Production I/O Characterization on the Cray XE6

Phil Carns[1], Kevin Harms [2], Rob Latham[1], Rob Ross[1]

[1]Mathematics and Computer Science Division

[2]Argonne Leadership Computing Facility

Argonne National Laboratory

Yushu Yao, Katie Antypas

National Energy Research Scientific Computing Center

Lawrence Berkeley National Laboratory

CUG 2013

Motivation

- I/O behavior plays a key role in productivity for large-scale HPC systems
  - Our target: Hopper, a 153,216 core Cray XE6 at NERSC
- Challenges in understanding I/O behavior
  - Hundreds of users across a wide spectrum of science
  - Applications vary in data volume, I/O strategy, and access methods
  - How can we consistently characterize production I/O behavior across applications?
  - How do we quickly identify applications that could most benefit from additional tuning assistance?

Approach

1. Adapt the **Darshan** I/O characterization tool for use in the Cray environment
   - Tune to reflect Hopper system characteristics
   - Integrate transparently for maximum coverage
2. Evaluate Darshan for production deployment
   - Measure overhead at scale for multiple workloads
3. Deploy Darshan
   - Store characterization data for post-processing and exploration
   - Provide immediate feedback to users
4. Develop tools that leverage Darshan data
   - Rapid feedback to power users
   - Metrics to automatically flag jobs that exhibit unusual I/O behavior for administrators

Darshan background

Darshan is an open source, application-level instrumentation library that uses link-time instrumentation for static executables and LD_PRELOAD for dynamic executables.

What does it record?

- Counters, histograms, and strategically chosen timestamps related to I/O activity
- POSIX, POSIX stream, MPI-IO, and limited HDF5 and PNetCDF functions
- Access patterns, access sizes, I/O time, alignment, datatypes, etc.
- Builds on characterization ideas from Kotz and Nieuwejaar Charisma study
- Does *not* record a complete trace of I/O operations

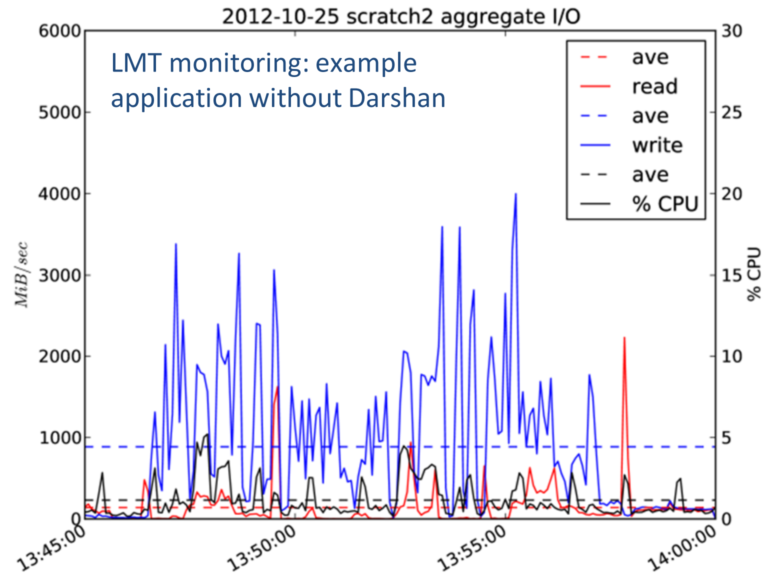
How does it store results?

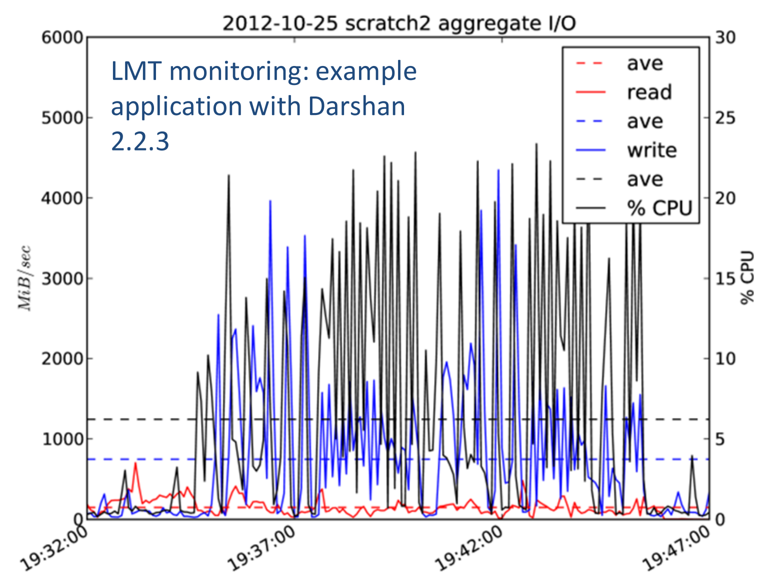
- Minimal overhead during execution
- Reduction, compression, and persistent storage at MPI_Finalize() time
- Produces a single, compact log for each instrumented job

Tuning example: shared-file instrumentation

- Initial performance experiments with Darshan 2.2.3 exhibited performance degradation for shared-file applications
- LMT monitoring of servers revealed that the raw I/O throughput was comparable, but CPU usage was much higher

- Source of overhead: issuing stat() calls on each rank to collect additional information at open() time
- Concurrent stat() at 12,288 processes can already add 2 seconds to file open time
- Worked to develop alternative instrumentation

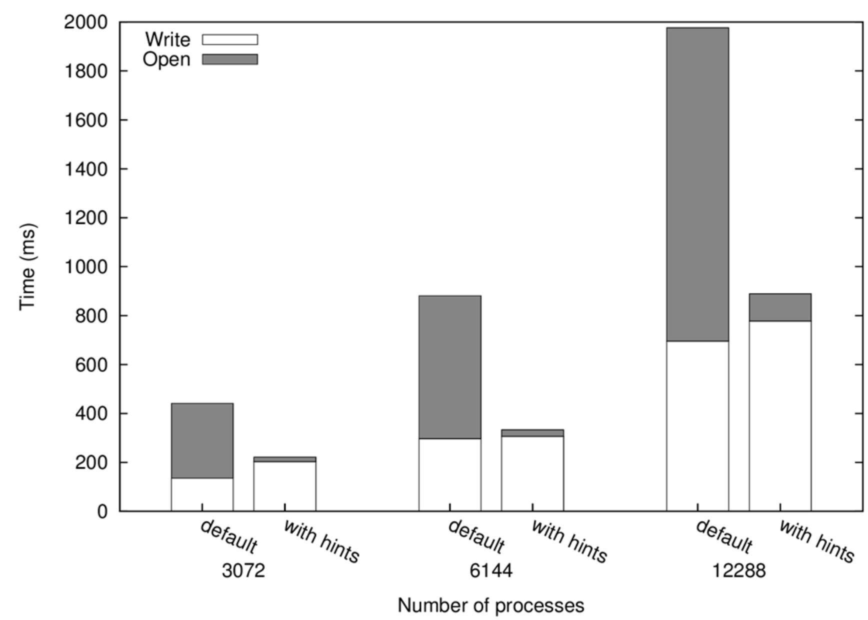




Tuning example: writing log files efficiently

- Darshan writes all results to a unified log file at shutdown after custom reduction and gzip compression
- Final results are typically quite small: hundreds of bytes per process, sometimes even less
- Regardless of how the application performed I/O, Darshan itself uses MPI-IO collectives internally to write results
- Portable and efficient: leverages aggregation and stripe alignment

- We improved MPI-IO collective performance even further with hints:
  - romio_no_indep_rw
  - cb_nodes=4
- Limit the number of processes that actually open the output file
- Drastically reduces the cost of writing data at larger process counts



System integration challenges

- Cray environment:
  - Multiple compilers (Cray, PGI, Pathscale, Intel, GNU)
  - Static linking by default
  - Unified cc, ftn, and CC compiler scripts

- Our requirements:
  - Support as many configurations as reasonably possible
  - Enable and disable via software module
  - Transparent for users (no need for different compilers or link options)

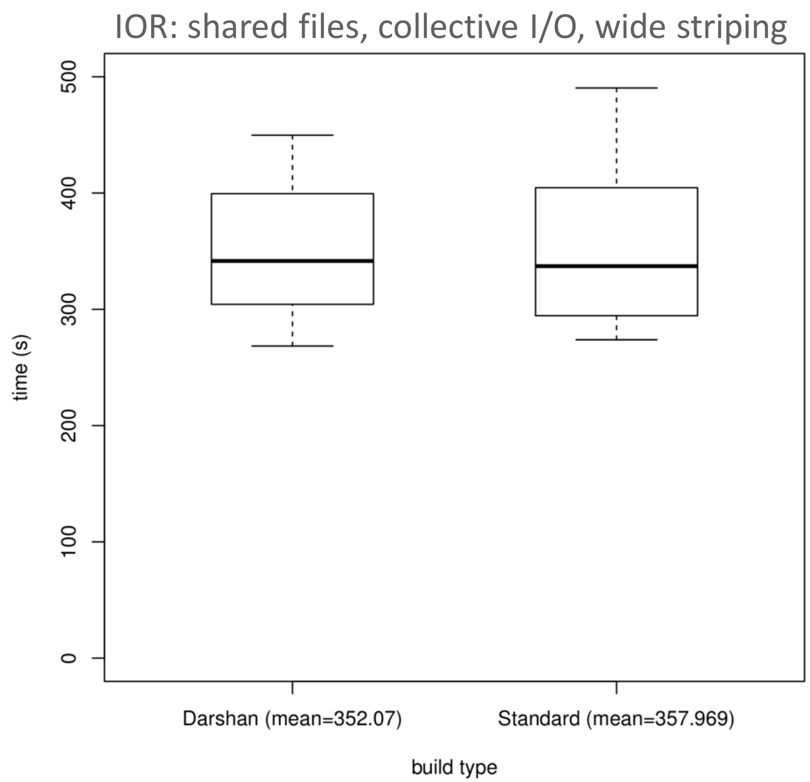
- We experimented with multiple deployment methods during this study
- Our plan moving forward is to use the PE_PRODUCT_LIST mechanism
  - This is a set of environment variables that can be used by software modules to specify additional linker options for the Cray compiler scripts

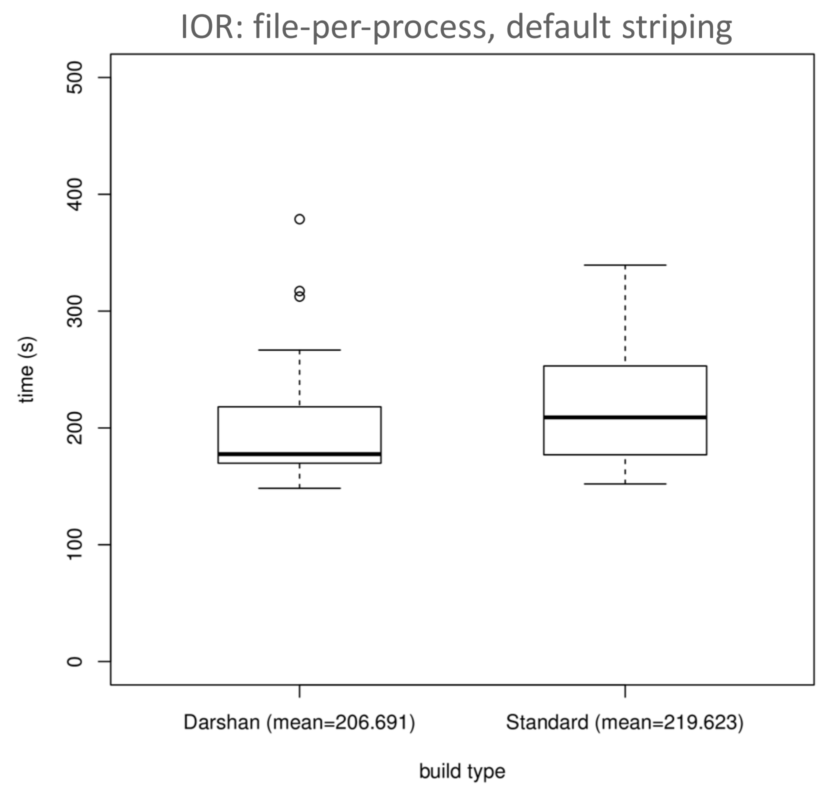
Evaluation: end-to-end overhead

After adapting Darshan to the Cray XE6 environment, our next goal was to evaluate the impact of Darshan on large-scale applications to verify suitability for production deployment.

- First experiment: measure end-to-end run time of IOR benchmark writing and reading 1.5 TiB of data at 12,288 processes
  - Includes all Darshan startup, instrumentation, and shutdown costs
  - Measured by timing the “aprun” command
  - Evaluate both shared-file and file-per-process examples
  - IOR configured to use MPI-IO; Darshan instrumented both the MPI-IO and POSIX API calls
- Individual runs are susceptible to variance
- Gathered 20 independent samples for each test case over a four day period and analyzed the results

Evaluation: end-to-end overhead

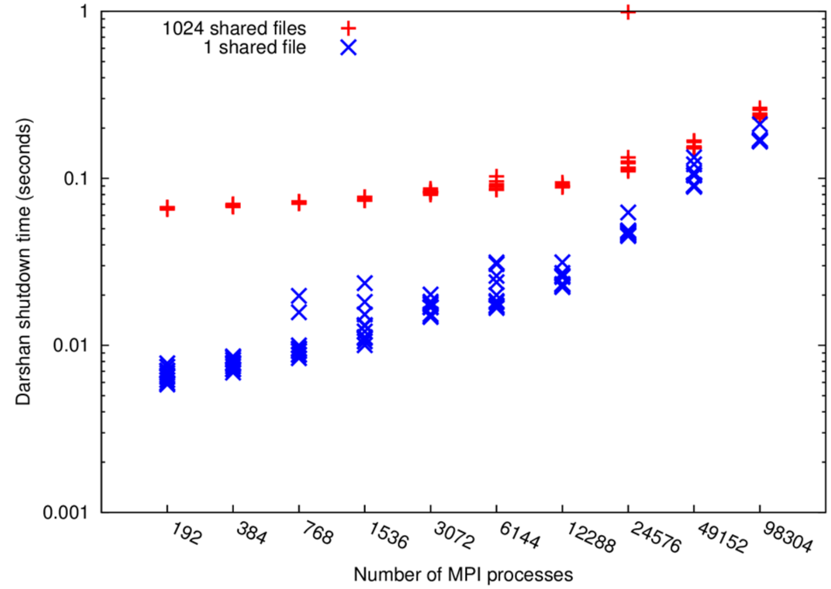


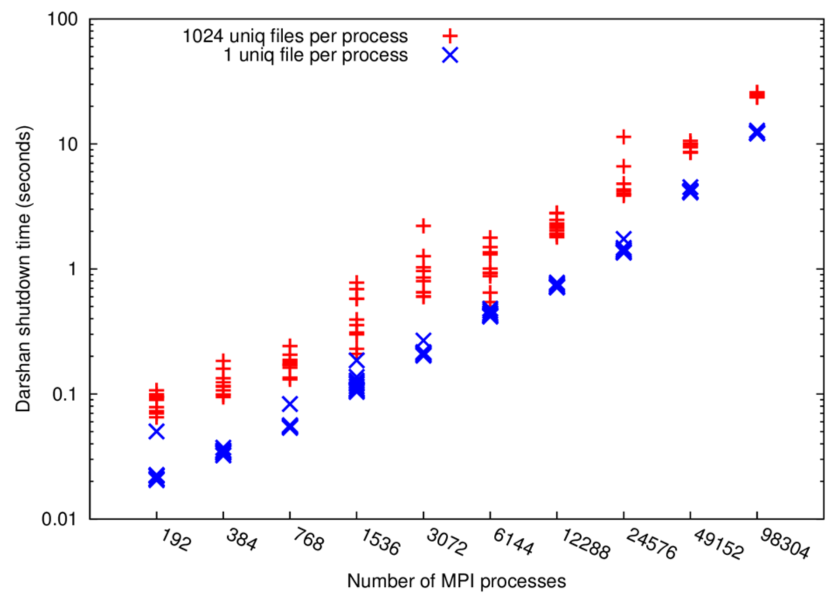


- Box plots of 20 samples each indicate no clear Darshan overhead at 12,288 processes relative to normal system variance
- Insufficient evidence for difference in mean run time (t-test)
- Other observations: variance is significant, and file-per-process access patterns perform better on Hopper at this scale

Evaluation: Darshan shutdown costs

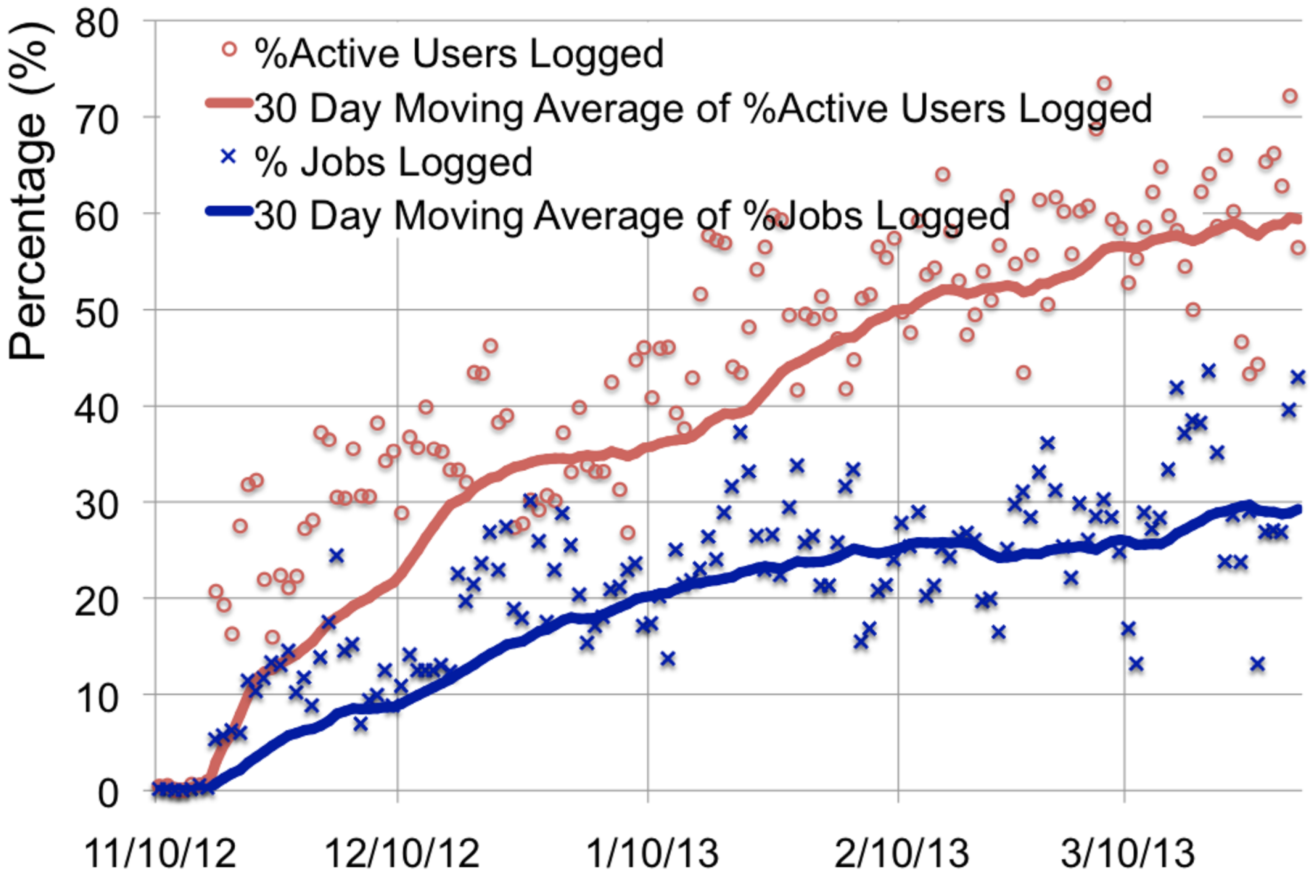
- End-to-end Darshan overhead is obscured by I/O variance, but we can measure internal Darshan mechanisms directly using microbenchmarks
- These examples show the total time required by Darshan to reduce, compress, and write log files at MPI_Finalize() time
- Average one-time cost for 98,304 processes:
  - 1 shared file: 0.17 seconds
  - 1024 shared files: 0.24 seconds
  - 1 file-per-process: 12.3 seconds
  - 1024 files-per-process: 24.8 seconds
- Largest scenario: emulates application opening 100,663,296 unique files
- Darshan falls back to less granular instrumentation if memory threshold is exceeded by opening too many files





Deployment: coverage as of March 2013

- Percentage of active users and jobs instrumented per day since initial Hopper deployment
- Nearing an average of 60% and 30%, respectively, by end of March 2013

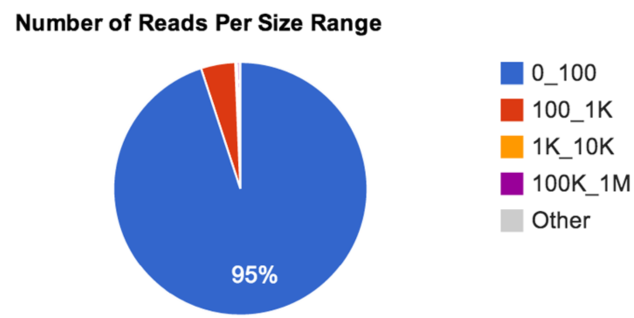


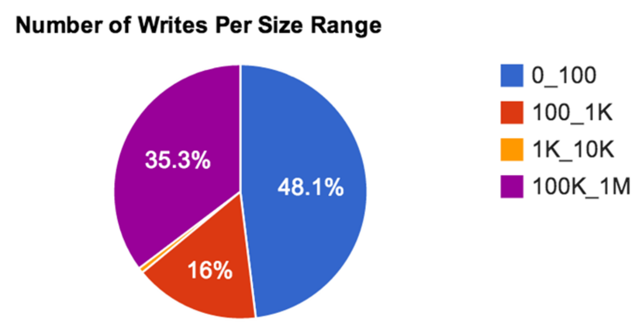
Deployment: user experience

- The NERSC web portal allows users to interact with their jobs and allocations
- This screenshot shows I/O summary information automatically generated from Darshan logs for completed jobs
- Provides rapid (within a few minutes) initial feedback on I/O behavior

IO Summary from Darshan

Start	End	Wallclock (secs)	MB Read	MB Written	Estimated I/O Rate (MB/sec)	Estimated Percent Time Spent in I/O
04-05 16:04:05	04-05 16:06:51	166	590.3	597.6	355.52	2.01%





Metrics: redundant read traffic

- Administrators can also filter logs using metrics designed to automatically identify applications that may benefit from tuning assistance
- We explored three example metrics that can be quickly computed from Darshan log data
- First example is redundant read traffic: applications that read more bytes of data from the file system than were present in the file
- Even with caching effects, this type of job can cause disruptive I/O network traffic through redundant file system transfers
- Candidates for aggregation or collective I/O

Summary of matching jobs:

Redundant read threshold	> 1 TiB
Total jobs analyzed	261,890
Jobs matching heuristic	671
Unique users matching heuristic	37
Largest single-job redundant read volume	547 TiB

Top example

- Scale: 6,138 processes
- Run time: 6.5 hours
- Avg. I/O time per process: 27 minutes
- Metric: Read 548.6 TiB of data from a 1.2 TiB collection of read-only files
- Used 8 KiB read operations and generated 457 X redundant read traffic

Metrics: metadata time ratio

- Percentage of cumulative I/O time spent performing metadata operations such as open(), close(), stat(), and seek()
- Close() cost can be misleading due to write-behind cache flushing, but metadata ratio is often a key indicator of inefficient file organization
- Most relevant for jobs that performed a significant amount of I/O
- Candidates for coalescing files and eliminating extra metadata calls like stat() where possible

Summary of matching jobs:

Thresholds	meta_time / nprocs $>$ 30 s nprocs $\geq$ 192 metadata_ratio $\geq$ 25%
Total jobs analyzed	261,890
Jobs matching heuristic	252
Unique users matching heuristic	45
Largest single-job metadata ratio	$>$ 99%

Top example

- Scale: 40,960 processes
- Run time: 229 seconds
- Max. I/O time per process: 103 seconds
- Metric: 99% of I/O time in metadata operations
- Generated 200,000+ files using 600,000+ write() and 600,000+ stat() calls

Metrics: small writes to shared files

- Small writes can contribute to poor performance as a result of poor file system stripe alignment, but there are many factors to consider:
  - Writes to non-shared files may be cached aggressively
  - Collective writes are normally optimized by MPI-IO
  - Throughput can be influenced by additional factors beyond write size
- We searched for jobs that wrote less than 1 MiB per operation to shared files without using any collective operations
- Candidates for collective I/O or batching/buffering of write operations

Summary of matching jobs:

Thresholds	> 100 million small writes 0 collective writes
Total jobs analyzed	261,890
Jobs matching heuristic	220
Unique users matching heuristic	11
Largest single-job small write count	5.7 billion

Top example

- Scale: 128 processes
- Run time: 30 minutes
- Max. I/O time per process: 12 minutes
- Metric: issued 5.7 billion write operations, each less than 100 bytes in size, to shared files
- Averaged just over 1 MiB/s per process during shared write phase

Summary and conclusions

Lessons learned while adapting Darshan for the Cray environment:

- Adapting system tools to the Cray XE6 environment requires consideration of file system characteristics and environment configuration
- "Minimal overhead" is a moving target! New platforms bring new challenges.
- I/O variance remains a significant factor in I/O performance
- Darshan performance is suitable for full-time production deployment

Deployment experience:

- It is possible to perform application-level I/O characterization of full-scale production workloads
- Darshan has been successfully deployed on the Hopper system at NERSC
  - Rolled out in stages from November 2012 to February 2013
  - 30% of jobs and compute hours instrumented by March 2013
- Deployment impact:
  - Immediate feedback to users on I/O behavior
  - Ability to automatically scan jobs using simple metrics to identify applications that need further assistance

Future work

- Continue to improve Darshan
  - Use system-specific mechanisms for characterization when available
  - Improve Cray environment integration
  - Modularization

- Continue to use Darshan data to solve I/O problems
  - Use metrics to identify and help the users who need it most
  - More rigorous analysis methods
  - Improve feedback for users
  - Broader workload studies

This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract Nos. DE-AC02-06CH11357 and DE-AC02-05CH11231 including through the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization.

Thank you to the NERSC management, administrators, and power users who helped to make this work possible.

Phil Carns

carns@mcs.anl.gov

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