging and tuning with Darshan



jobid: [redacted]	uid: [redacted]	nprocs: 4096	runtime: 175 seconds

Most Common Access Sizes

access size	count
67108864	2048
41120	512
8	4
4	3

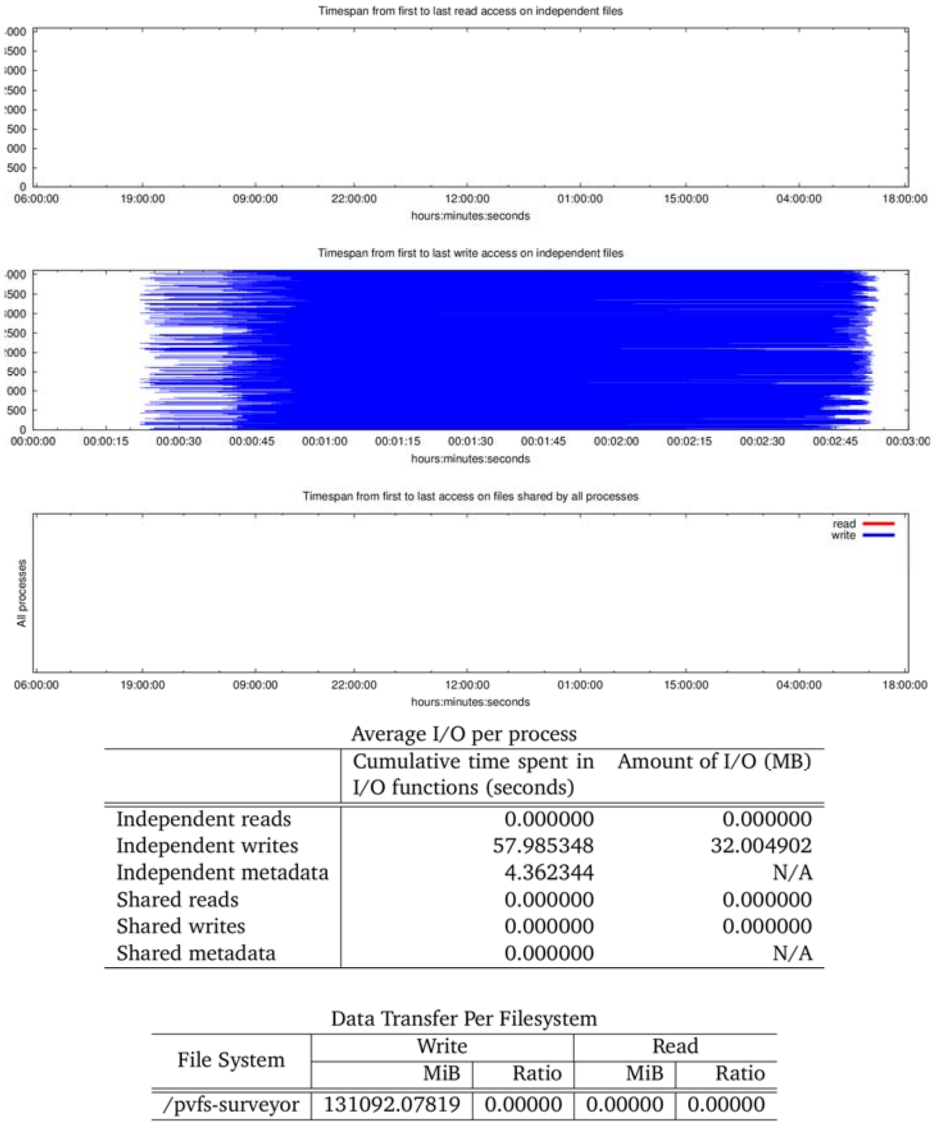
File Count Summary

type	number of files	avg. size	max size
total opened	129	1017M	1.1G
read-only files	0	0	0
write-only files	129	1017M	1.1G
read/write files	0	0	0
created files	129	1017M	1.1G

- Example output from job summary tool, available to all users

- Behavioral bug example:
- Mismatch between number of files vs. number of header writes
- The same header is being overwritten 4 times in each data file

Examples of debugging and tuning with Darshan



Average I/O per process

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MB)
Independent reads	0.000000	0.000000
Independent writes	57.985348	32.004902
Independent metadata	4.362344	N/A
Shared reads	0.000000	0.000000
Shared writes	0.000000	0.000000
Shared metadata	0.000000	N/A

Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/pvfs-surveyor	131092.07819	0.00000	0.00000	0.00000

- Possible performance bug:
- Why do some processes begin writing 25 seconds later than others?

Broad observations

- It is possible to instrument **entire leadership systems during production activity**
- System studies paint a different picture of I/O behavior than what we might expect from case studies and anecdotes
- Applications do crazy things (depending on your point of view!)
- There are several reasons:
  - Artifacts from porting:
    - Tuning strategies vary wildly across systems
    - This is a burden for application developers
  - Fumbling around in the dark:
    - How do users find out about I/O behavior and how to improve it?
    - Manual instrumentation is time consuming and prone to error
  - No one universal I/O strategy is consistently "the best"
- I/O behavior varies depending on the scale of the job

Opportunities for system software

- Useful observations for system software developers:
  - Frequent periods of “mostly” idle time, due to capability workload, maintenance, etc.
  - Files are either deleted quickly, or left unmodified for extended periods of time
- How can we take advantage of those characteristics?
  - Replication, compression, erasure coding, hierarchical storage, reorganization, etc.
- Where are our APIs (or advocacy of those APIs) falling short?
  - Many apps are rolling their own layout and aggregation strategy, despite efforts to provide tools in MPI-IO and HLLs

Future work

- Darshan everywhere, all the time, ***resistance is futile***
  - Improve coverage
  - Deploy on more systems that have different job characteristics for comparison
  - TACC example:
    - LD_PRELOAD instrumentation, coverage of mpich1, mpich2, openmpi and several compilers
    - Tuning Darshan itself for different I/O systems
- Continue file system and device instrumentation, possibly expand into vendor-specific instrumentation
- Automate how we leverage the data:
  - identification of applications that need help
  - tuning suggestions (for application or system parameters)
- Correlation and prediction of characteristics vs. expected performance
- Upcoming Darshan releases

Is the data and software available?

Yes! Announcing the Darshan Data Repository:

http://www.mcs.anl.gov/darshan/data

- Anonymized version of data from the time period studied in the paper
- Just Darshan data for now, not iostat or fsstats
- We hope to post more data in the future
- Darshan 2.1.1 (available in a few days) will be able to parse anonymized logs, in the mean time please use svn trunk

- Darshan itself has been available for over a year
- Open source and portable across a variety of systems

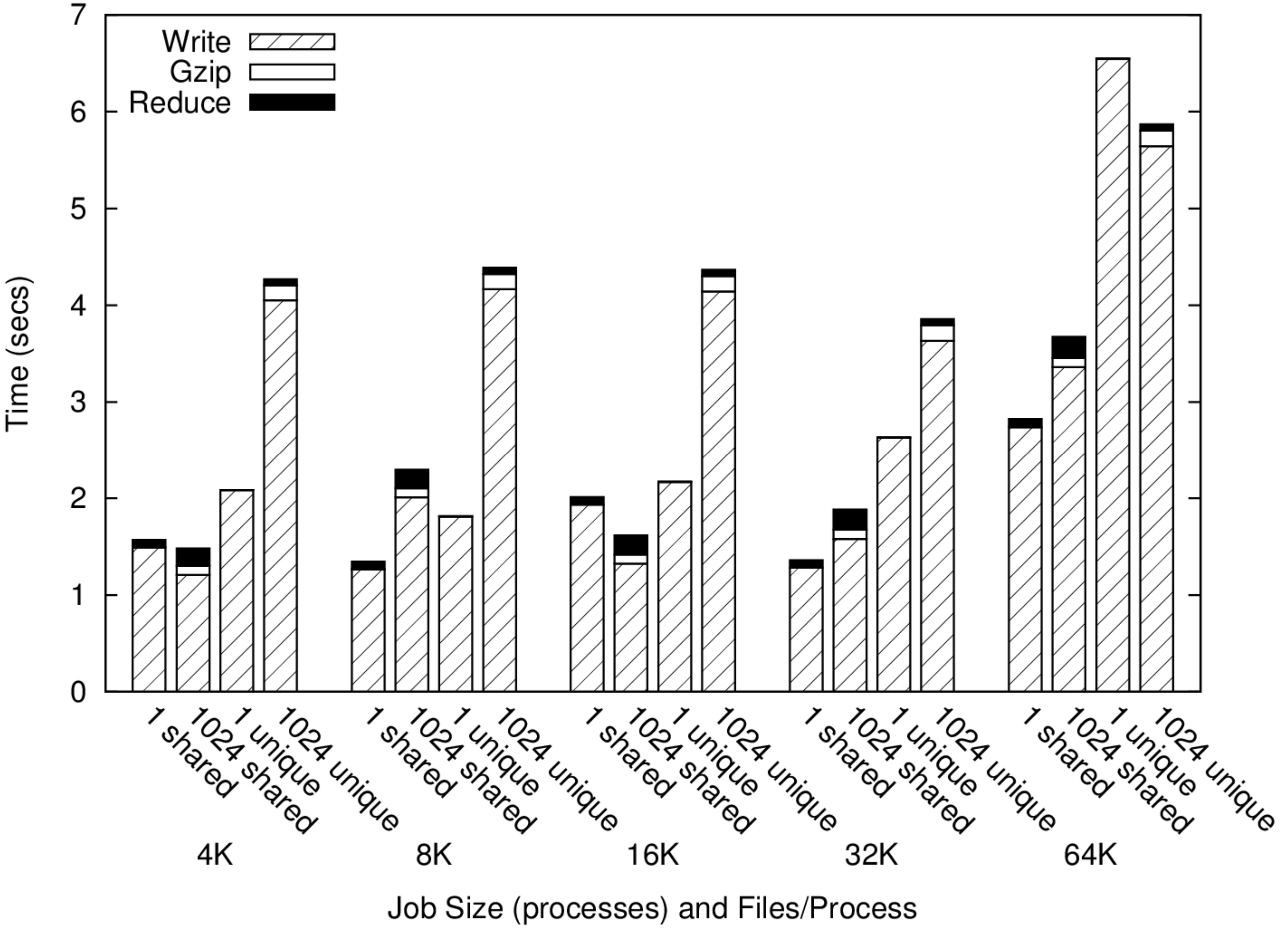
- This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.
- Thank you to the ALCF management, administrators, and power users at Argonne
  - Positive, helpful responses from everyone involved
- Thank you to the authors of fsstats, iostats, and the Charisma project

Phil Carns

pcarns@mcs.anl.gov

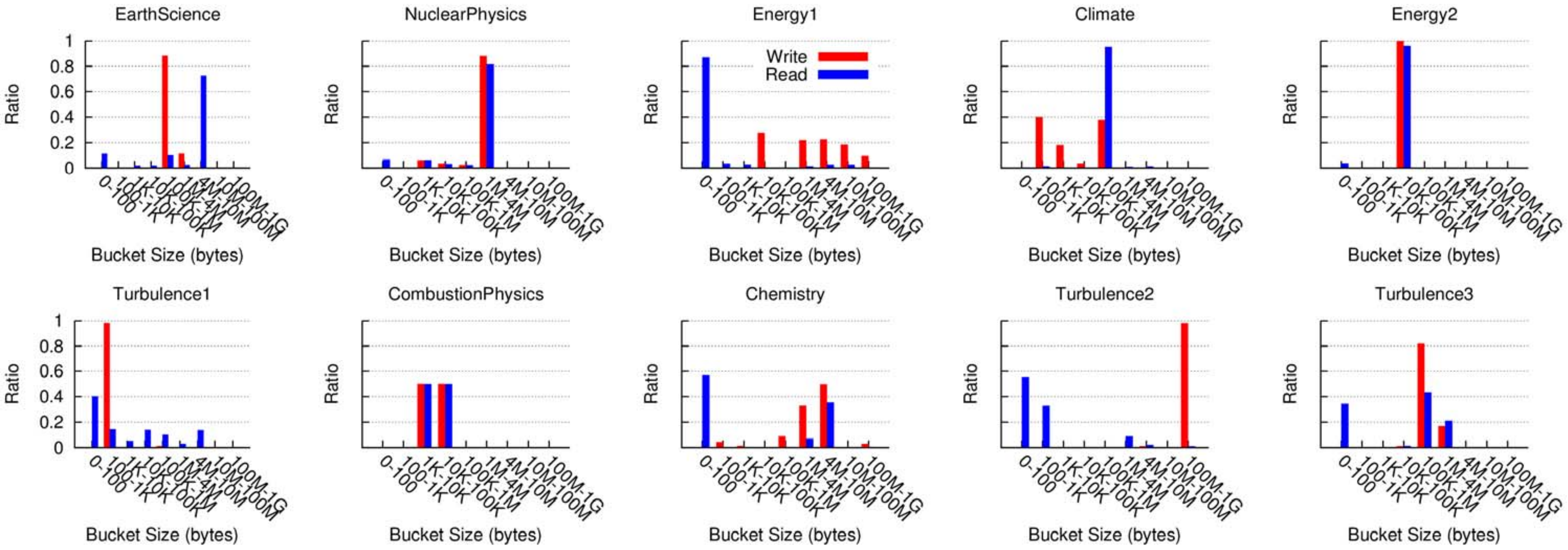
http://www.mcs.anl.gov/darshan

Darshan overhead at job shutdown time

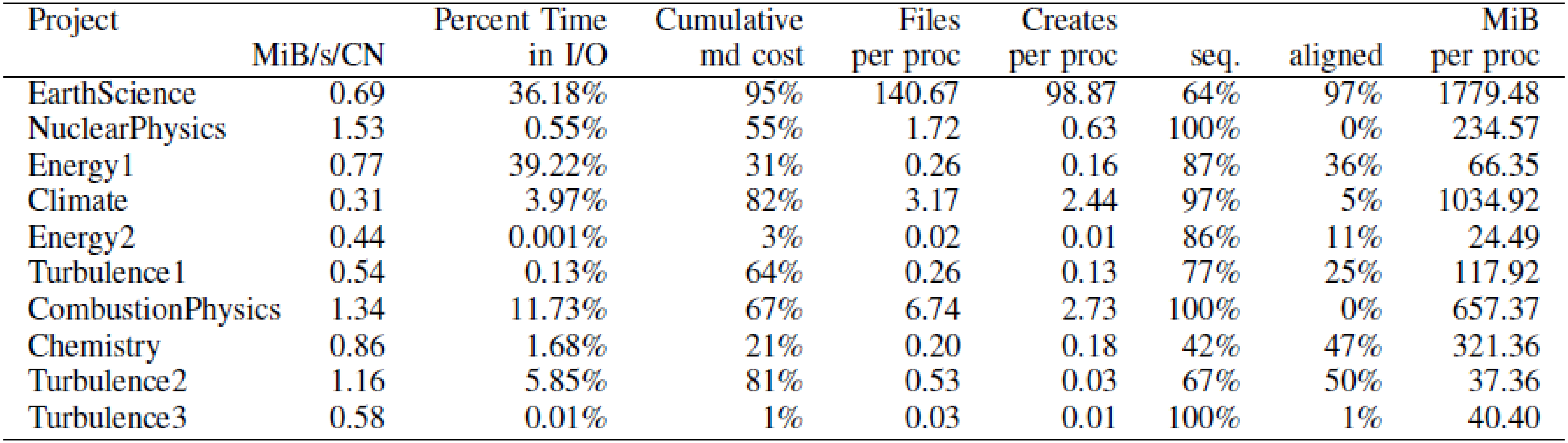


Largest example:
67 million files
characterized

Access size histograms per project



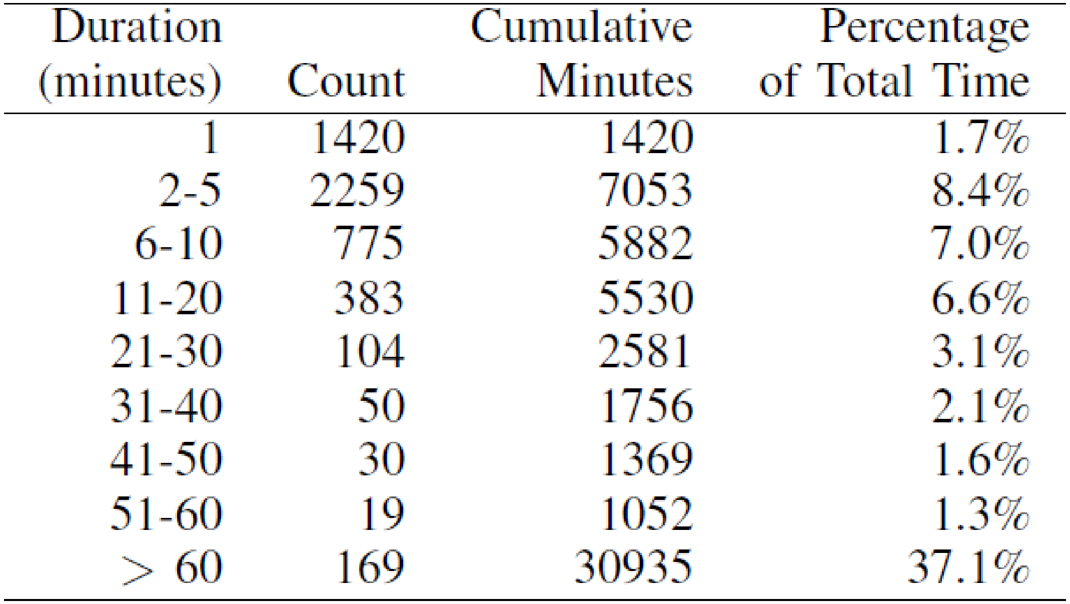
I/O Characteristics by project



Project	MiB/s/CN	Percent Time in I/O	Cumulative md cost	Files per proc	Creates per proc	seq.	aligned	MiB per proc
EarthScience	0.69	36.18%	95%	140.67	98.87	64%	97%	1779.48
NuclearPhysics	1.53	0.55%	55%	1.72	0.63	100%	0%	234.57
Energy1	0.77	39.22%	31%	0.26	0.16	87%	36%	66.35
Climate	0.31	3.97%	82%	3.17	2.44	97%	5%	1034.92
Energy2	0.44	0.001%	3%	0.02	0.01	86%	11%	24.49
Turbulence1	0.54	0.13%	64%	0.26	0.13	77%	25%	117.92
CombustionPhysics	1.34	11.73%	67%	6.74	2.73	100%	0%	657.37
Chemistry	0.86	1.68%	21%	0.20	0.18	42%	47%	321.36
Turbulence2	1.16	5.85%	81%	0.53	0.03	67%	50%	37.36
Turbulence3	0.58	0.01%	1%	0.03	0.01	100%	1%	40.40

Trends: burstiness

Breakdown of time periods in which throughput was below 5% of peak capacity



Duration (minutes)	Count	Cumulative Minutes	Percentage of Total Time
1	1420	1420	1.7%
2-5	2259	7053	8.4%
6-10	775	5882	7.0%
11-20	383	5530	6.6%
21-30	104	2581	3.1%
31-40	50	1756	2.1%
41-50	30	1369	1.6%
51-60	19	1052	1.3%
> 60	169	30935	37.1%

- Data is only 1 minute granularity, so some extremely short bursts are averaged out
- More precision needed to observe time periods with zero activity