CMSC 714
Lecture 22
Parallel I/O

Alan Sussman and Abhinav Bhatele

Notes

- Midterm exams returned Thursday
- Group Project presentations need to be scheduled
  - Email me with preference for day to present, Dec. 5 or 7
    - If not enough of you volunteer for Dec. 5 I will have have to pick
  - final report due Tuesday, December 12

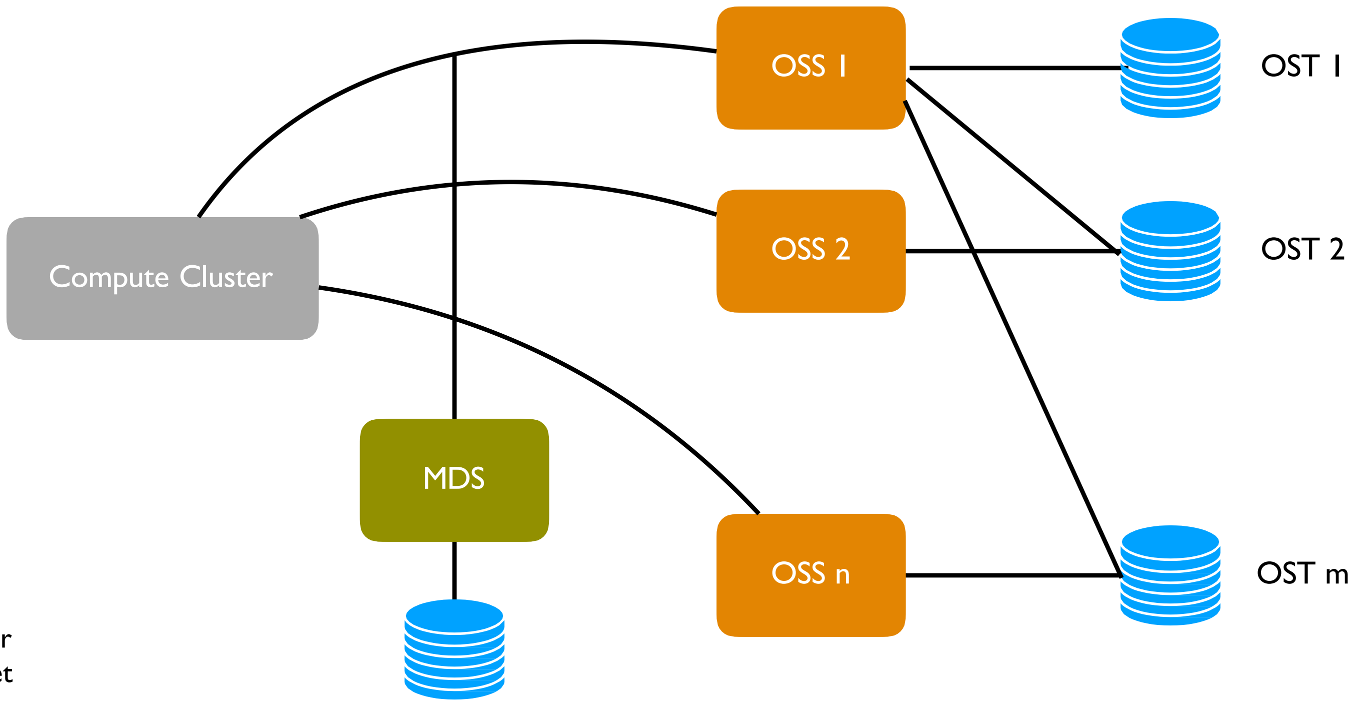
When do parallel programs perform I/O?

- Reading input datasets
- Writing numerical output
- Writing checkpoints

Non-parallel I/O

- Designated process does I/O
- All processes send data to/receive data from that one process
- Not scalable

Parallel filesystem



MDS = Metadata Server
MDT = Metadata Target
OSS = Object Storage Server
OST = Object Storage Target

Different parallel filesystems

- Lustre: open-source (lustre.org)
- BeeGFS: community supported (beegfs.io)
  - Commercial support too
- GPFS: General Parallel File System from IBM, now called Spectrum Scale
- PVFS: Parallel Virtual File System: another open source, but no longer in wide use – a branch called OrangeFS and is still being supported

Tape drive (archive) and burst buffers

- Store copy of data on magnetic tapes for archival purposes
- Burst buffers: fast, intermediate storage between compute nodes and the parallel filesystem
  - Typically some form of flash memory, for persistence, high capacity, and speed (reads and writes)
- Two designs:
  - Node-local burst buffer – e.g., Frontier, Summit @ ORNL
  - Remote (shared) burst buffer – e.g., Cori @ LBL

I/O libraries

- High-level libraries: HDF5, NetCDF
  - Self-describing data formats w/associated libraries
  - Metadata stored with the data
- Middleware: MPI-IO
  - MPI-like I/O interface for collective I/O
- Low-level: POSIX IO
  - Standard Unix/Linux I/O interface

Different I/O patterns

- One process reading/writing all the data
- Multiple processes reading/writing data from/to shared file
- Multiple processes reading/writing data from/to different files
- Different performance depending upon number of readers/writers, file sizes, filesystem, etc.

IBM GPFS

- Designed to support high throughput parallel applications, including multimedia
  - well suited for scientific computations
  - still used in some of Top 500 supercomputers
- Main idea is to use parallel I/O to increase performance and scale to large configurations
  - increase bandwidth by spreading reads and writes (even to a single file) across multiple disks, especially for sequential access
  - avoid the "one file per parallel process" model, or sending all I/O through one node
  - use internal high performance switch, plus separate I/O nodes, for I/O from parallel processes running on compute nodes
  - files can be both striped across multiple I/O nodes, and across multiple disks in each I/O node

IBM GPFS

- Each node runs a demon (mmfsd) to provide I/O services
  - one demon runs a *metanode service*, to serve file metadata (ownership, permissions), and inode/directory updates
  - one demon runs a *stripe group manager*, to keep track of available disks
  - a *token manager* to synchronize concurrent access to files, maintain consistency across caches
  - each application node demon mounts a file system and performs file accesses (through switch, to I/O nodes that have the disks with the data)
- Client-side caching
  - inside Virtual Shared Disk (VSD) layer in kernel (server is on I/O nodes)
  - *pagepool* in each application node's memory
  - read-ahead discovers sequential and constant stride access patterns
  - write behind allows application to continue after data copied into pagepool – cost is extra copy to pagepool
- Experiments show that GPFS scales well to very high absolute performance for sequential accesses
  - need big transfer sizes for non-sequential accesses to get decent performance – use MPI-IO to aggregate (collective I/O)
  - 1 server can handle up to maybe 6 clients – this is technology dependent (switch, disks, processors)

Burst Buffers

- Intermediate storage layer between compute nodes and disk (parallel file system)
  - Slower, but higher capacity, than on-node memory (DRAM)
  - Faster, but lower capacity, than disk storage on file system
- On this system (LBL NERSC Cori), burst buffer is on I/O nodes connected to same network as compute nodes (a “dragonfly” Cray Aries network)
  - As SSDs, and with a POSIX filesystem interface
  - But only available to compute nodes using it, for a limited time – but can persist across multiple jobs
  - 144 I/O nodes with burst buffers for >1660 compute nodes, and a 27PB Lustre parallel file system
  - Burst buffer resources allocated via SLURM, using DataWarp services
    - Striped across nodes, or in “private” mode
    - Looks like a separate file system

Burst Buffers – Use Cases

- Original target is high bandwidth checkpoint-restart
- But several other scenarios at NERSC
  - Complex I/O patterns with high IOPs – e.g., non-sequential table lookups
  - Out-of-core applications
  - Workflows – to couple multiple applications – e.g., store data between simulation and analysis components, or for analysis/visualization (in-situ, in-transit, or interactive)
- Results show burst buffer performs at least well as Lustre, and scales better
  - Both in IOPS, and I/O bandwidth
    - But work still needed on writes to shared file with MPI-IO
  - And the science use cases show the complexity of getting best performance for applications in using burst buffers