CSE 5449: Intermediate Studies in Scientific Data Management

Lecture 13: Mining I/O logs for root causes of performance bottlenecks

Dr. Suren Byna

The Ohio State University

E-mail: byna.1@osu.edu

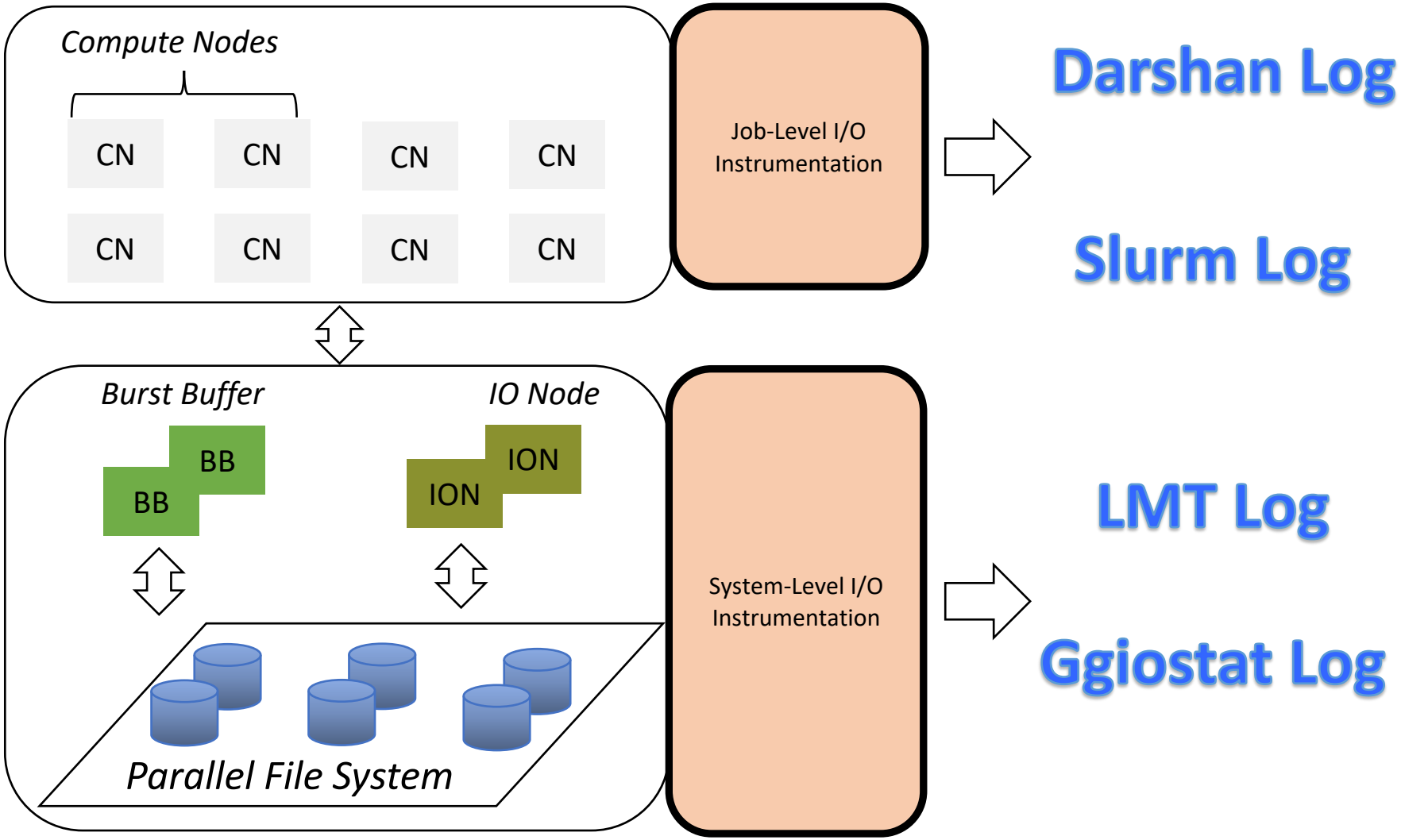
https://sbyna.github.io

02/21/2023

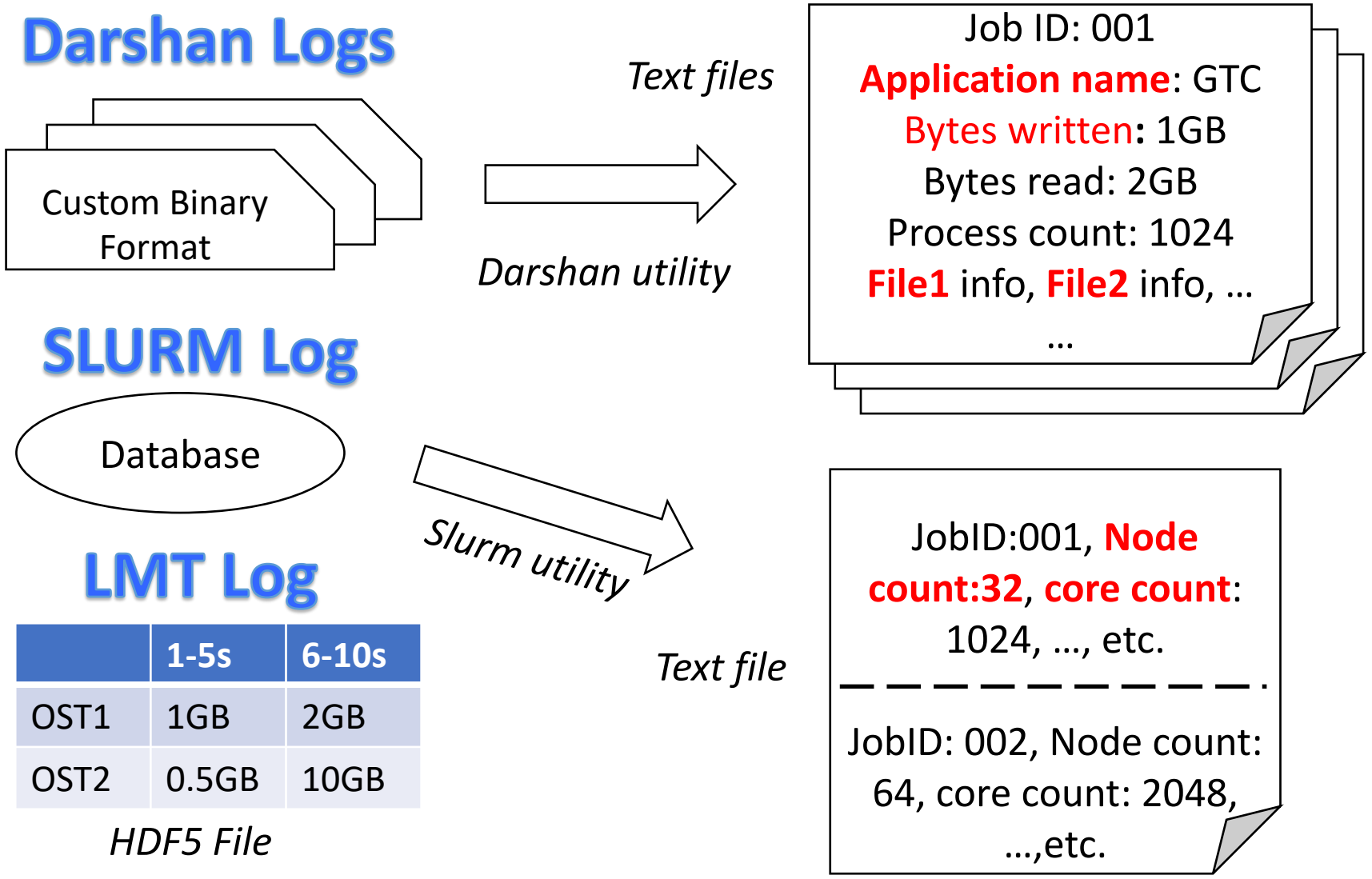
Today's class

- Any questions?
- Progress of the project
- Today's class –
  - Performance analysis – mining I/O logs for root causes of performance bottlenecks

I/O instrumentation data are collected at different I/O stages



Diverse storage formats of different instrumentation data



What can we learn from multi-level I/O instrumentation data

- Analytics based on Darshan
  - **Application-level I/O statistics:** e.g., finding the top I/O consumers, low-bandwidth jobs, etc.
  - **Application-level I/O performance analysis**: e.g., finding the root causes for applications' poor I/O performance.
- Analytics based on Slurm
  - **Application-level I/O statistics:** Finding the number of nodes used by a job.
- Analytics based on LMT logs
  - **System-level I/O statistics:** Calculating the amount of data written to/read on the parallel file system during a given period, etc.

4

Challenges of analyzing existing I/O logs

- Ad hoc interfaces and data formats of different logs require nontrivial manual effort.

- Lack of a read-friendly data format for fast parallel analytics.
  - E.g., Darshan produces one log file for each job.
  - Analysis that requires all jobs' information takes a long time.

- Users have to manually look at each job's log file to identify the root causes of its poor I/O performance.

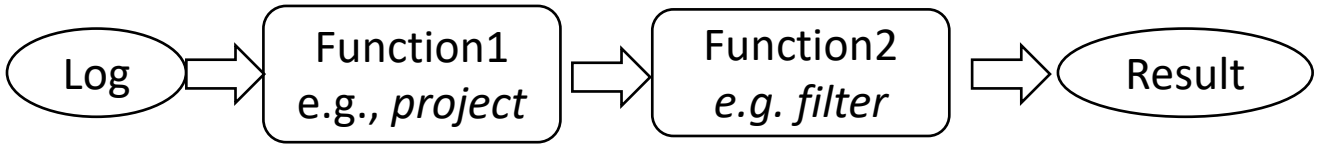
5

Existing approaches in I/O analytics

	Representative works	Unified interfaces and data format?	Use parallel analytics?	Support root cause analysis for poor I/O performance based on applications' I/O patterns?
Application-level	HPDC'15, CUG'16, ICCCN'13, etc.	No	No	No
System-level	ICDCS'17, PDSW'15, SC'13, etc.	No	No	No
Multi-Level	GUIDE (SC'16)	Yes	Yes	No
Multi-Level	Pytokio (CUG'18)	Yes	No	No
Multi-Level	IOMiner	Yes	Yes	Yes

Overview of IOMiner

- Provide a set of functions for I/O statistics and performance analysis on multi-level instrumentation data
  - Statistics: e.g. finding the top I/O-intensive applications, total I/O traffic on the parallel file system during a given period.
  - Performance analysis: e.g. identify the potential root causes for applications' poor I/O performance.
- Leverage Spark for parallel I/O analytics
  - Loading and analyzing I/O logs with a single process is not scalable.
  - Parallel databases are not readily available on HPC.
- Use SparkRDD to chain together a set of analytics functions and compose a high-level analytics



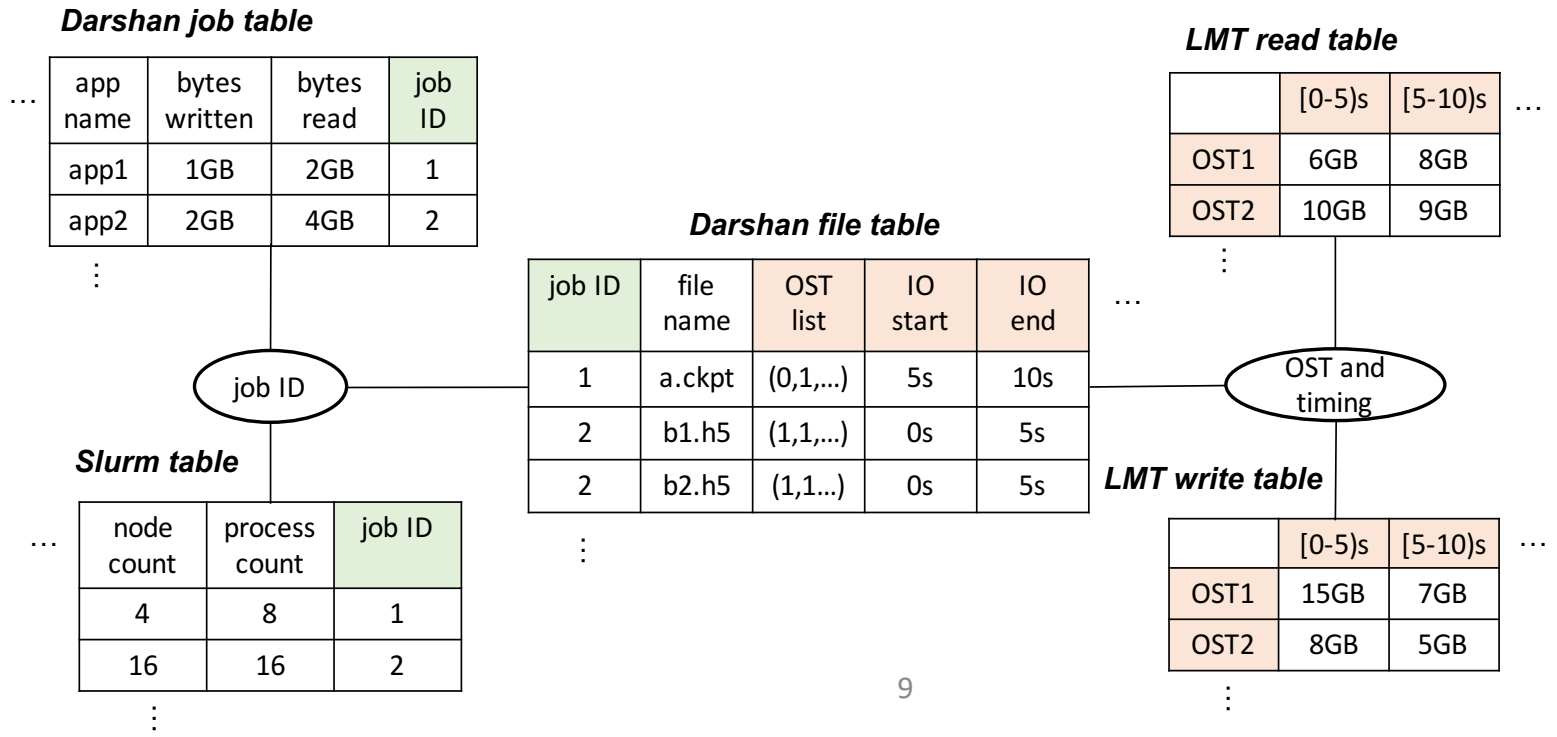
High-level interfaces

- Storage construction
  - miner.init_stores(start_date, end_date)
  - Format Darshan, LMT, and Slurm logs during a period into a unified table format
  - darshan_tuples = miner.load(“darshan_job”)
  - Load darshan table into memory
- Common query functions layered on top of Spark
  - filter, project, group, join, sort, percentile, bin, etc.
  - E.g. darshan_tuples.project(“job_id”, ”bytes_written”).filter(“bytes_written < 1GB”)
  - Find the jobs who have written less than 1GB data.
- Identify root factors for jobs with poor I/O performance
  - E.g. perf_factors = low_bw_tuples.extract_perf_factors()
  - Return a list of performance contributing factor values for low-bandwidth jobs, such as small I/O percent, non-consecutive I/O percent, etc.

8

Unified storage format

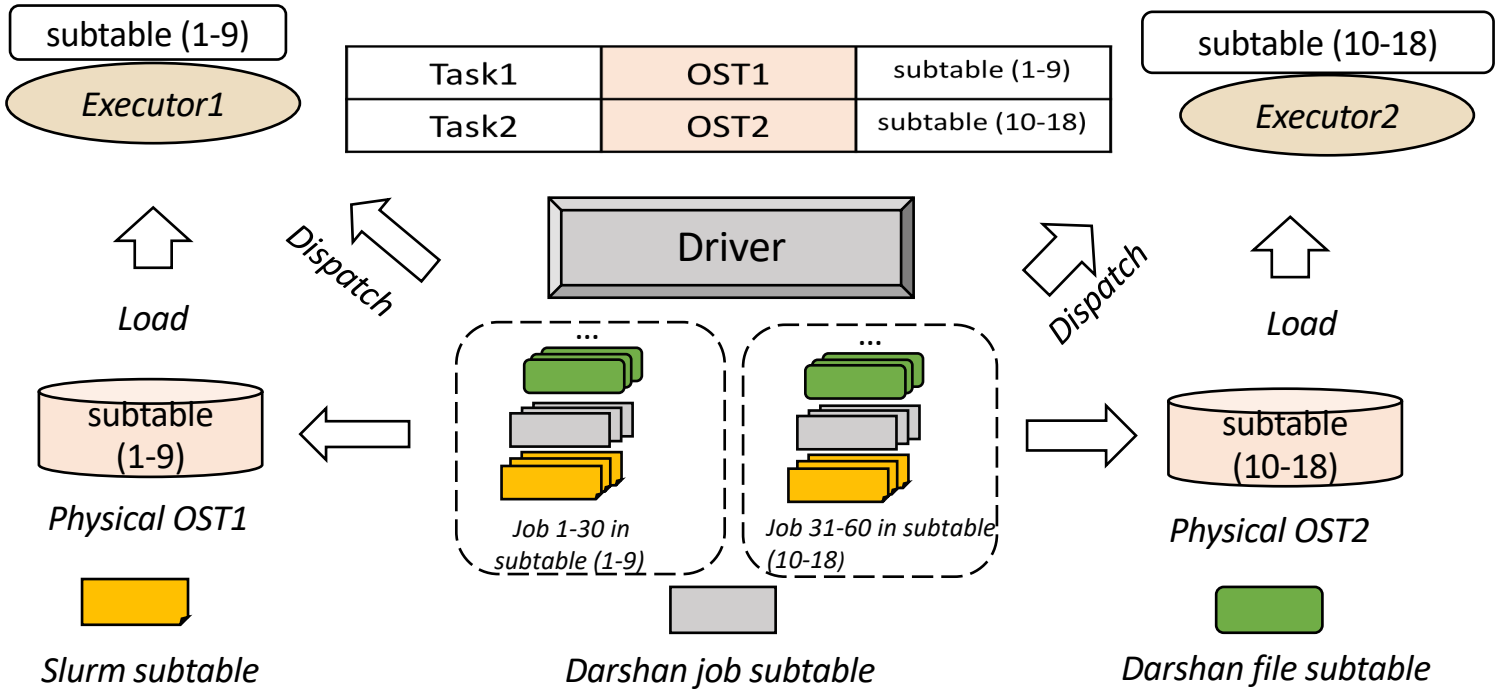
- Different instrumentation data have distinct formats
- Format Darshan, Slurm, and LMT logs into tables.
- Associate different tables with the colored fields.



9

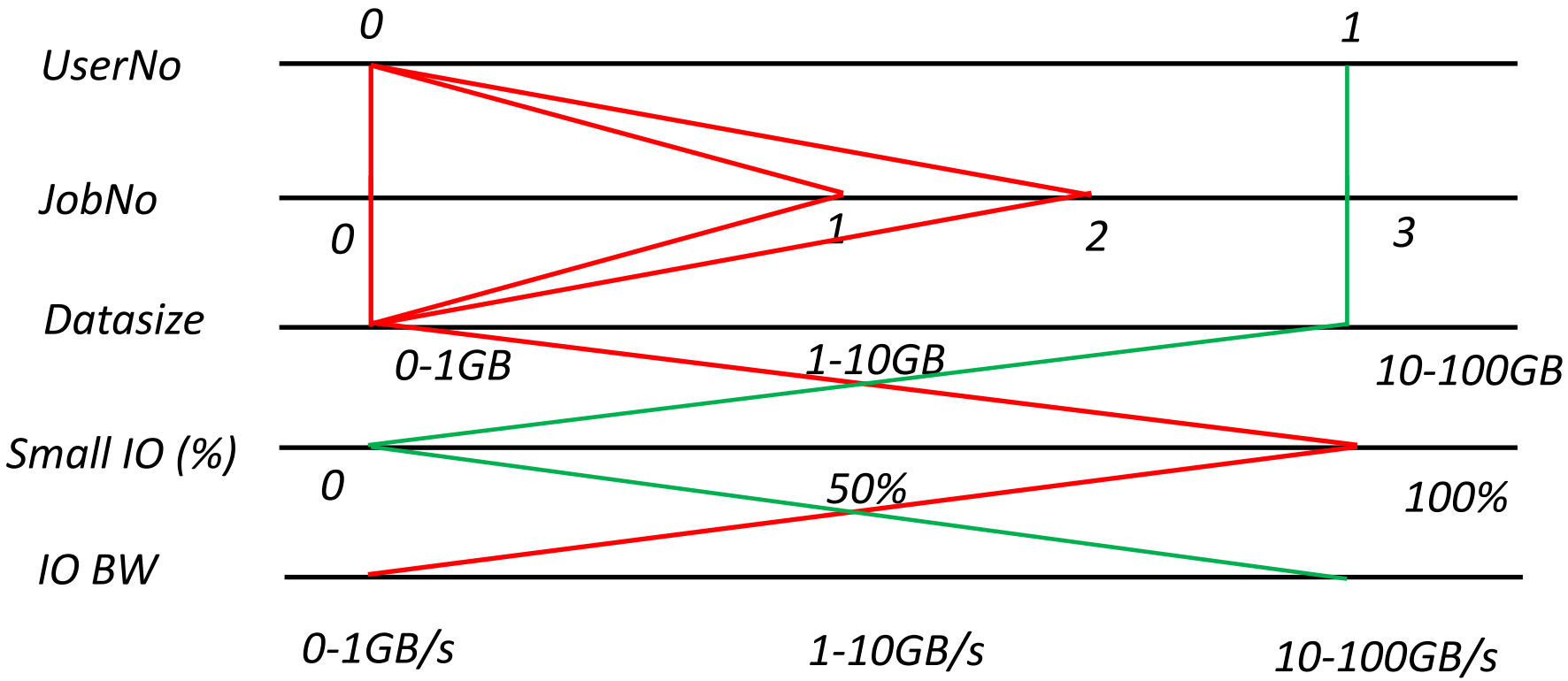
Data layout for parallel data analytics based on Spark

- However, using a few large tables does not yield good parallelism.
- Split Darshan, Slurm tables into subtables, and Spark driver dispatches these tables to different executors.



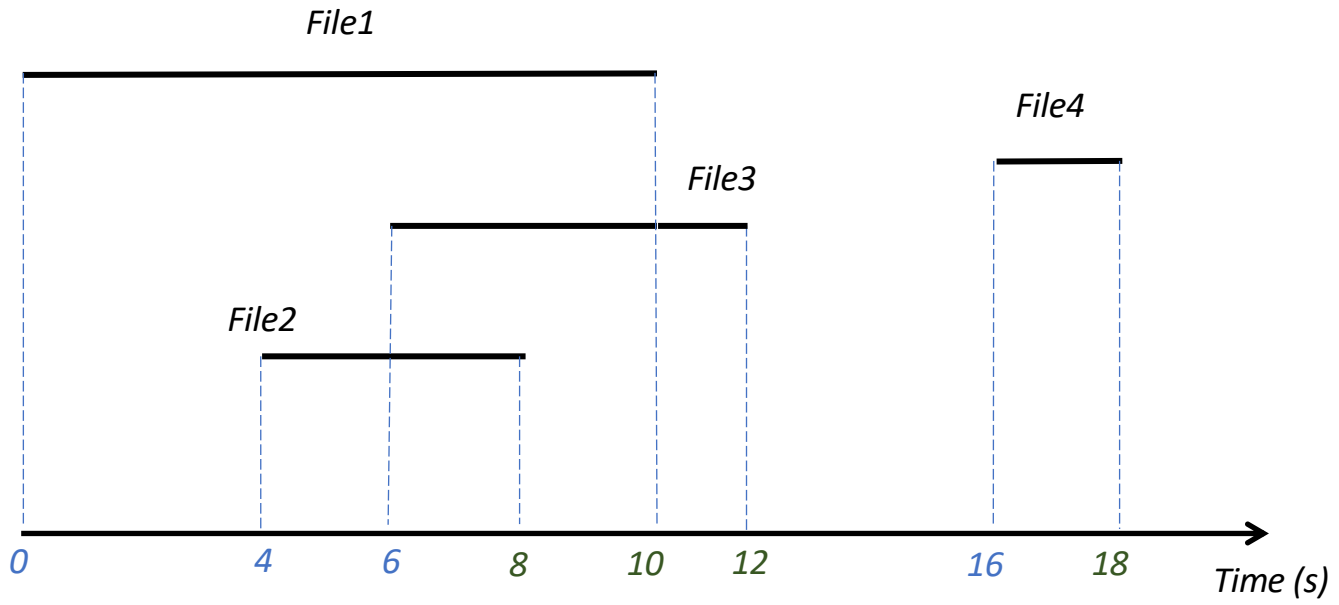
Parallel Coordinates Plot for Application-Level IO Analysis

- Parallel coordinates plot clusters jobs based on their IO metric values, such as data size, small IO (%) and IO BW.
- Each line represents a job, each color represents a user



Root cause analysis: finding the bottleneck files

- Each job can read/write a large number of files, complicating the process of root cause analysis.
- IOMiner automatically filters the bottleneck files that determine the I/O timing on the critical path based on sweep-line algorithm.



Calculating the common contributing factor values of bottleneck files

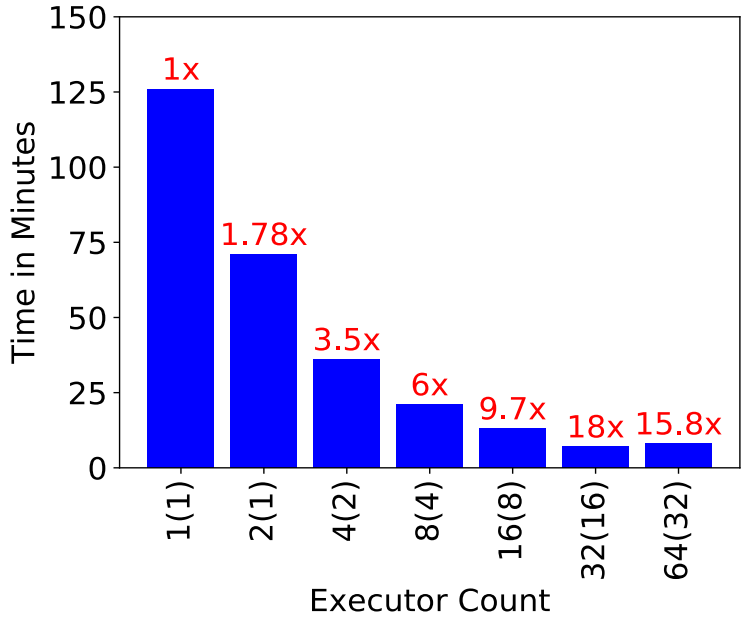
- For each bottleneck file, calculate the following metrics.
- Small I/O percent: percent of small I/O requests among all I/O requests
  - High number of small I/O operations often cause long I/O time.
- Non-consecutive I/O percent: I/O requests not requesting consecutive bytes streams
  - Non-consecutive I/O operations can increase I/O randomness
- Contention level: the ratio of the process count accessing a file to the number of OSTs hosting that file.
  - A high value is a possible indicator that a large number of processes are writing/reading the same OST.
- Whether or not collective I/O is used.
  - Collective I/O reduces the number of small and random I/O.

Experiment Setup

- Hardware Configuration
  - Cori supercomputer from LBNL with 2400 Intel “Haswell” nodes and 9700 Intel Xeon Phi “Knights Landing” nodes.
  - A center-wide Lustre file system.
- Coverage
  - 0.45 million Darshan logs from 3.15 million successfully completed jobs during Jan 2018.
  - Analyzed Darshan logs cover 9.4% of read/write traffic on LMT, and 20.2% of CPU hours from Slurm
- Analysis performed
  - System-wide statistics such as sequential I/O ratio, dominating I/O patterns.
  - Analysis of root cause for applications’ low I/O performance.

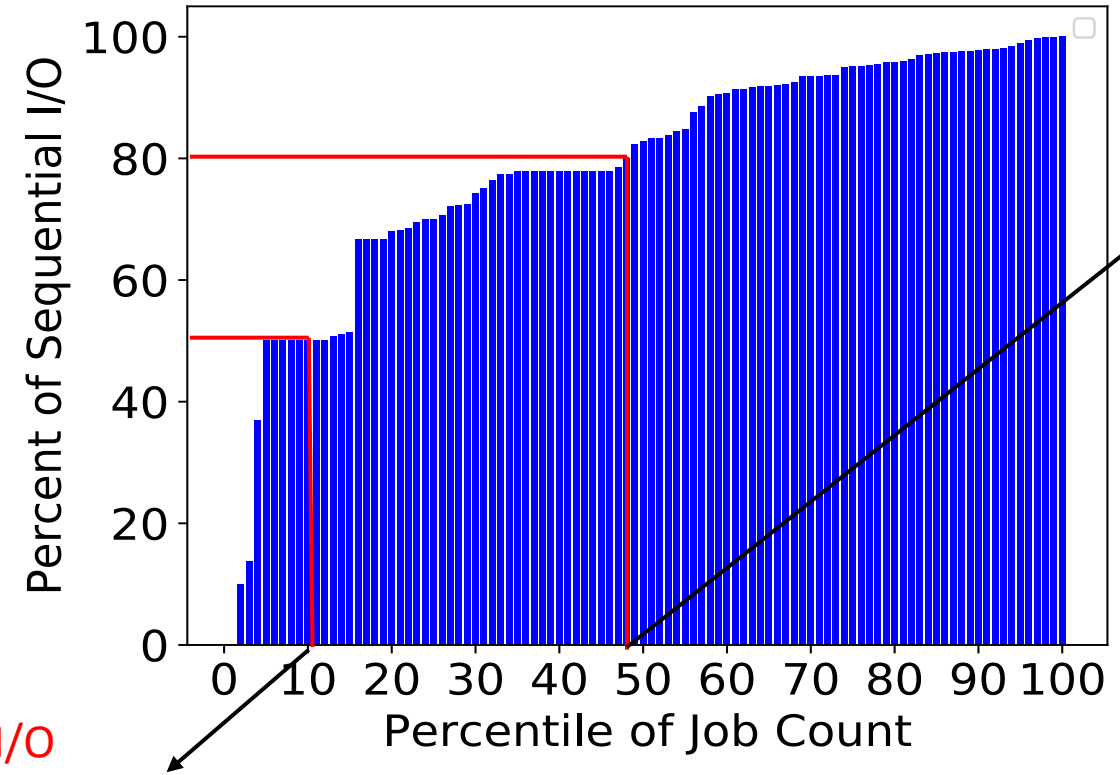
Performance of IOMiner

- Measure the time of scanning all Darshan subtables to find out the jobs with customized stripe setting.
- IOMiner achieves up to 18x speedup.
- Only 0.08% jobs use customized stripe setting.



How many jobs adopt sequential I/O pattern?

- More than 50% jobs' sequential I/O ratio is above 80%.
- Only 10% jobs' sequential I/O ratio is below 50%.

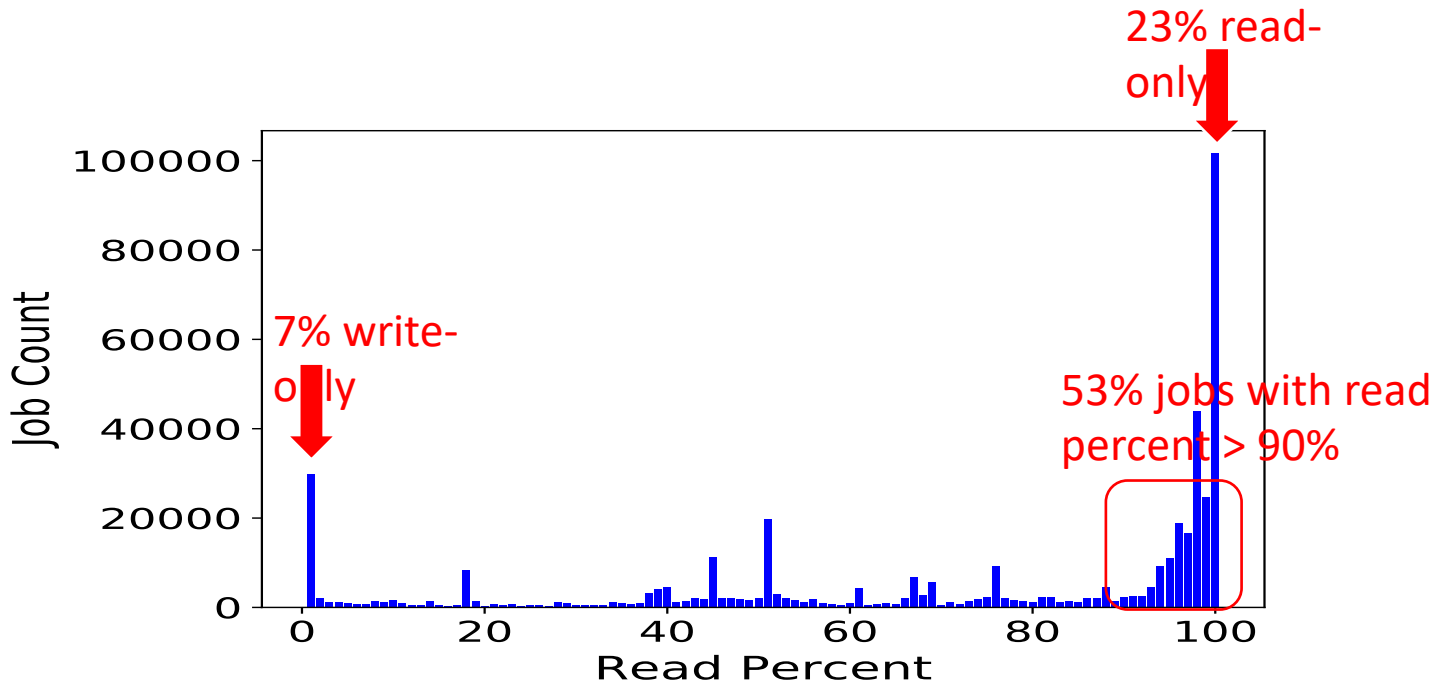


Less than 50% jobs' sequential I/O percent is below 80%

10% jobs' sequential I/O percent is below 50%.

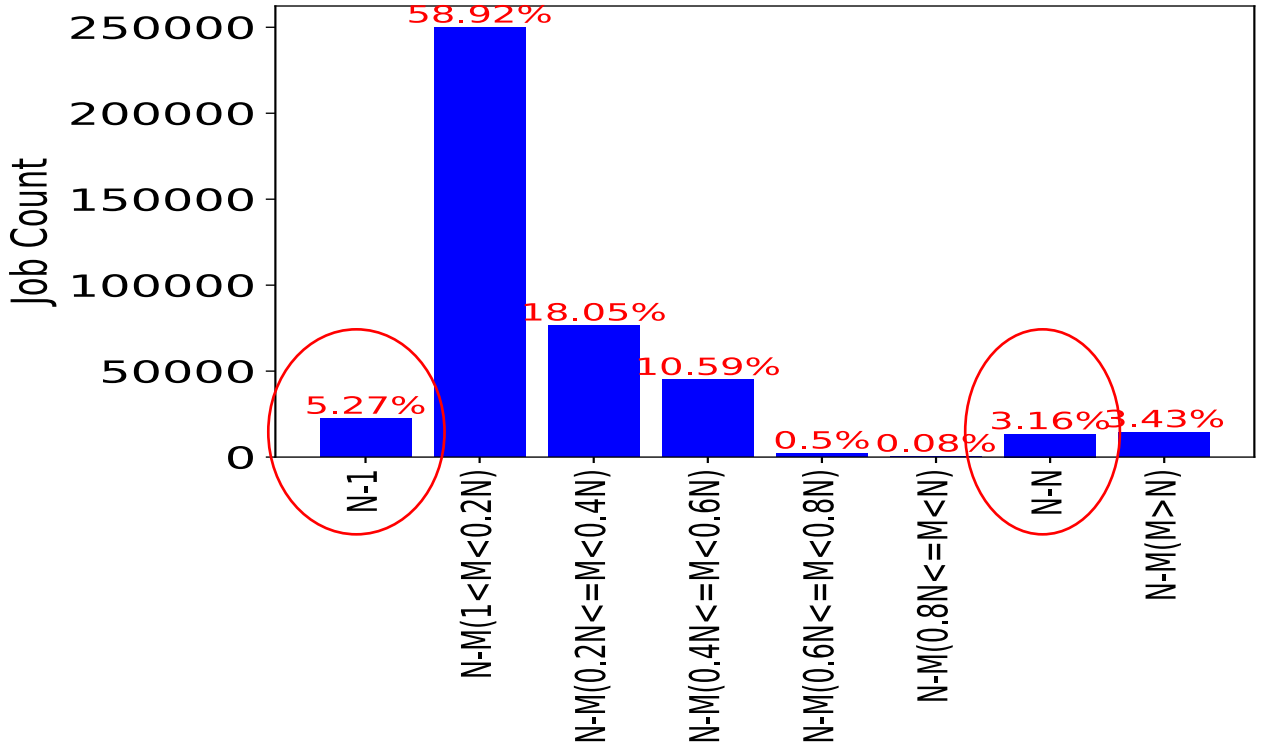
How many jobs are read-intensive?

- Read percent is measured by (bytes_read/total I/O size) (53%)
- 7% and 23% jobs are read-only and write-only, 53% jobs' read percent are above 90%.
- Improving read performance carries at least same consequence as write.



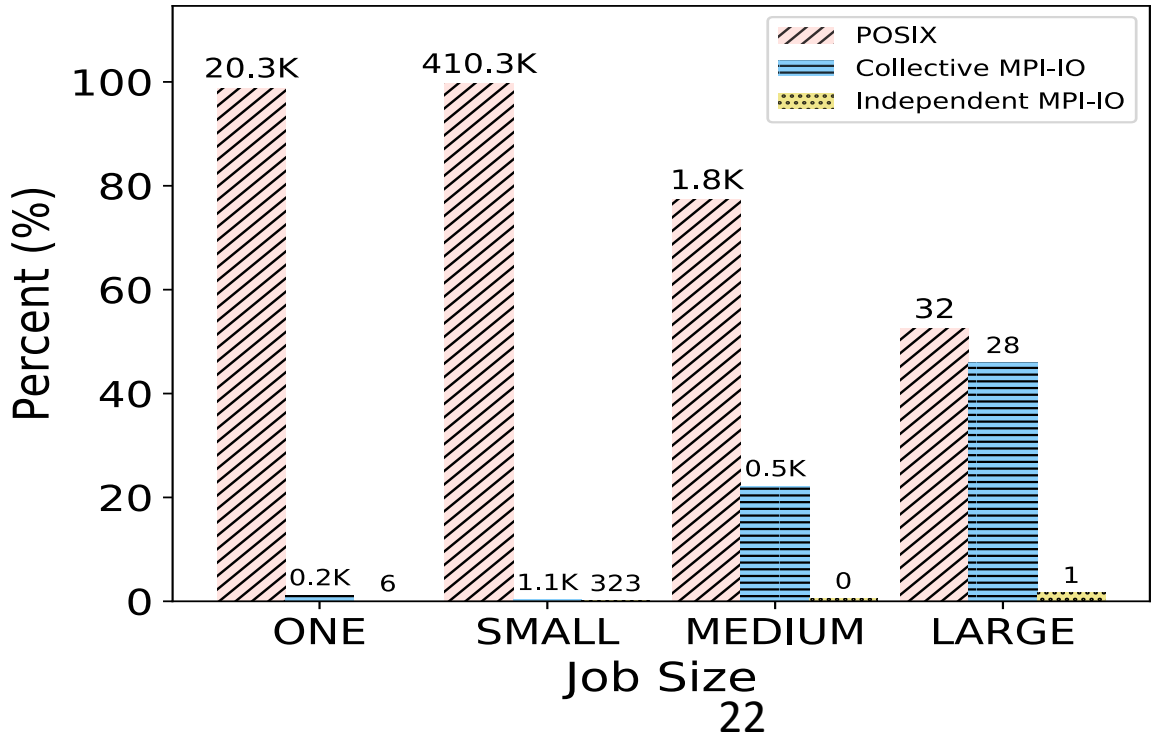
Are N-N and N-1 the most common I/O patterns?

- N: the number of processes. M: the number of files.
- N-1 and N-N jobs are only a small fraction of all the jobs, most jobs exhibit N-M patterns.



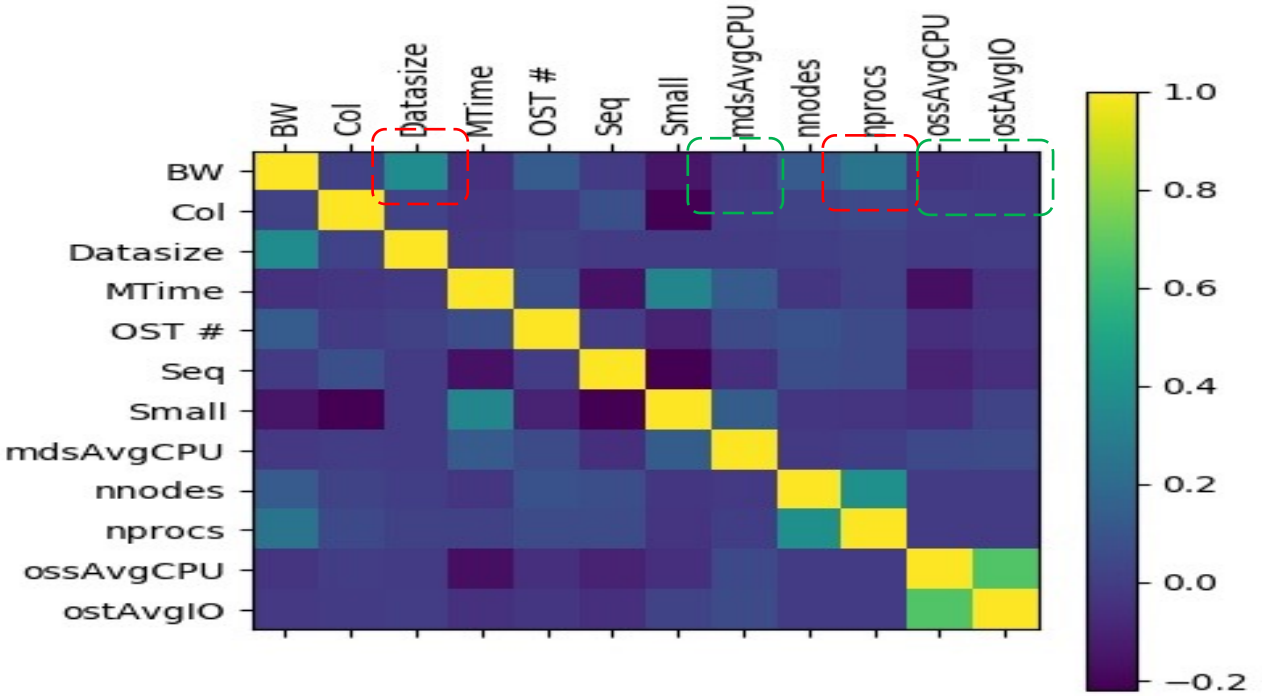
Is MPI-IO widely adopted?

- ONE: jobs using 1 process. SMALL:[2,1024]. MEDIUM:[1025,8192], LARGE:>8192
- POSIX I/O is still dominating all the cases. Collective MPI-IO is mostly enabled in MPI-IO.



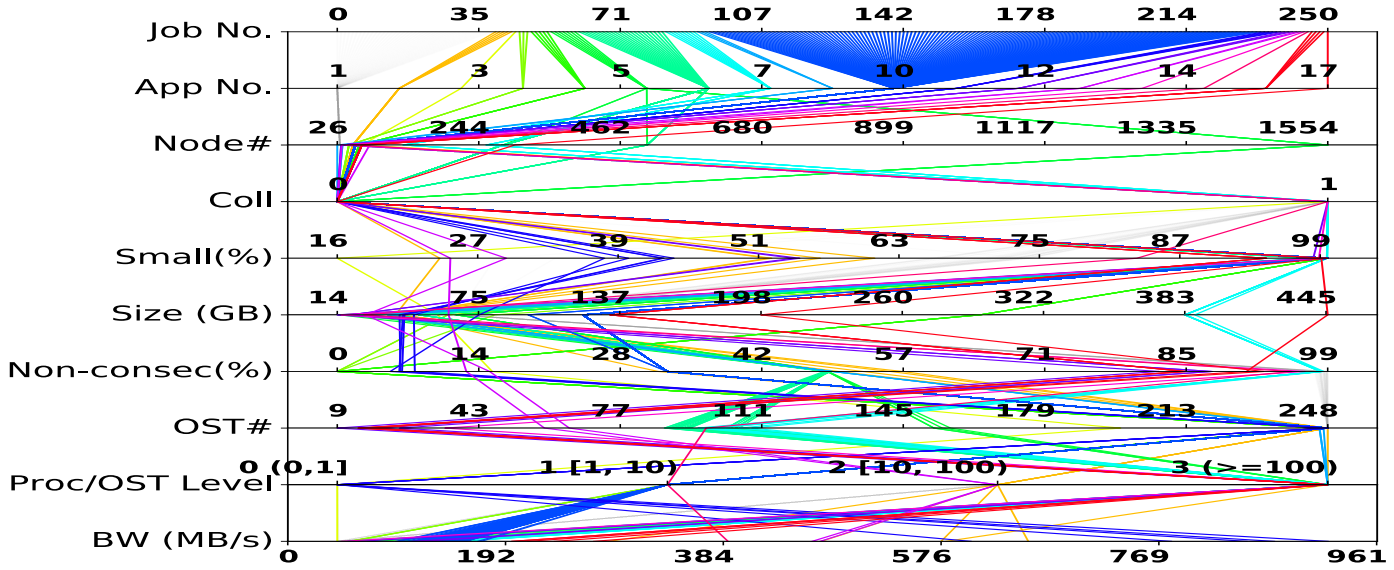
How factors correlate with each other and impact I/O bandwidth?

- The max correlation of job-level factors with bandwidth is data size (0.38) and process count (0.25) in red box
- Correlation between system-level factors and bandwidth is negligible (between -0.05 and 0.05) in green box.
- *No factor alone has dominating platform-wide IO impact due to the diversity of jobs' IO profile, which mandates application-level analysis.*



Analyzing the root causes for applications' poor I/O performance

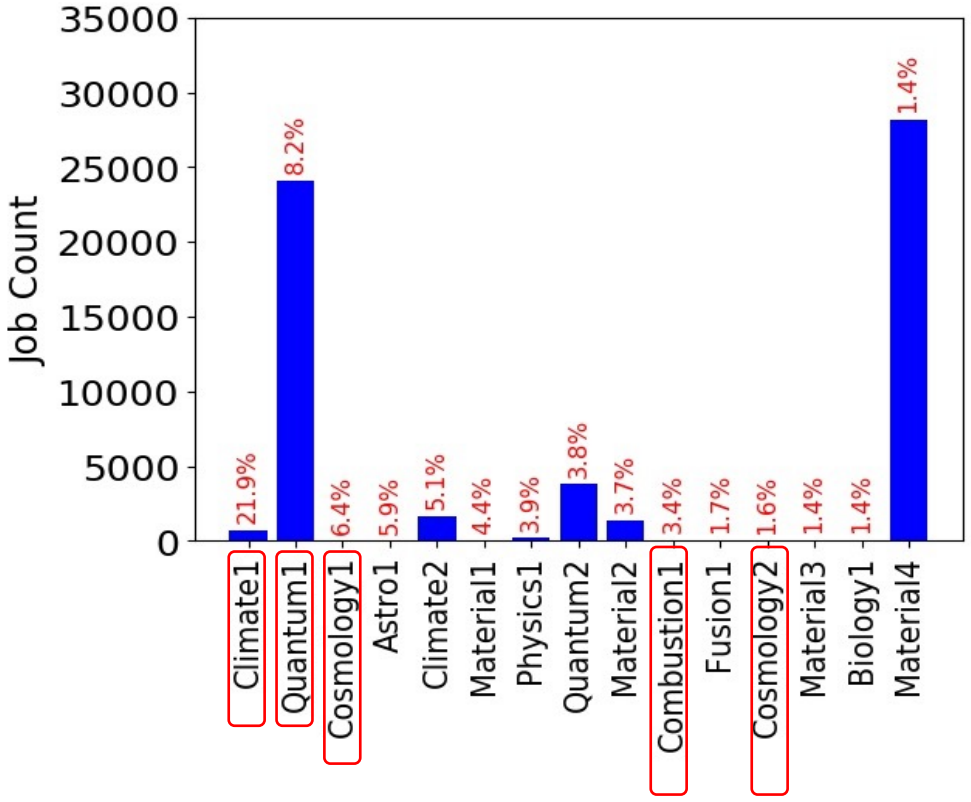
- Each line represents a job, and has values on multiple metrics. The bottom metric is bandwidth. Jobs within the same applications have the same colors.
- There are no single contributing factor that can explain most jobs' poor I/O bandwidth.



Jobs with > 1000 processes, write > 10GB, but read bandwidth below 1GB/s.

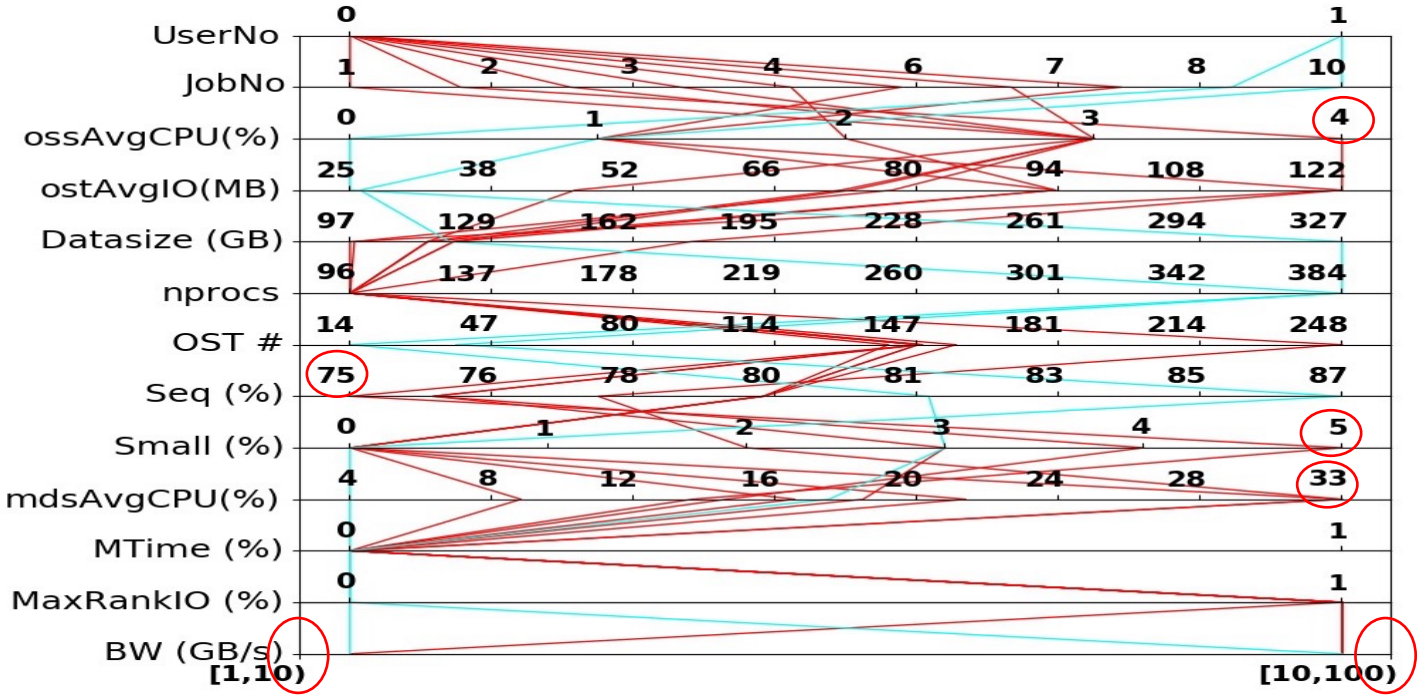
Top CPU core-hour consumer applications

- Group applications by executable name and calculate the aggregate core hours of each application.
- The top 15 applications consume 74% of total CPU core hours of 88,000 jobs.



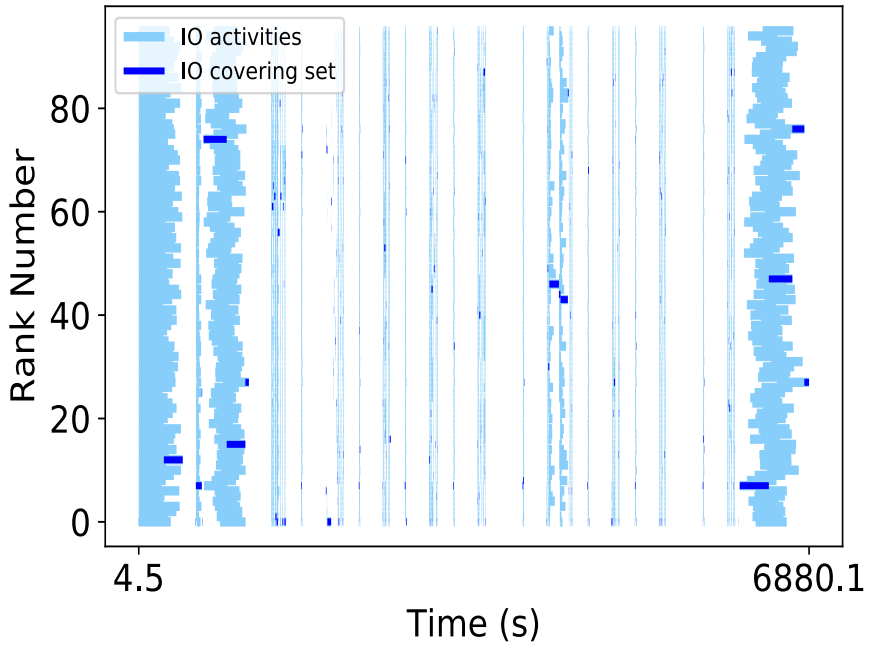
Application-Level Analysis of Cosmology1 IO

- Cosmology1's IO is well-formed: all jobs have high sequential IO (>75%), low small IO ratio (< 5%). Low metadata and storage server CPU utilization (<4% and <33%), etc.
- *However, IO bandwidth varies between [1,10) GB/s and [10,100)GB/s, which needs a job-level analysis.*

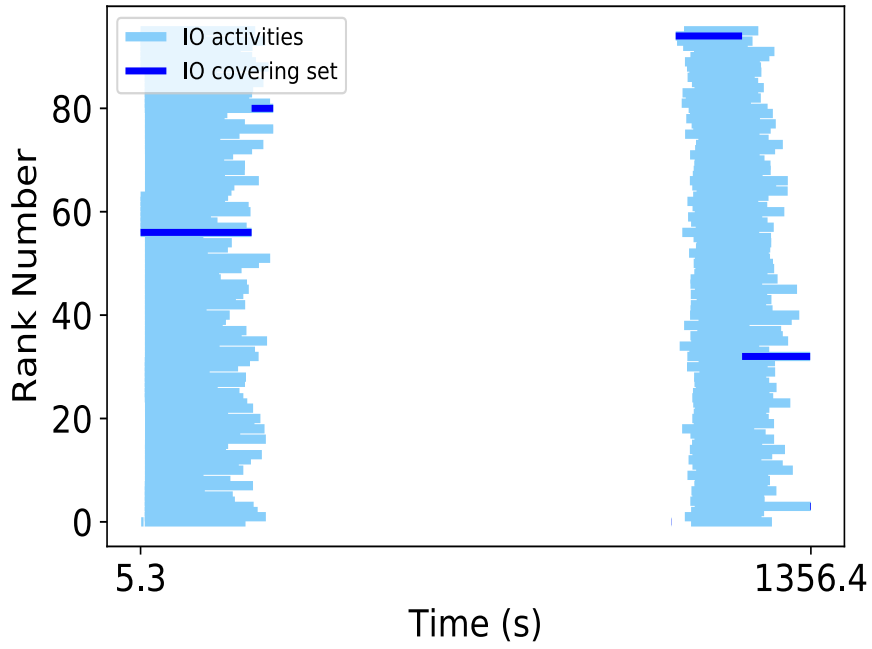


Job-Level Analysis of Cosmology1 IO

- Jobs with [1,10) GB/s is due to its frequent I/O phases, I/O time of each I/O phase is bottlenecked by the slowest rank (a rank is a process).
- Refer to paper for other [1,10) GB/s and [10,100) GB/s cases.



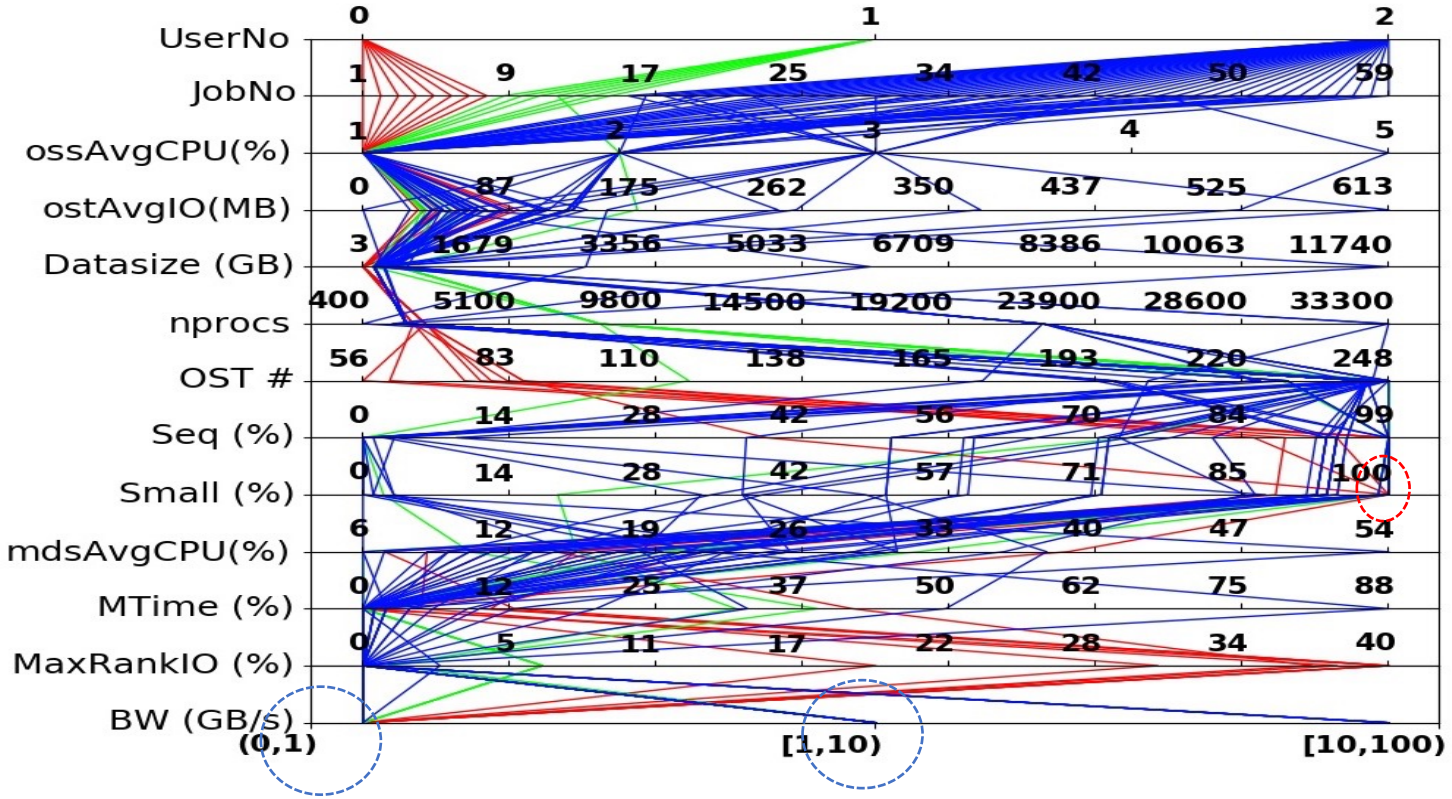
(a) Job with [1, 10) GB/s



(b) Job with [10, 100) GB/s

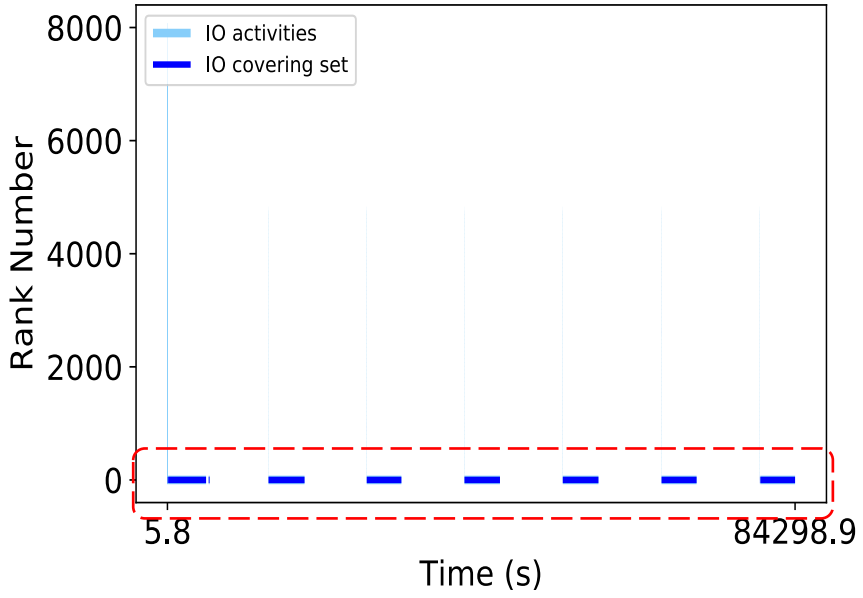
Application-Level Analysis of Combustion1 IO

- There are 59 jobs belonging to 3 users.
- User 0's jobs (red) are all bottlenecked by too many small I/O (100%).
- User 1's jobs (green) fall in both [0, 1) GB/s and [1,10) GB/s.

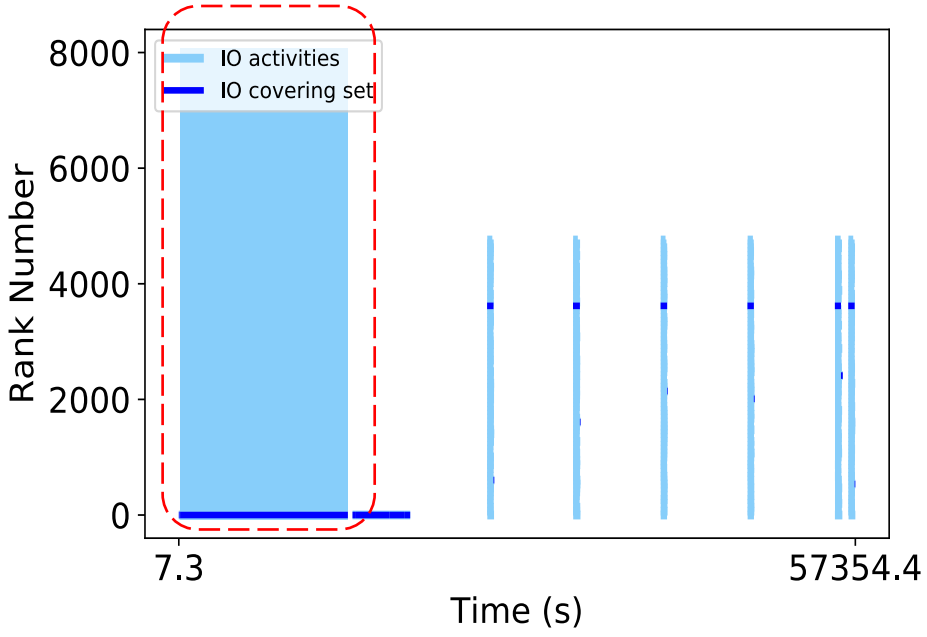


Job-Level Analysis of Combustion1 IO

- Zoom-in one representative User 1's job in each bandwidth category.
- [0, 1)GB/s job is bottlenecked by rank 0 undertaking excessively higher IO workload.
- [1, 10) GB/s jobs is bottlenecked by all ranks reading a shared file, but Darshan log indicates only one rank performs actual read.



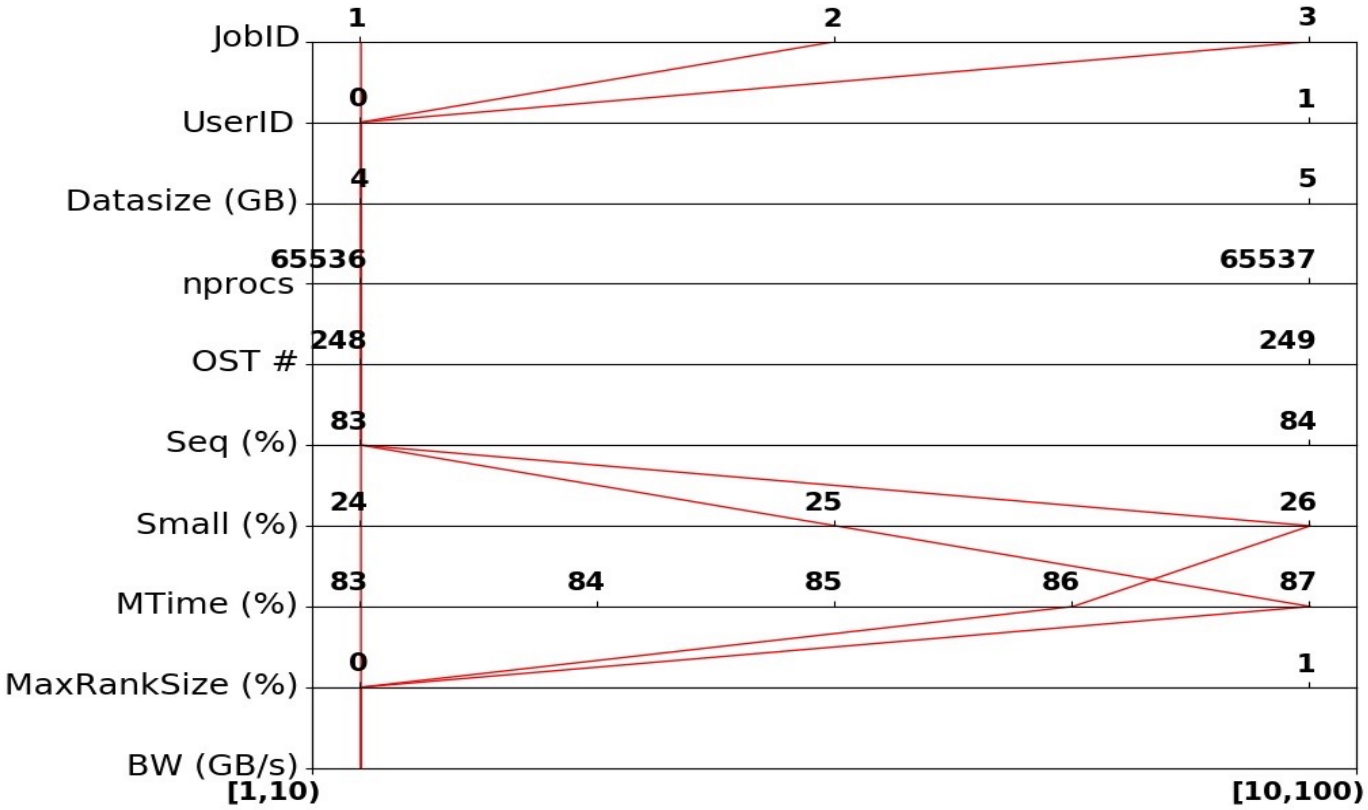
(a) Job with [0, 1) GB/s



(b) Job with [1, 10) GB/s

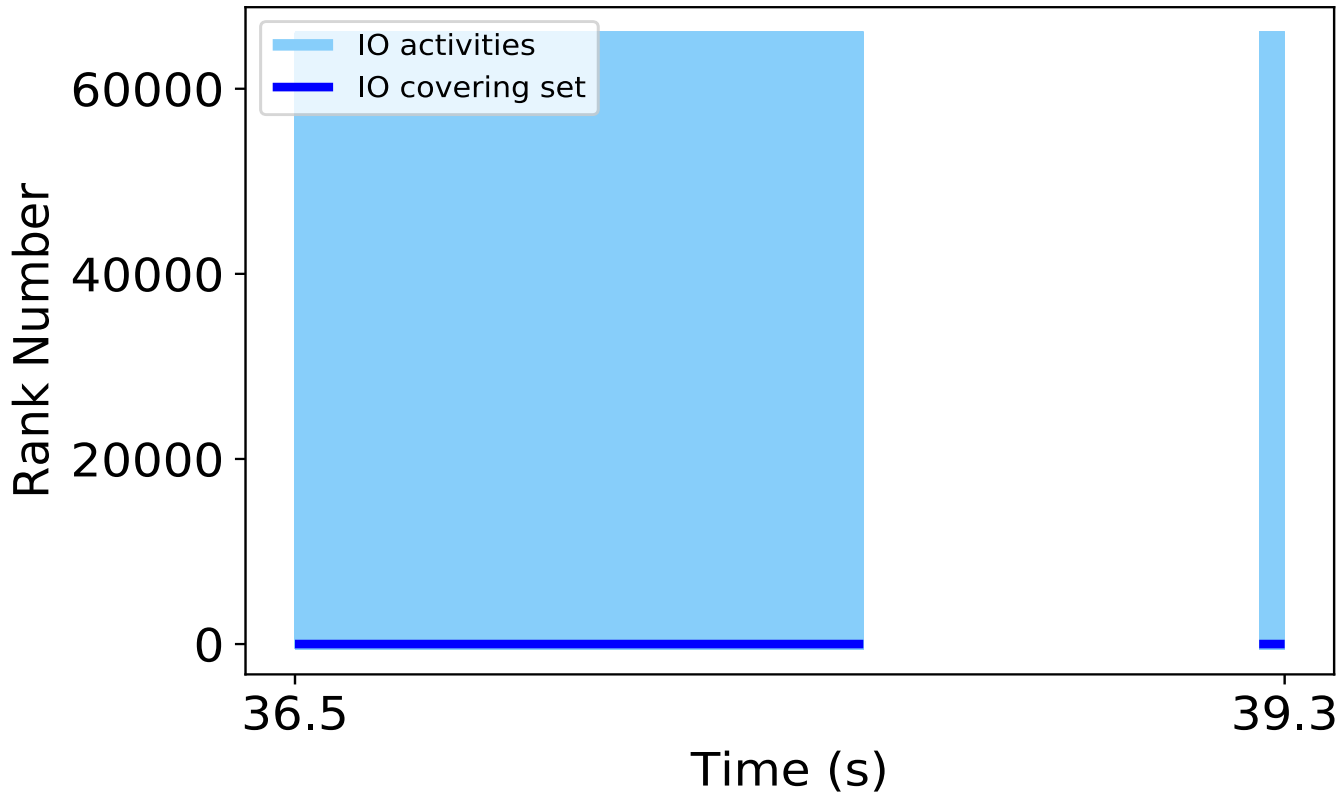
Application-Level Analysis of Cosmology2 IO

- 3 jobs were run by the same user.
- All three jobs' I/O bandwidth is within [1, 10)GB/s



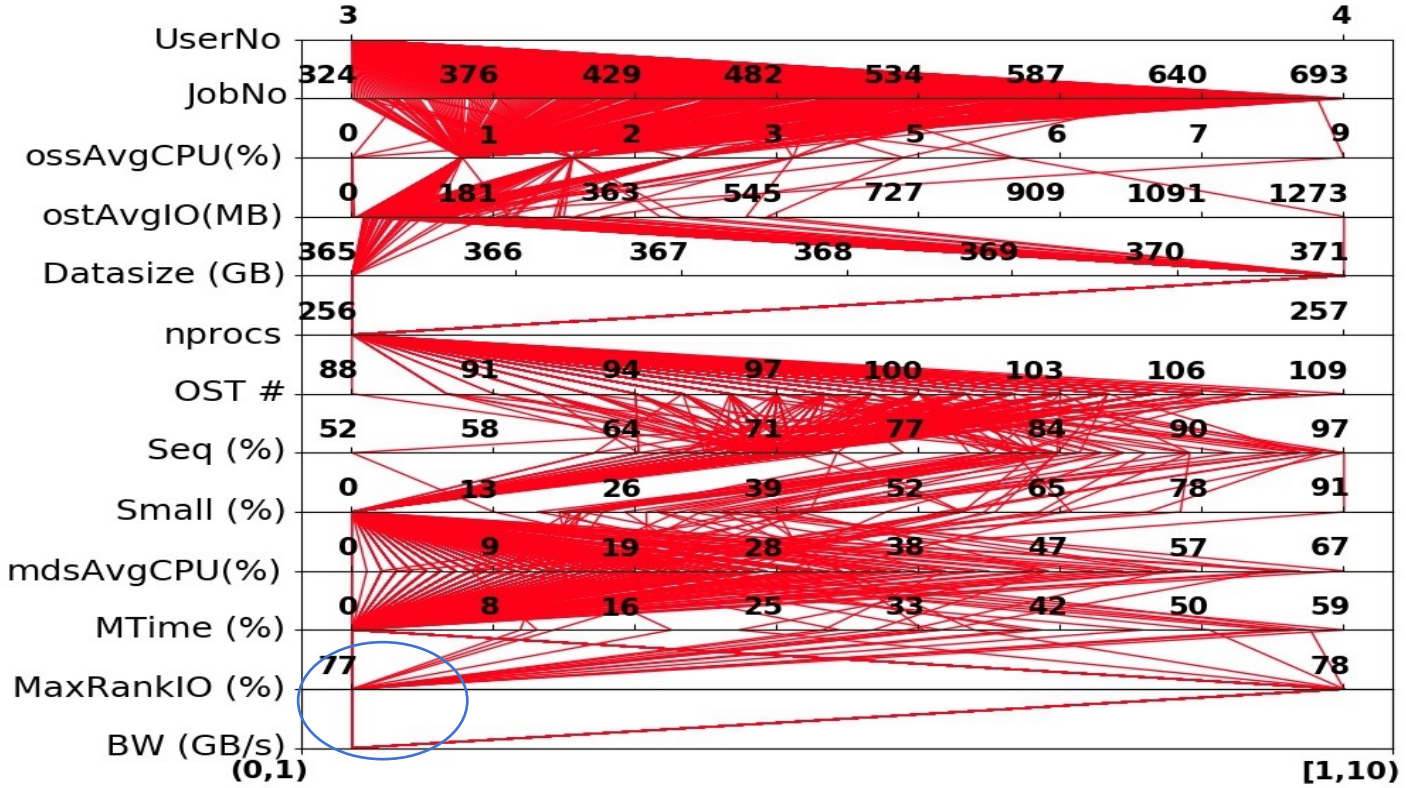
Job-Level Analysis of Cosmology2 IO

- All the processes concurrently access a shared file
- *However, this file is striped on only one OST (default stripe count 1)*



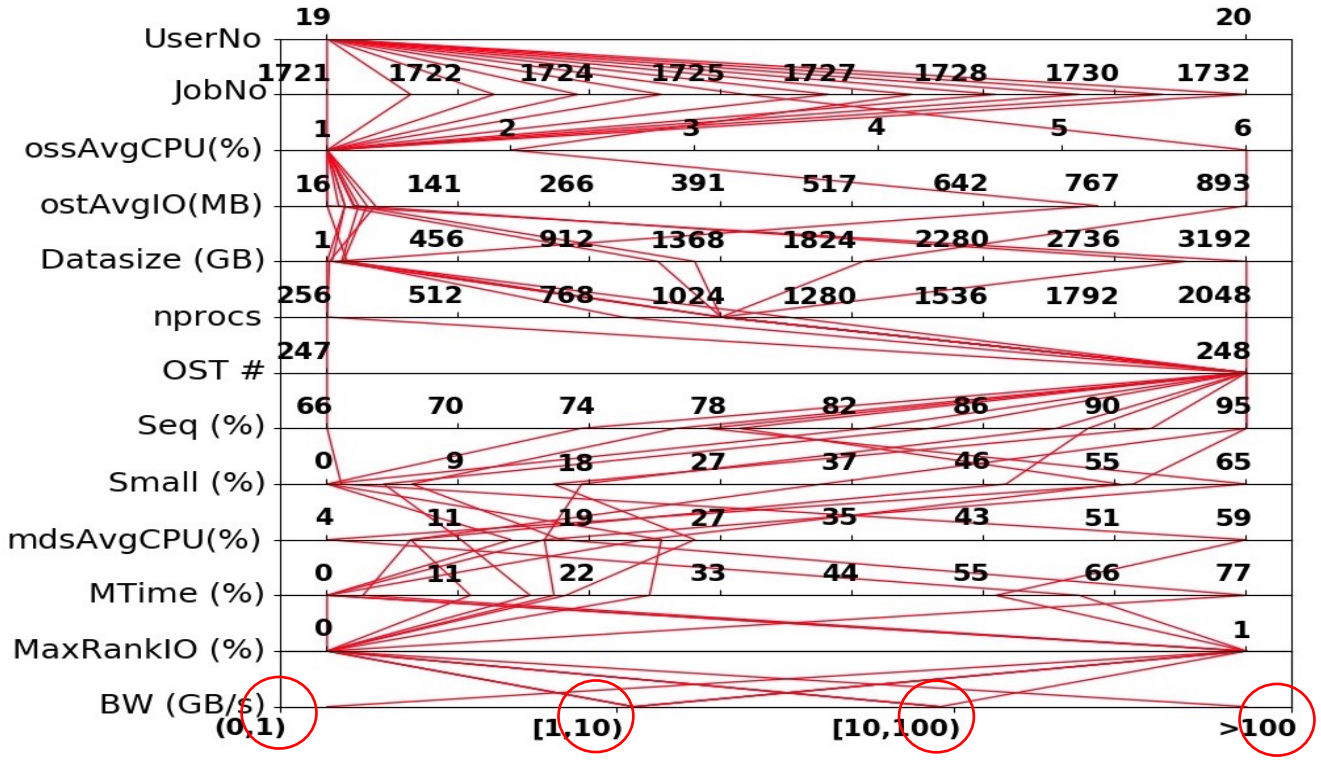
Job-Level Analysis of Climate1 IO

- MaxRankIO: the percent of data accessed by the rank with max IO.
- *All the jobs' bandwidth are bottlenecked by one rank performing most of IO workload.*

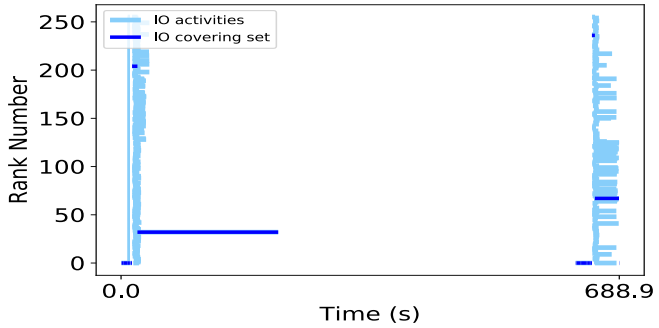


Application-Level Analysis of Quantum1 IO

- One user's job's I/O bandwidth spans across 4 ranges, and the I/O bottlenecks of each range is not directly perceivable from the parallel coordinate plot.

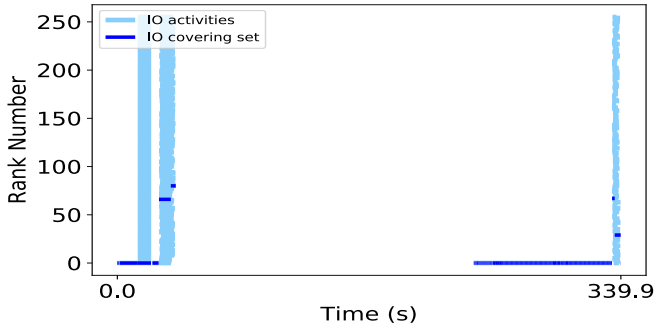


Job-Level Analysis of Quantum1 IO



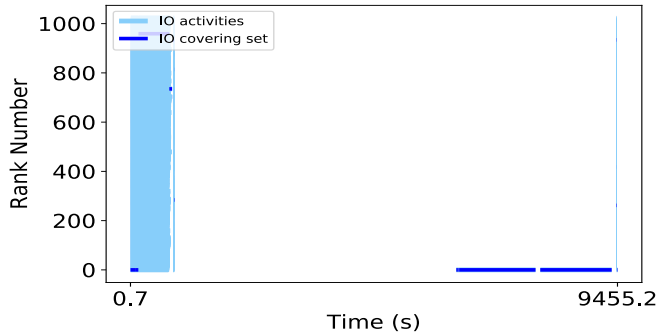
(0, 1]GB/s job

One rank IO suffers from transient metadata load change



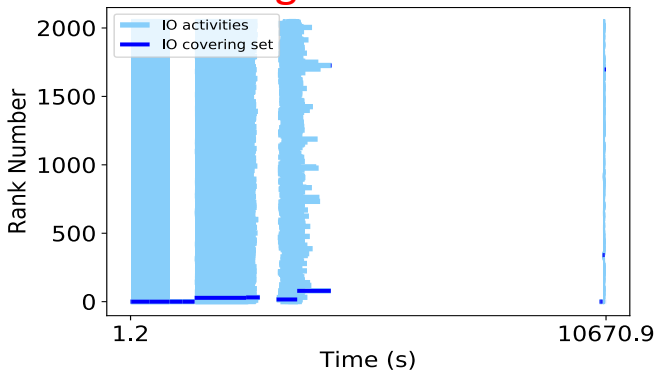
(1, 10]GB/s job

Rank 0 writes a large number of configuration files



(10, 100]GB/s job

Rank 0 takes more time than others due to its larger workload, but bandwidth is still good as other ranks also write a lot of data



(100, 1000]GB/s job

There are four I/O phases, each I/O phase follows read after write pattern

Summary of today's class

- Mining I/O logs for root causes of performance bottlenecks

- References:
  - Teng Wang, Suren Byna, Glenn Lockwood, Nicholas Wright, Phil Carns, and Shane Snyder, "IOMiner: Large-scale Analytics Framework for Gaining Knowledge from I/O Logs", IEEE Cluster 2018 https://sdm.lbl.gov/~sbyna/research/papers/201809_IOMiner_Cluster_2018_Teng.pdf
  - Teng Wang, Suren Byna, Glenn Lockwood, Philip Carns, Shane Snyder, Sunggon Kim, and Nicholas Wright, "A Zoom-in Analysis of I/O Logs to Detect Root Causes of I/O Performance Bottlenecks", IEEE/ACM CCGrid 2019 https://sdm.lbl.gov/~sbyna/research/papers/2019/201905-CCGrid-ZoomIn_IO.pdf

- Next class: Tuning I/O performance automatically