Ed Karrels

NCSA student employee / research assistant

Advisor: William Gropp

NCSA manager: Daniel Katz

Area of research: parallel I/O

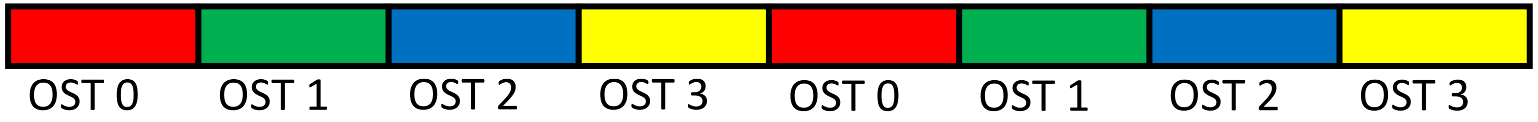
Parallel I/O

- Improving MPI-IO collective routines
  - MPI_File_{read,write}_all
  - Useful with multidimensional arrays
  - Coalesce small reads/writes
  - Parallelize I/O (on parallel file system)

Lustre

- Common HPC parallel file system
- Familiar POSIX API
- Use “striping” to parallelize data accesses
  - Many small files
  - One large file, blocks distributed round-robin

```
lfs setstripe --stripe-size 1M --stripe-count 4 ~/scratch/mydata
```

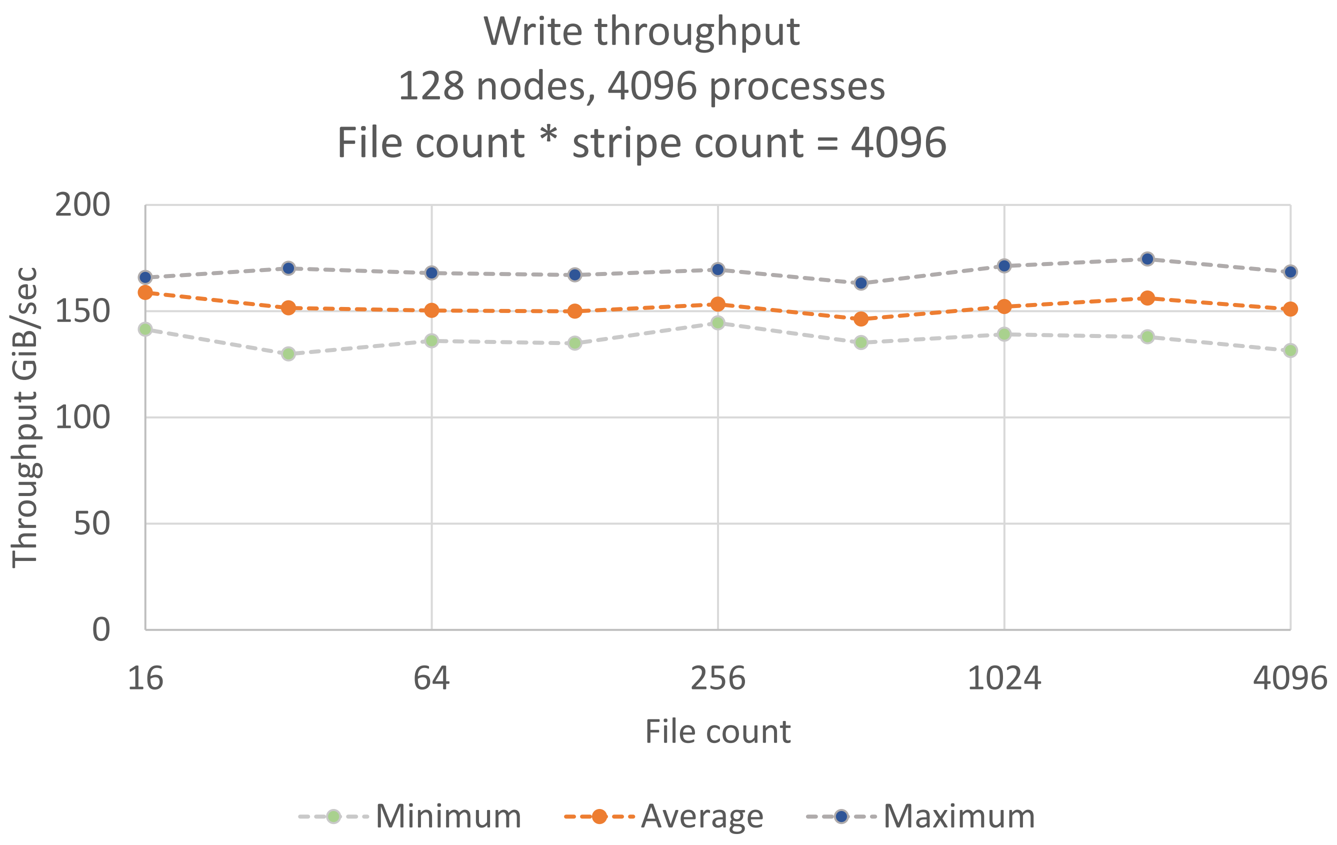


Lustre Throughput on Blue Waters

- One node: 520 MiB/sec
  - Access size >= 256 KiB
- 8k nodes: 1150 GiB/sec
  - 2200x speedup
- No parallelism with single node; need multiple nodes
- Simple solution: one file per process
  - Many files, unwieldy
- Harder: many stripes in one file
  - Difficult to coordinate, different node writes each block
- How about both?

More stripes, fewer files

- N files, each with M stripes = (N*M)-way parallelism



Network congestion with higher node count

Machine learning – shuffled input data

- Many small data in one large file
  - Data file + (offset,length) index
- Train on random permutation of data
- Many seeks → very slow
- Solution: out-of-core shuffle

Out-of-core shuffle

- Similar to “`sort`” command (or “`shuf`”)
    1. Read large blocks into memory
    2. Shuffle in-memory
    3. Write to temp files
    4. Randomly merge temp files
- Overall IO time: 4 * data size
    - No seeks, all streaming

Throughput on HAL

- NFS: poor and counterintuitive performance
  - Write throughput improved with frequent flushes (each 1MB of data)
    - 40 MiB/s → 200 MiB/s
  - Read throughput improved with multiple threads reading at random offsets

Throughput in MiB/sec

# of threads	Access pattern	
	Sequential	Random
1	40.3	111
4	34.9	242

- New SSD-based storage, shuffle still useful?