Got Burst Buffer. Now What?

Early experiences, exciting future possibilities,
and what we need from the system to make it work

Adam Moody

The Salishan Conference on High-Speed Computing
April 26, 2016

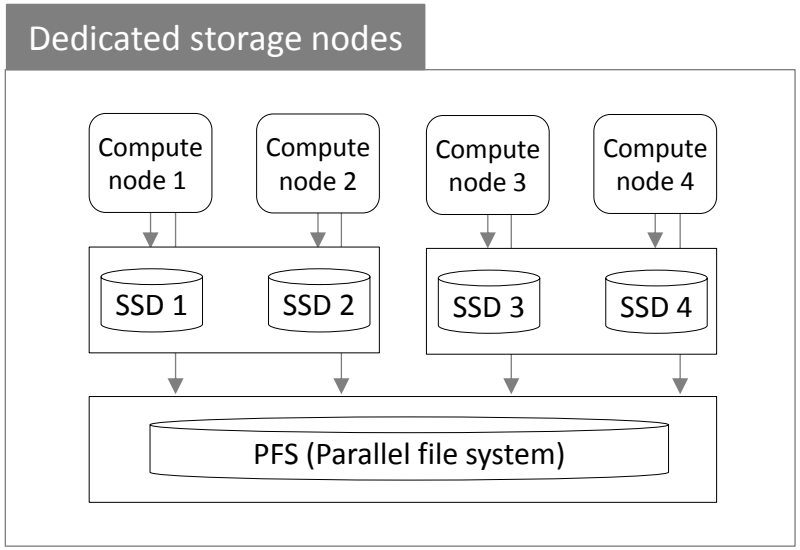
LLNL-PRES-690158

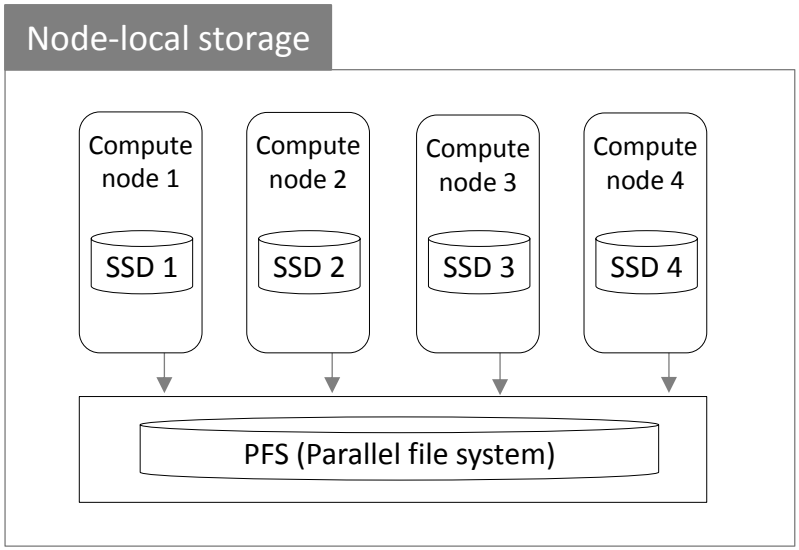
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Burst Buffer 101: quickly absorb write, stream data to parallel file system at slower rate

- Two common designs:
  - Dedicated storage nodes, shared among compute nodes
  - Storage local to each compute node

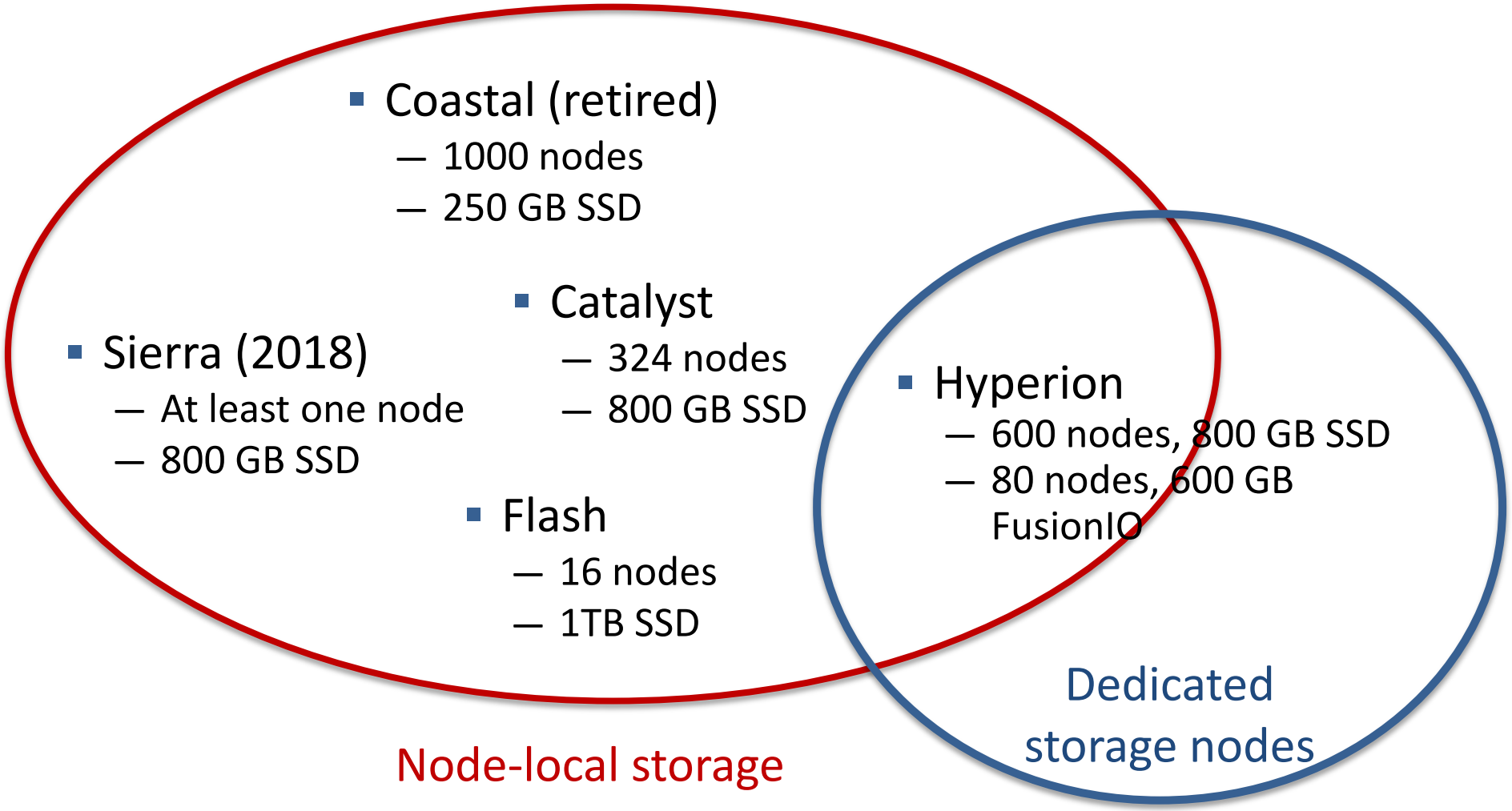




Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, Carlos Maltzahn, "On the Role of Burst Buffers in Leadership-Class Storage Systems", MSST 2012

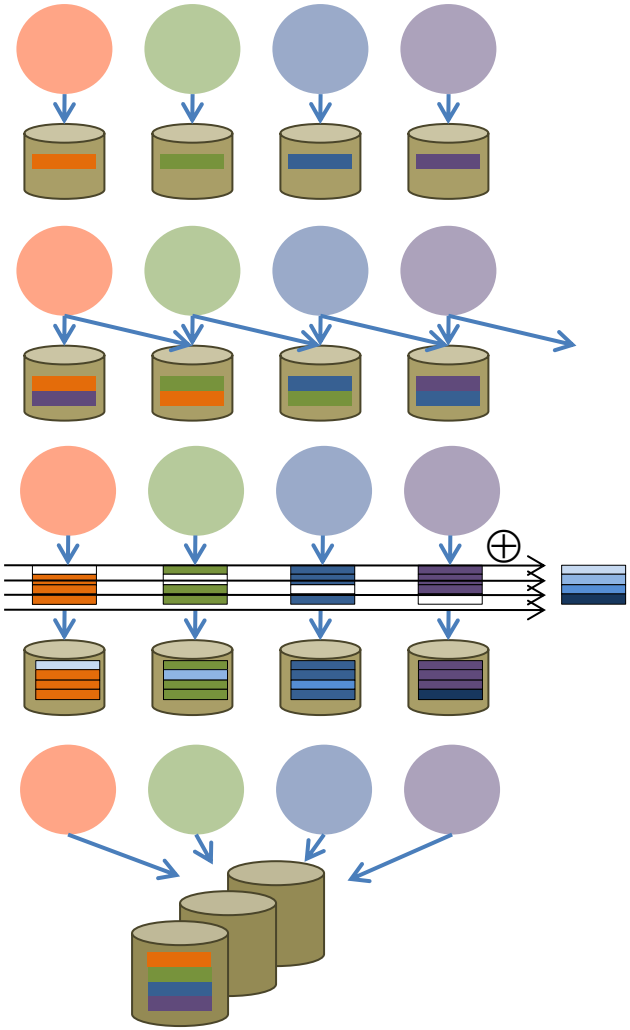
Kento Sato et al, "A User-level InfiniBand-based File System and Checkpoint Strategy for Burst Buffers", CCGrid 2014

LLNL has a history (and future) of machines with node-local storage



Checkpoint / Restart

Checkpoint levels vary in cost and reliability, can recover from most failures using checkpoints 100-1000x faster than PFS

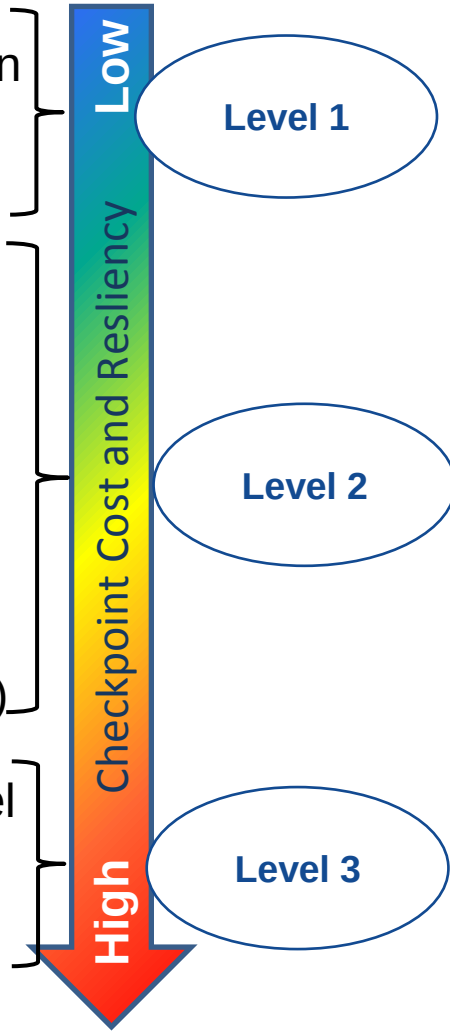


Single: Store checkpoint data on node's local storage, e.g. disk, memory

Partner: Write to local storage and on a partner node

XOR: Write file to local storage and small sets of nodes collectively compute and store parity redundancy data (RAID-5)

Stable Storage: Write to parallel file system



Application developers can use new interfaces to abstract details of burst buffers

- The Scalable Checkpoint/Restart (SCR) library abstracts burst buffer details from application
- But what do I give up?
  - No support for sharing files between processes
  - Cannot update files after declaring them to be complete
  - Can only access checkpoint files during a specific window
- Plans to extend to support general output sets (not just checkpoints)
- Not the only one:
  - **Fault Tolerance Interface** (FTI) from Argonne National Laboratory
  - **Hierarchical Input Output** (HIO) from Los Alamos National Laboratory

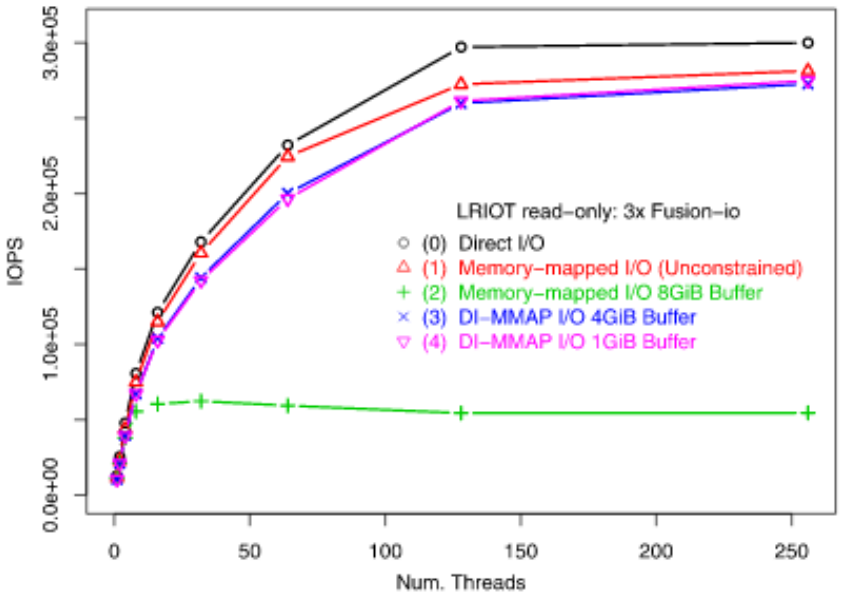
These middleware projects need new support from resource managers and burst buffer APIs

- Need portable way to determine topology and properties of storage
  - File path
  - Unique identifier
  - Capacity
  - Speed
  - Resiliency
- New interfaces considered for PMIx standard may help
- Prestage / poststage
  - Need for scripts to determine which checkpoint to load
  - Need to determine whether final checkpoint was written out
- Keep checkpoint in cache between consecutive jobs, or extend job runtime of current job

Extended Memory

Data Intensive Memory Mapped IO (DI-MAP) provides fast access to non-volatile memory

- With different classes of memory, apps must manage where and when to move data from slow to fast memory.
- This can be explicit or implicit.
- For implicit, mmap is nice.
- Linux memory map runtime:
  - Optimized for shared libraries
  - Does not expect mmap data to churn
  - Does not expect mmap data to exceed memory capacity
- DI-MMAP:
  - Optimized for frequent evictions
  - Optimized for highly concurrent access
  - Expects to churn memory-mapped data

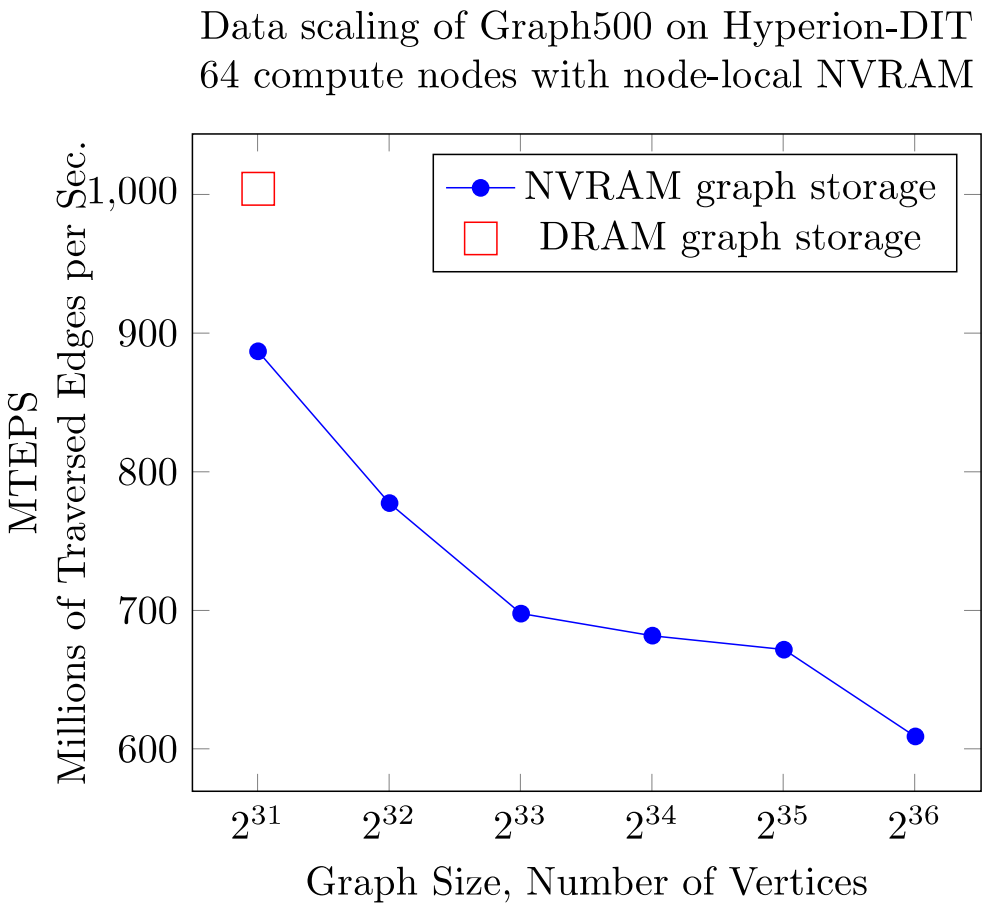


Brian Van Essen, Henry Hsieh, Sasha Ames, Roger Pearce, Maya Gokhale,
"DI-MMAP—a scalable memory-map runtime for out-of-core data-intensive applications", Cluster Computing 2013.

NVRAM augments DRAM to process trillion edge graphs [IPDPS 2013]

Our approach can process 32x larger datasets with only a 39% performance degradation in TEPS by leveraging node-local NVRAM for expanded graph data storage.

By exploiting both distributed memory processing and node-local NVRAM, significantly larger datasets can be processed than with either approach in isolation.

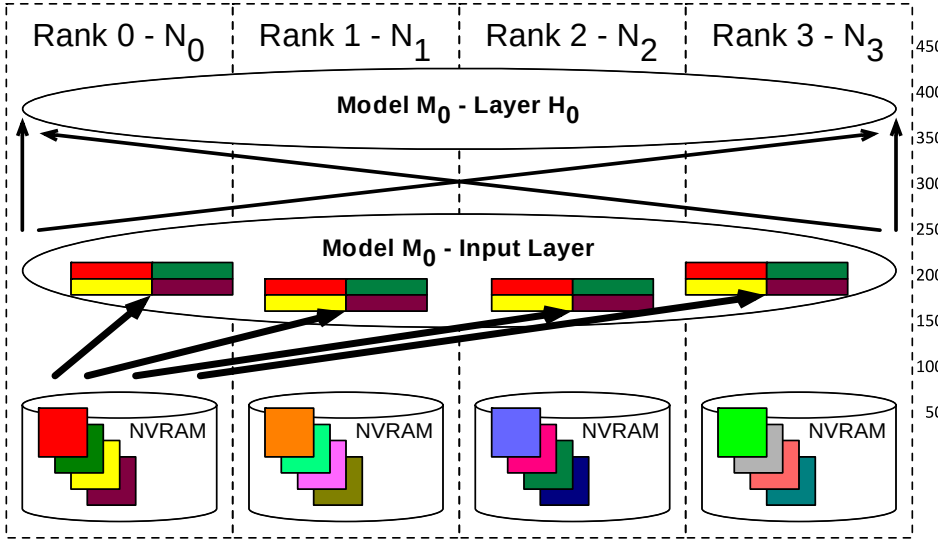


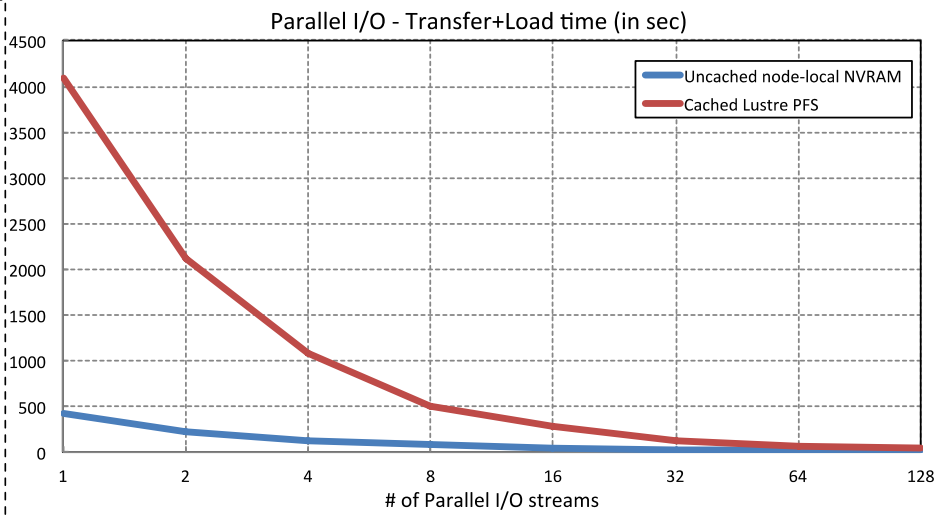
Roger Pearce, Maya Gokhale, Nancy M. Amato,
"Scaling Techniques for Massive Scale-Free Graphs in Distributed (External) Memory", IPDPS 2013

Machine Learning & IO Scheduling

Using storage to improve machine learning

- Training must read large input set many times in random order
- Local storage removes load from parallel file system
- What's needed?
  - Support from resource manager to persist dataset between runs
  - Schedule jobs close to data
  - Distributed file system to shard large datasets





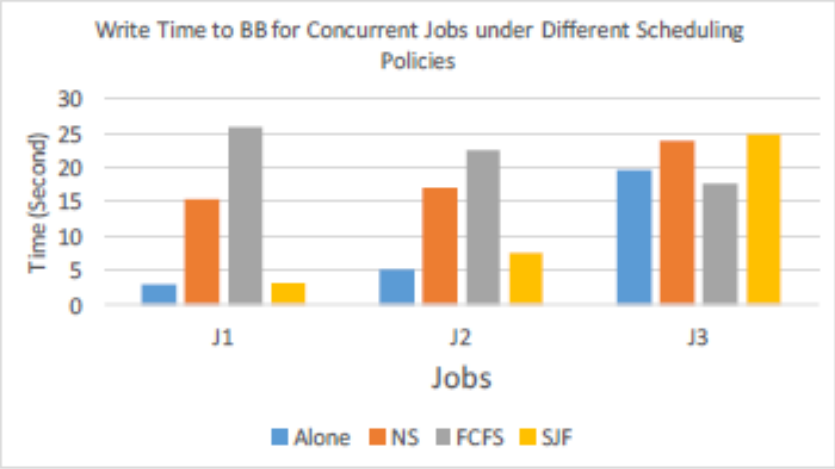
Brian Van Essen et al, "LBANN: Livermore Big Artificial Neural Network HPC Toolkit", MLHPC 2015

Using machine learning to improve storage

- Since early 2015, LLNL has been tracking
  - Aggregate Lustre counters for each job
  - Copy of each job batch script
- For each job waiting in the queue, apply machine learning to features in job script to predict expected run time and expected Lustre stats
- Simulate job scheduler to predict start of each job
- Predict future load on file system
  - Ryan McKenna, Michela Taufer, “Forecasting Storms in Parallel File Systems”, SC 2015 Poster
- Burst buffers and knowledge of job IO requirements enables resource scheduler to control load on file system
  - Stephen Herbein, Michela Taufer, “Scalable IO-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters”, HPDC 2016.
  - See Stephen’s poster here at Salishan tomorrow night!

It's beneficial to coordinate IO between compute nodes and burst buffer nodes

- Which BB nodes should a job write to?
- When should each job write?
- How to control who writes when?

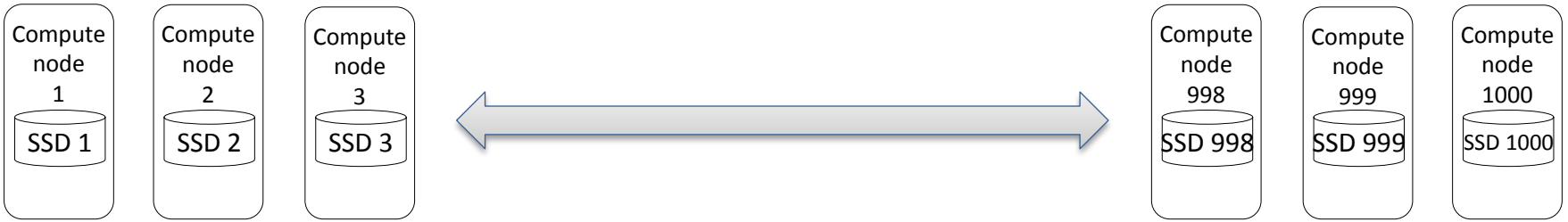


- Work shows that controlling these factors matters
  - Sagar Thapaliya, Jay Lofstead, "Managing I/O Interference in a Shared Burst Buffer System"

User-level file systems

Consider the possibilities of specialized, user-level "file systems" that are setup and torn down with each job

- Enable application to mount more than one user-level file system
  - Specialize each file system for different task
  - Intercept application libc calls (e.g., techniques from Darshan)
  - Intercept file system operations from batch script using FUSE
  - Let app "mount" file systems based on its different IO requirements
- Store data on parallel file system between jobs using pre/post stage if needed
- Precursors: SCR, FTI, PLFS, HIO, CRUISE



What's needed for distributed services?

- Need efficient asynchronous services
  - Where is my MPI_THREAD_MULTIPLE performance?
- Services are processes, too
  - MPI can't hog the CPU or the network
  - Services will need same performance as the MPI job
- Blocking vs polling for incoming data
  - Some HPC networks only provide polling
  - Enable blocking in MPI without losing significant performance
  - Ability to disable blocking from within app (MPI_T)

Recap

Recap: What's needed from system software to utilize burst buffers, node-local storage?

- Resource manager
  - Interface to query storage properties
  - User hooks in pre/post stage, avoid unnecessary data staging
  - Persist dataset between runs
  - Schedule jobs to nodes where data are (like Big Data job schedulers)
  - Track IO counters and batch script for each job
  - Schedule for IO, not just node utilization
  - Need fast launch and wire-up for distributed services
- Memory management
  - Good first step: ensure mmap works well for extended memory
- MPI
  - Enable more asynchronous operations / services
  - Enable other services to use network and CPU (MPI can't be the only one)
  - Enable efficient blocking while waiting for incoming messages
  - Support true communication overlap for large Isend/Irecv