SCALABLE COMPUTING LABORATORY

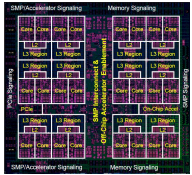
VANDERBILT UNIVERSITY

SC3260 / SC5260

Parallel Filesystem

Lecture by: Ana Gainaru

Multi-core CPU

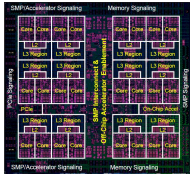


- **Shared memory programming model**
  - Cilk, OpenMP, multi threading
  - Threads are executed independently
- Multi-level memory hierarchy
  - Threads access memory independently
  - Consecutive data fetched by each thread request

VANDERBILT UNIVERSITY

Recap

Multi-core CPU



- **Shared memory programming model**
  - Cilk, OpenMP, multi threading
  - Threads are executed independently
- Multi-level memory hierarchy
  - Threads access memory independently
  - Consecutive data fetched by each thread request

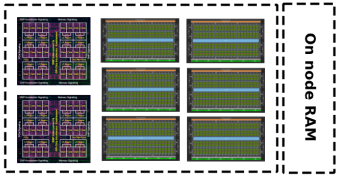
GPU

- **GPU programming model**
  - Cudal, OpenCL, Kokkos
  - Threads are executed grouped in warps
- Grouped memory access
  - Data for all threads in one warp is fetched in one access for each instruction

Example on ACCRE: Nvidia Pascal GeForce and Intel Xeon Westmere

VANDERBILT UNIVERSITY

Nodes

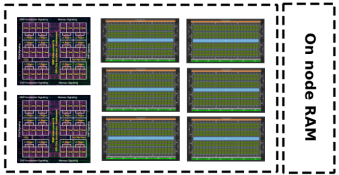


- **Hybrid CPU/GPU code**
  - Use CPU for tasks with complex logic
  - Use GPU for SIMD type of tasks
- Memory transfer between CPU/GPU explicit
  - Overlap computation/communication if possible

VANDERBILT UNIVERSITY

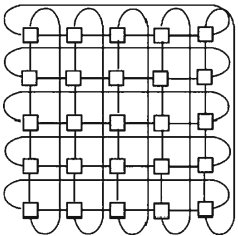
Recap

Nodes



- **Hybrid CPU/GPU code**
  - Use CPU for tasks with complex logic
  - Use GPU for SIMD type of tasks
- Memory transfer between CPU/GPU explicit
  - Overlap computation/communication if possible

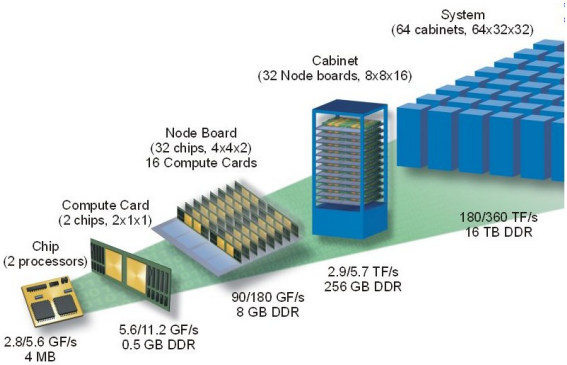
Multi-node system



- **Distributed system**
  - MPI programming language
  - Multiple nodes are connected through links (network)
  - Shared memory/GPU programming inside one node
- Memory transfer is done explicit
  - Processes have only access to their own memory
  - Transfer using messages through the network

VANDERBILT UNIVERSITY

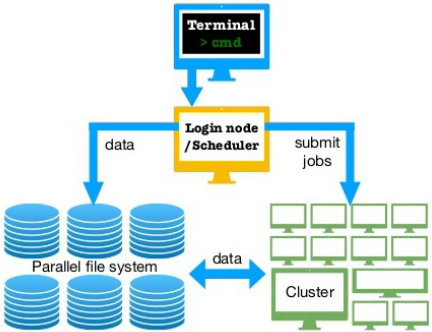
High Performance Computing



- **Large-scale distributed systems**
  - Shared by multiple users
  - Running multiple applications at the same time
  - Complex middleware to make everything work

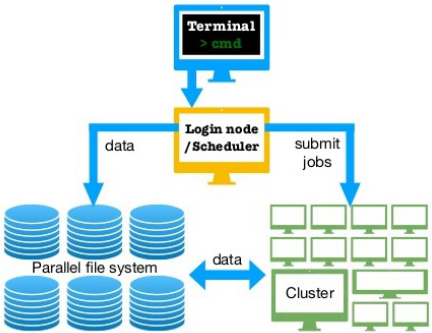
VANDERBILT UNIVERSITY

Middleware



VANDERBILT
UNIVERSITY

Middleware



- **Four important sub-systems**
  - Distributed operating system
    - Memory management, processes and communication management
  - Parallel file system
    - Access performance, resiliency, security
  - Scheduler
  - Daemons on compute nodes
    - Performance monitoring, fault tolerance

VANDERBILT UNIVERSITY

Today

Parallel file system

- What is a Parallel File Systems
- How to use a Parallel File Systems
- How are Parallel File Systems designed?
- **Current large-scale implementations**
  - GPFS, Lustre, PVFS
- Performance

What are parallel file systems?

- A filesystem has two main functions:
  - To organize and maintain the files namespace
  - To store the contents of the files and their attributes
- Store application data persistently
  - Usually extremely large datasets that can't fit in memory
- Provide global shared namespace (files, directories)
- The file system data correspond to the actual file contents.

What are parallel file systems?

- Designed for parallelism
  - Concurrent (often coordinated) access from many clients
- Designed for high-performance
  - Operate over high-speed networks (IB, Myrinet, Portals)
  - Optimized I/O path for maximum bandwidth

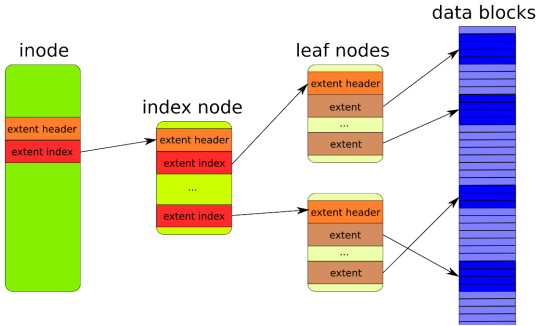
- PFS contain data and metadata
- Metadata is a set of information about files. They contain, for example:
  - Data position on the disks
  - File sizes
  - Creation, last modification and last access dates
  - The owners (UID and GID) and the permissions
  - ...

Sequential file systems

Local sequential file systems

- A local sequential filesystem is a filesystem that can not be directly accessed remotely.
- Only one client can access it (the operating system of the machine).
- In general, there is no parallelism (one simultaneous access at a time).

Example: structure of a large file on an ext4 filesystem



Parallel File Systems

A parallel filesystem is designed to enable simultaneous accesses to a filesystem to multiple clients

The differences with a simple shared filesystem is the level of parallelism:

- Multiple clients can read and write simultaneously, not one at a time.
- The distribution of data: a client will get good performance if the data is spread across multiple data servers.

- This parallelism is transparent to the client which sees the filesystem as if it was local
- In addition to the functions of a local filesystem, a parallel filesystem must efficiently manage potential conflicts between different clients.
  - The preferred approach is to use locks to limit and control concurrent accesses to a given file or directory

Parallel vs. Distributed

How are Parallel File Systems different from Distributed File Systems?

- **Data distribution**
  - Distributed file systems often store entire objects (files) on a single storage node
  - Parallel file systems distribute data of a single object across multiple storage nodes

Parallel vs. Distributed

How are Parallel File Systems different from Distributed File Systems?

- **Data distribution**
  - Distributed file systems often store entire objects (files) on a single storage node
  - Parallel file systems distribute data of a single object across multiple storage nodes
- **Symmetry**
  - Distributed file systems often run on architectures where the storage is co-located with the application (not always, e.g. GoogleFS, Ceph)
  - Parallel file systems are often run on architectures storage is physically separate from the compute system (not always true here either)

Parallel vs. Distributed

How are Parallel File Systems different from Distributed File Systems?

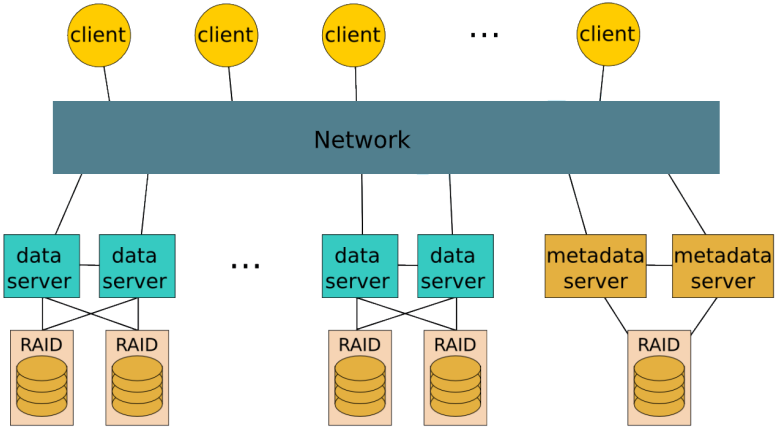
- **Data distribution**
  - Distributed file systems often store entire objects (files) on a single storage node
  - Parallel file systems distribute data of a single object across multiple storage nodes
- **Symmetry**
  - Distributed file systems often run on architectures where the storage is co-located with the application (not always, e.g. GoogleFS, Ceph)
  - Parallel file systems are often run on architectures storage is physically separate from the compute system (not always true here either)
- **Fault Tolerance**
  - Distributed file systems take on fault-tolerance responsibilities
  - Parallel file systems run on enterprise shared storage

Parallel vs. Distributed

How are Parallel File Systems different from Distributed File Systems?

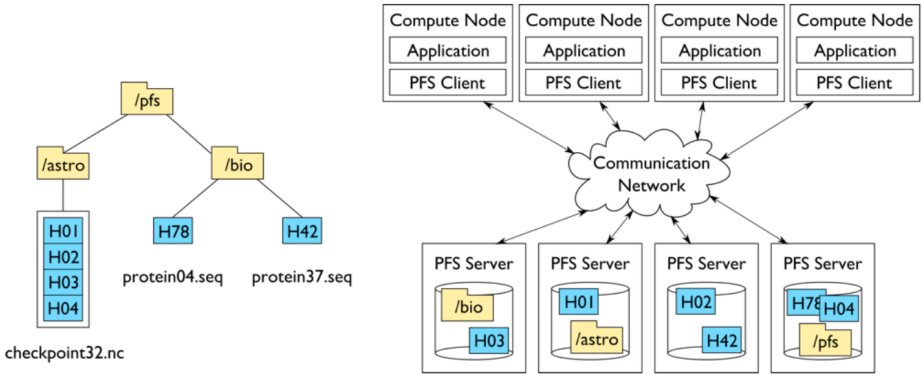
- **Data distribution**
  - Distributed file systems often store entire objects (files) on a single storage node
  - Parallel file systems distribute data of a single object across multiple storage nodes
- **Symmetry**
  - Distributed file systems often run on architectures where the storage is co-located with the application (not always, e.g. GoogleFS, Ceph)
  - Parallel file systems are often run on architectures storage is physically separate from the compute system (not always true here either)
- **Fault Tolerance**
  - Distributed file systems take on fault-tolerance responsibilities
  - Parallel file systems run on enterprise shared storage
- **Workloads**
  - Distributed file systems are geared for loosely coupled, distributed applications (think data-intensive)
  - Parallel file systems target HPC applications, which tend to perform highly coordinated I/O accesses, and have massive bandwidth requirements

Typical architecture



VANDERBILT UNIVERSITY

Parallel File System



- Provide a directory tree all nodes can see (the global name space)
- Map data across many servers and drives (parallelism of access)
- Coordinate access to data so certain access rules are followed (useful semantics)

VANDERBILT UNIVERSITY

Typical Architecture

A parallel filesystem is comprised of:

- **Clients** They will read or write data to the filesystem
- **One or more metadata servers** They manage metadata and placement of data on the drives, as well as access control locks
  - for example to avoid that 2 clients modify the same part of a file simultaneously
- **A number of data servers** These store all the data. For some parallel filesystems, data and metadata can be handled by the same server
- **One or more networks** (dedicated or not) for interconnecting these components

Who uses a parallel file system

Computational Science Applications

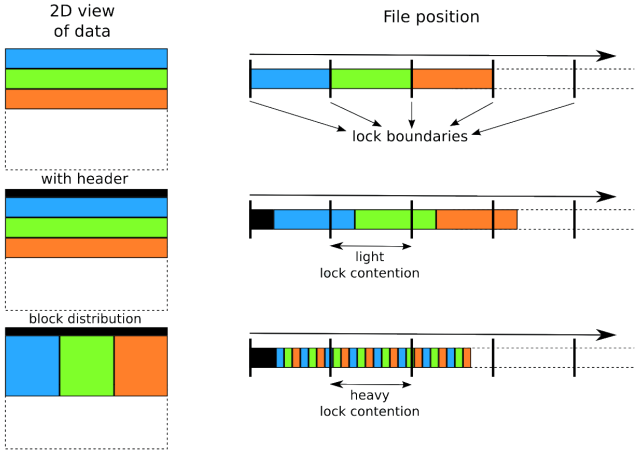
Problems are increasingly computationally challenging

- Large parallel machines needed to perform calculations
- Critical to leverage parallelism in all phases

Data access is a big challenge

- Using parallelism to obtain performance
- Finding usable, efficient, portable interfaces
- Understanding and tuning I/O

Examples


2D view
of data
with header
block distribution
File position
lock boundaries
light
lock contention
heavy
lock contention

Application and Storage Data Models

- Applications have data models appropriate to domain
  - Multidimensional typed arrays, images composed of scan lines, variable length records
  - Headers, attributes on data
- I/O systems have very simple data models
  - Tree-based hierarchy of containers of folders and files
- **High-level I/O libraries help map between these data models**

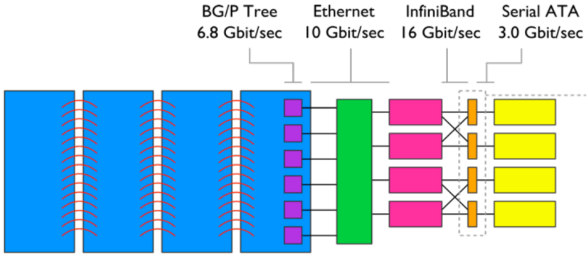
How to use the parallel file system

Scientific applications perform I/O to parallel file system in primarily one of two ways

- **Shared-file(N-to-1)** A single file is created, and all application tasks write to that file (usually to completely disjoint regions)
  - Increases usability: only one file to keep of by application
  - Can create lock contention and hinder performance on some systems

- **File-per-process(N-to-N)** Each application task creates a separate file, and writes to that only that file.
  - Avoids lock contention on file systems that use locks to maintain POSIX consistency
  - Applications running today create as many as 100,000 tasks
  - Impossible to restart application with different number of tasks

Where are Parallel File Systems deployed?



Gateway nodes run parallel file system client software and forward I/O operations from HPC clients.

640 Quad core PowerPC 450 nodes with 2 Gbytes of RAM each

Commodity network primarily carries storage traffic.

900+ port 10 Gigabit Ethernet Myricom switch complex

Storage nodes run parallel file system software and manage incoming FS traffic from gateway nodes.

136 two dual core Opteron servers with 8 Gbytes of RAM each

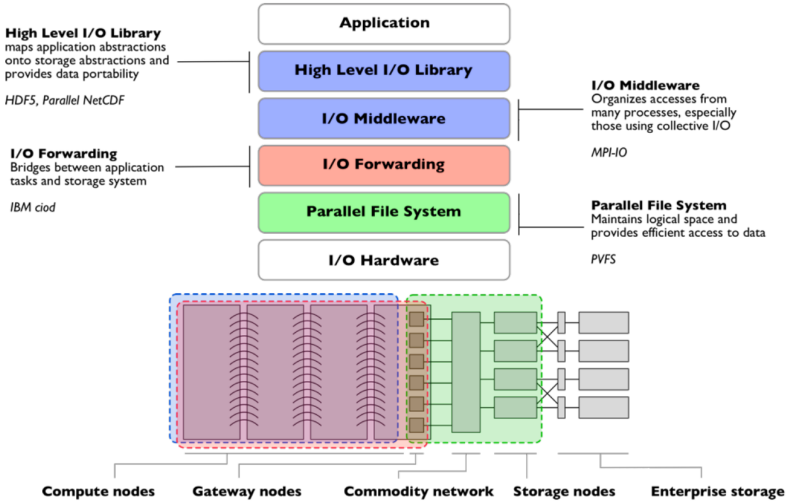
Enterprise storage controllers and large racks of disks are connected via InfiniBand or Fibre Channel.

17 DataDirect S2A9900 controller pairs with 480 1 Tbyte drives and 8 InfiniBand ports per pair

Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.

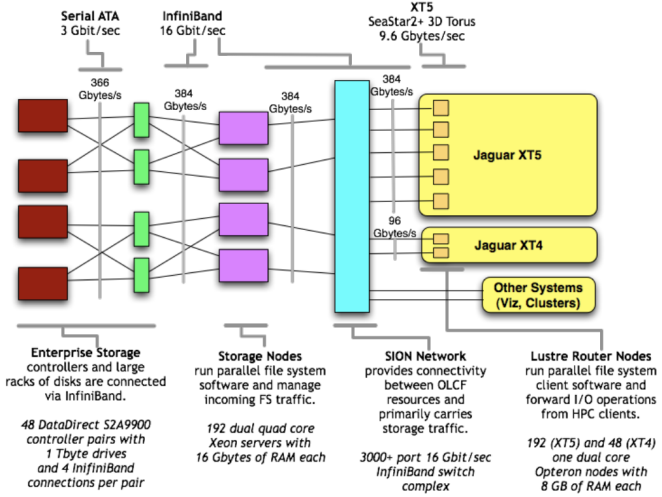
VANDERBILT UNIVERSITY

I/O Software Stack



VANDERBILT UNIVERSITY

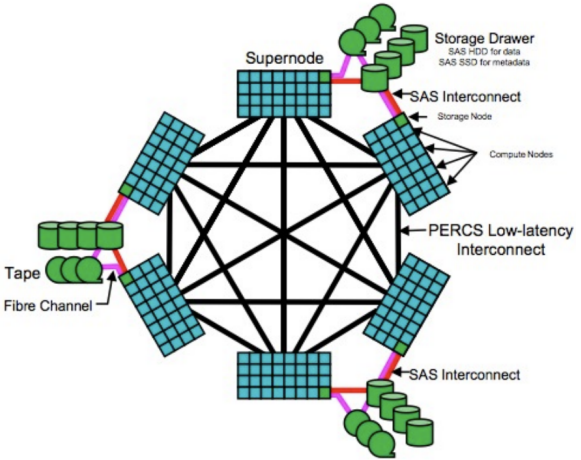
Jaguar Storage System



VANDERBILT UNIVERSITY

Blue Waters Storage System

- File System runs directly on compute nodes
- Storage nodes and physical storage embedded in compute racks
  - Metadata embedded as well
- All I/O messages ue internal fabric
  - Lower latency to storage
  - Reduced cost
  - May cause contention between I/O heavy and communication heavy application
- Example GPFS (ANL): 1.5 TB/s peak bandwidth and 18 PB Storage



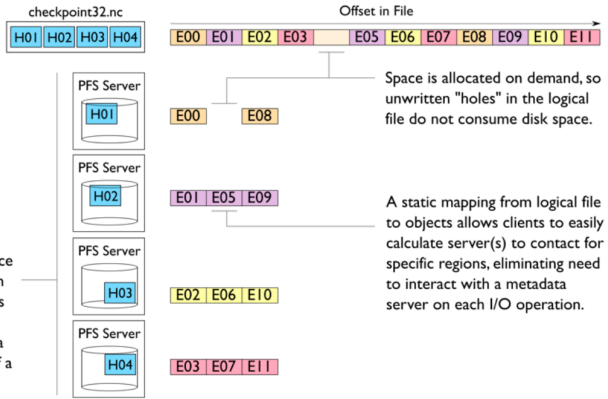
How are Parallel File Systems designed?

Data distribution in parallel file systems

Logically a file is an extendable sequence of bytes that can be referenced by offset into the sequence.

Metadata associated with the file specifies a mapping of this sequence of bytes into a set of objects on PFS servers.

Extents in the byte sequence are mapped into objects on PFS servers. This mapping is usually determined at file creation time and is often a round-robin distribution of a fixed extent size over the allocated objects.



VANDERBILT UNIVERSITY

File striping

- A file will usually be cut into pieces of fixed size (called stripes or chunks) and disseminated between different servers
- A read or write of the file will therefore be done in parallel on different file servers
- The speed of writing or reading will be the sum of the rates obtained on these servers.

Data Distribution

File striping

- A file will usually be cut into pieces of fixed size (called stripes or chunks) and disseminated between different servers
- A read or write of the file will therefore be done in parallel on different file servers
- The speed of writing or reading will be the sum of the rates obtained on these servers.

Data integrity and redundancy

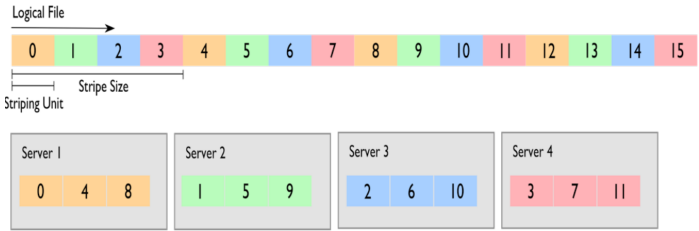
- The parallel filesystem must also ensure data integrity and system redundancy
  - Each data and metadata server manages several drives that use a local filesystem with RAID support ensuring data integrity in case of loss of one or more disks.
  - Data can be replicated in several different places.
  - A data or metadata server may be able to manage disks from another server and take over it in case of failure.
  - An alternative pathway for the data may exist (two different networks, for example).

Data Distribution

Round-round is a reasonable default solution

- Works consistently for a variety of workloads
- Works well on most systems
- Used by the majority of current PFS: GPFS, Lustre, PVFS

- **Can you think of a system where this might not work so well?**
- **What other distributions could be used?**

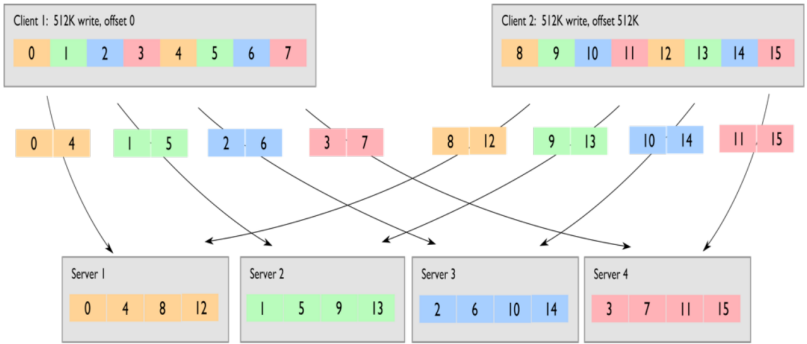


VANDERBILT UNIVERSITY

Data Distribution

Clients perform writes/reads of file at various regions

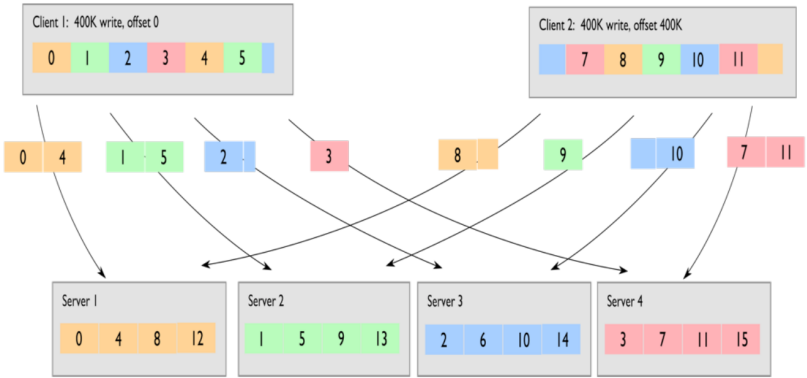
- Usually depends on application workload and number of tasks



VANDERBILT UNIVERSITY

Data Distribution

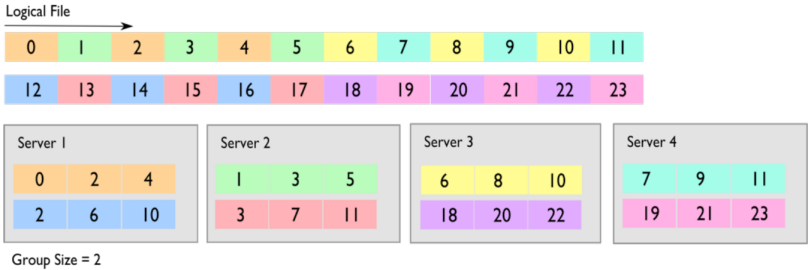
- Sizes of requests, alignment to striping unit is important



VANDERBILT UNIVERSITY

Data Distribution

- What happens when we have many servers (hundreds)?
- Two-dimensional distributions help
- Can also limit number of servers per file



VANDERBILT UNIVERSITY

Classes of Parallel File Systems: Blocks vs. Objects

Block-Based Parallel File Systems (Shared-disk)

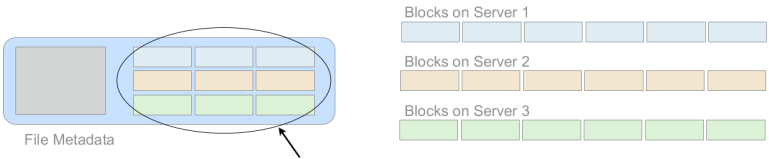
- Blocks are fixed-width
- File growth requires more blocks
- Blocks distributed over storage nodes
- Suffer from block allocation issues, lock managers
- Example: GPFS

Object-based Parallel File Systems

- Variable-length regions of the file
- A file has a constant number of objects
- Objects are given global identifiers (object-ids, handles, etc.)
- File growth increases the size of object(s)
- Objects are easier to manage and distribute
- Space allocation is managed locally on a per-object basis
- Examples: Lustre, PVFS

Blocks vs. Objects

- Metadata for a file includes distribution information
- Block-based file systems require dynamic metadata for distribution information



- Object-based file systems only need static metadata for distribution information

VANDERBILT UNIVERSITY

What is POSIX I/O?

A set of interfaces defined in 1970s:

- fd = open(filename, mode);
- read(fd, buffer, size);
- write(fd, buffer, size);

Specification also defines rules for maintaining consistency

- Two processes writing to overlapping regions must get consistent results from I/O system
- Easy on local file systems
- Distributed/Parallel file systems must manage consistency via locks
- Other alternatives exist

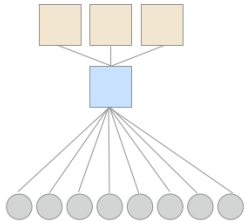
Let's look at an example

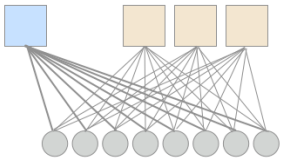
https://github.com/vanderbiltscl/SC3260_HPC/blob/master/lectures/4_pfs_example.pdf

POSIX consistency

How do POSIX interfaces/semantics affect Parallel File Systems?

- Overlapping regions create a major problem
- How does the Parallel File System provide POSIX consistency semantics?
- Two Choices:
  - **Centralized Management** All client requests are made to a broker server, which can serialize the requests to overlapping regions of a file and perform them in isolation where necessary
  - **Distributed Locking** Clients request a lock from a lock manager for the region of data they wish to access. Once a lock has been granted, clients can write exclusively to the region. This requires a Distributed Lock Manager (DLM): a server that hands out locks to clients as they request them.





VANDERBILT UNIVERSITY

Locks and concurrency

Purpose

- To ensure consistency of data and metadata, parallel filesystems usually use locks that limit concurrent access to this information
- This allows, among others, to ensure the atomicity of read/write operations
- For example, a process writes a block of data and one wants to read it at the same time. The use of a lock guarantees that the reader will read the data block as it was before or after the change (as it gets the lock before or after the writer), but never a mixture of both.

Implementation

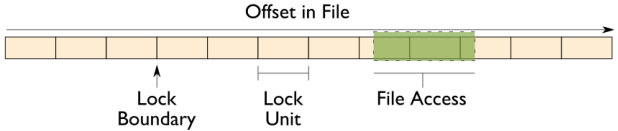
- Depending on the filesystem, the locks on the data are managed on a file-level or on a stripe-level basis
- They are aligned to certain boundaries
  - Example: size of memory pages for Lustre and size of the filesystem block for GPFS
- There are 2 main types of locks:
  - Exclusive locks for writes limiting access to a range from a single client
  - Shared locks for read accesses to a range by any number of clients and preventing changes/concurrent writes

Locking in Parallel File Systems

Most current parallel file systems use locks to manage concurrent access to files

- Files are broken up into lock units
- Clients obtain locks on units that they will access before I/O occurs
- Enables caching on clients as well (as long as client has a lock, it knows its cached data is valid)
  - Client can optimize small I/O with readahead
- Locks are reclaimed from clients when others desire access
- Locks are delegated and revoked through distributed lock managers

If an access touches any data in a lock unit, the lock for that region must be obtained before access occurs.



VANDERBILT UNIVERSITY

Example: https://github.com/vanderbiltscl/SC3260_HPC/blob/master/lectures/4_pfs_example.pdf

Distributed Lock Managers

- **Implementation burden**
  - Distributed Lock Managers add complexity to file system
  - What if the Distributed Lock Manager node fails?

Distributed Lock Managers

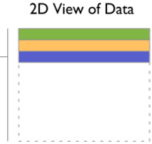
- **Implementation burden**
  - Distributed Lock Managers add complexity to file system
  - What if the Distributed Lock Manager node fails?
- **Locks are expensive**
  - Round-trip latencies between clients and Distributed Lock Manager
  - What happens on client failure?

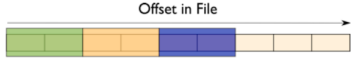
Distributed Lock Managers

- **Implementation burden**
  - Distributed Lock Managers add complexity to file system
  - What if the Distributed Lock Manager node fails?
- **Locks are expensive**
  - Round-trip latencies between clients and Distributed Lock Manager
  - What happens on client failure?
- **Solution: Lets just not write to overlapping regions**
  - Most applications do not write to overlapping regions concurrently anyway

Locking and Concurrent Access

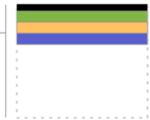
The left diagram shows a row-block distribution of data for three processes. On the right we see how these accesses map onto locking units in the file.

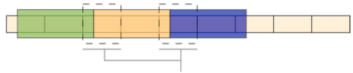




When accesses are to large contiguous regions, and aligned with lock boundaries, locking overhead is minimal.

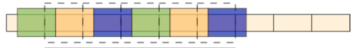
In this example a header (black) has been prepended to the data. If the header is not aligned with lock boundaries, false sharing will occur.





These two regions exhibit *false sharing*: no bytes are accessed by both processes, but because each block is accessed by more than one process, there is contention for locks.

In this example, processes exhibit a block-block access pattern (e.g. accessing a subarray). This results in many interleaved accesses in the file.



When a block distribution is used, sub-rows cause a higher degree of false sharing, especially if data is not aligned with lock boundaries.

PFS Implementation

- Simply does not provide POSIX consistency guarantees
- If two clients write to overlapping regions concurrently, **undefined results!**
- Computational Science doesn't often access overlapping regions
  - Better to use MPI to communicate shared state in a distributed fashion
  - Event notification shouldn't be implemented in the file system
  - Editing files is done interactively, not by clients accessing regions concurrently

PFS Implementation

- Simply does not provide POSIX consistency guarantees
- If two clients write to overlapping regions concurrently, **undefined results!**
- Computational Science doesn't often access overlapping regions
  - Better to use MPI to communicate shared state in a distributed fashion
  - Event notification shouldn't be implemented in the file system
  - Editing files is done interactively, not by clients accessing regions concurrently

What about appending records to a file?

- Don't care about offset, just want to append
- Requires (atomically) updated file size information
- Shared file pointers? Google FS uses *record append*

Caching

A cache is a local copy close to the one that uses it

- Its purpose is to accelerate performance.
- In a parallel filesystem, caches are used in two ways:
  - **Data servers side** Caches are located prior to the disks in random access memory (faster) and can be read and write
  - **Clients side** Data consistency between different clients must be assured. This is done via locks.
    - For example, a client which has write access will flush its caches to data servers if the corresponding lock is removed
    - Another case, if data is cached on some clients and another begins writing in the same part of the file, read caches will be invalidated (ie the data on them can no longer be used) before starting to read the newly written data from the data servers.

Each parallel filesystem has its own way of managing caches

Metadata in Parallel File Systems

- A single metadata server creates a single point of contention (hotspot)
  - Many clients try to open the same file at the same time: Creates an N-to-1 pattern of lookup requests
  - Many clients try to create new files at once: Creates an N-to-1 pattern of create requests
    In addition they require disk access too
- How can metadata be distributed across metadata servers?
  - Depends on underlying design (blocks vs. objects)

Main parallel filesystems

The most commonly used parallel filesystems in supercomputers are:

- Lustre
- GPFS
- PVFS2/OrangeFS
- PanFS

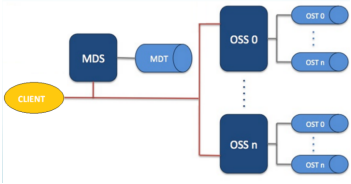
Lustre Architecture

Lustre is an open source parallel filesystem used by more than half of the Top 500 supercomputers (among others Summit, Sequoia and K computer). It runs on all major networks (InfiniBand, Myrinet, Quadrics, TCP/IP...).

- A Lustre system consists of
  - A metadata server (MDS) which manages a Meta Data Target (MDT) filesystem
  - Optionally, a backup metadata server that can take control in case of failure on the primary MDS
  - A series of data servers (called Object Storage Servers, OSS) that manage each several Object Storage Targets (OST)
  - clients

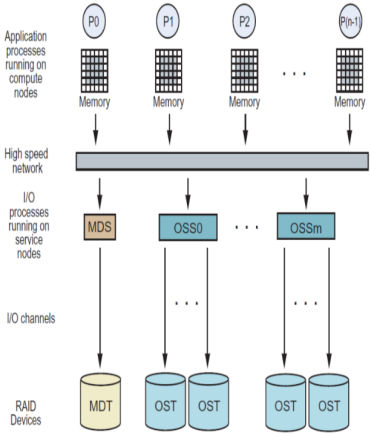
Lustre is the most used file system on today's clusters

- Metadata Server (MDS) makes metadata stored in the MDT(Metadata Target) available to Lustre clients.
  - The MDS opens and closes files and stores directory and file Metadata such as file ownership, timestamps, and access permissions on the MDT.
  - Each MDS manages the names and directories in the Lustre file system and provides network request handling for the MDT.
- Object Storage Server(OSS) provides file service, and network request handling for one or more local OSTs.
- Object Storage Target (OST) stores file data (chunks of files)



VANDERBILT UNIVERSITY

- Once a file is created, write operations take place directly between compute node processes (P0, P1, ...) and Lustre object storage targets (OSTs), going through the OSSs and bypassing the MDS.
- For read operations, file data flows from the OSTs to memory. Each OST and MDT maps to a distinct subset of the RAID devices.

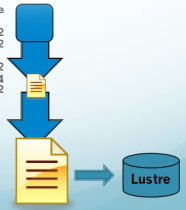


VANDERBILT UNIVERSITY

Case Study: Buffered I/O on Lustre

- A post processing application writes a 1GB file.
- This occurs from one writer, but occurs in many small write operations.
  - Takes 1080 s (18 minutes) to complete
- IO buffers were utilized to intercept these writes with 4 64 MB buffers.
  - Takes 4.5 s to complete. A 99.6% reduction in time

```
File "ssef_cn_2008052600f000"
                    Calls             Seconds         Megabytes   Megabytes/sec   Avg Size
      Open              1            0.001119
      Read            217            0.247026          0.105957        0.428931        512
      Write       2083634            1.453222       1017.398927      700.098632        512
      Close             1            0.220755
      Total       2083853            1.922122       1017.504884      529.365466        512
      Sys Read          6            0.655251        384.000000      586.035160   67108864
      Sys Write        17            3.848807       1081.145508      280.904052   66686072
      Buffers used              4 (256 MB)
      Prefetches                6
      Preflushes               15
```



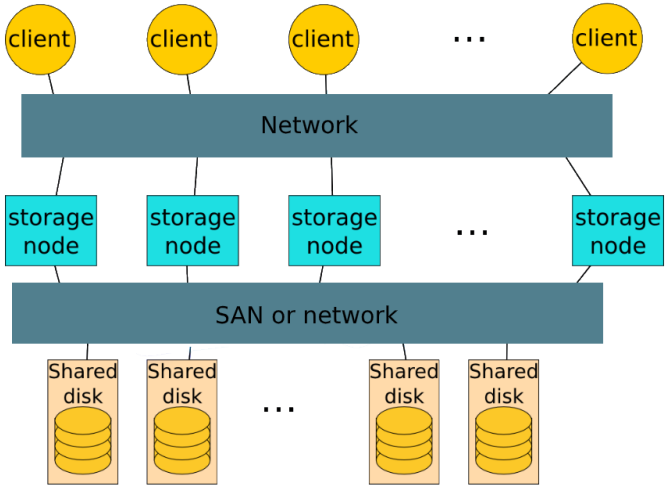
VANDERBILT UNIVERSITY

GPFS

- GPFS is a parallel filesystem developed by IBM. It is commercially licensed and used in many current supercomputers (including Mira).
- A GPFS system consists of:
  - A series of storage servers that deal with data and metadata (which can be separated or not),
  - A series of shared disks (SAN-attached or network block devices) and accessible through any storage server,
  - Clients.
- Metadata is distributed on different storage servers with a single node responsible for the metadata of a given file.
- Locks on a file (in bytes intervals) are, depending on circumstances, distributed among the various nodes or managed by a specific node.

GPFS is the file system used by ACCRE

GPFS Architecture



VANDERBILT UNIVERSITY

PVFS 2

PVFS (Parallel Virtual File System) and OrangeFS are open source parallel filesystems. They are very similar and vary only in the details. They run on the major networks (InfiniBand, Myrinet, Portals, TCP/IP...) currently used by small campus clusters.

- A PVFS2 or OrangeFS system consists of
  - One or several metadata servers,
  - A series of data servers,
  - Clients.
- They have some special features
  - Optimized for MPI with support for derived datatypes
  - Designed without locks (lockless or stateless)
    - This greatly simplifies things, but, for example, atomicity is not guaranteed if 2 clients write to the same location at the same time (the end results can be a mixture of the 2!)
- Do not follow the POSIX semantics.
- As Lustre, when a client wants to access a file, it contacts a metadata server which provides information on the data servers to use and then the client communicates directly with these servers to read or write.

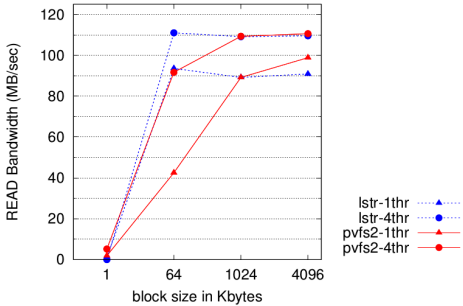
Parallel File System Comparisons

	PVFS2	LUSTRE
Striping Pattern	Configurable; defaults to round-robin	Round-robin
Stripe Size	Configurable; per directory	Configurable; per file/directory
Striping Width	Configurable; Up to # of available I/O servers	Configurable; Up to # of available I/O servers
Caching	No data caching in file clients; optional attribute caching	On file clients and file servers
Locking	Not supported by base protocol	Strong data, metadata locking at byte, page (4KB) granularity; Distributed intent-based locking
Semantics	“Non-conflicting writes”	POSIX semantics
Metadata Management	Multiple servers	Active-backup pair of servers

Performance Comparisons

PVFS2 and Lustre can both achieve the maximum achievable single-client bandwidth of about 110MB/s for large block sizes. For small blocks, performance difference due to:

- Lack of I/O parallelism in the application when using 1 thread
- Lack of prefetching policies in the PVFS2 file client



Read bandwidth; Single client, 12 servers

VANDERBILT UNIVERSITY

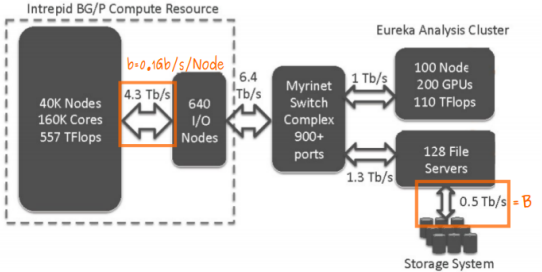
Real world implementation

How do storage systems look like in current clusters?

Real world implementation

Example Mira cluster at Argonne

- Dedicated I/O network
- I/O nodes distributed uniformly through the network between compute nodes
- Burst buffers at every I/O node
- GPFS

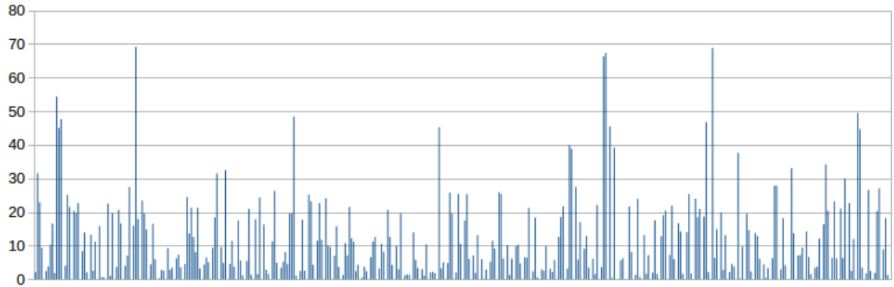


VANDERBILT UNIVERSITY

Congestion

Contention due to resource sharing (network, I/O)

- Multiple layers of I/O buffers: Compute Node / IO nodes / Disks
  - Aggregate IO requests
- Scheduling application I/O to avoid congestion



Analysis of the Intrepid system @Argonne: I/O throughput decrease (percentage per application, over 400 applications).

VANDERBILT UNIVERSITY

I/O Best Practices

- **Read small, shared files from a single task**
  - Instead of reading a small file from every task, it is advisable to read the entire file from one task and broadcast the contents to all other tasks.
- **Limit the number of files within a single directory**
  - Incorporate additional directory structure
  - Set the Lustre stripe count of such directories which contain many small files to 1
- **Avoid opening and closing files frequently**
  - This avoids excessive overhead
- **Consider available I/O middleware libraries**
  - For large scale applications that are going to share large amounts of data, one way to improve performance is to use a middleware library; such as ADIOS, HDF5, or MPI-IO

I/O Best Practices for Lustre

- **Place small files on single OSTs**
  - If only one process will read/write the file and the amount of data in the file is small (< 1 MB to 1 GB) , performance will be improved by limiting the file to a single OST on creation
  - This can be done by: *# lfs setstripe PathName -s 1m -i -1 -c 1*
- **Place directories containing many small files on single OSTs**
  - If you are going to create many small files in a single directory, greater efficiency will be achieved if you have the directory default to 1 OST on creation
  - *# lfs setstripe DirPathName -s 1m -i -1 -c 1*

Further reading

For advanced details on the design of Parallel File System refer to
Chapter 18 from **High Performance Computing, Modern Systems and Practices**, Thomas Sterling et al, 2018, Pages 549-578

Lustre
`http://doc.lustre.org/lustre_manual.pdf`
GPFS implementation on ACCRE
`https://www.vanderbilt.edu/accre/technical-details/`

The slides for today's lecture follow these presentations:

- Parallel File Systems, Argonne, Data-Intensive Computing guest lecture, Samuel Lang
- Introduction to parallel filesystems, Maison de la Simulation, Philippe Wautelet, CNRS - IDRIS, PATC Training session
- Lustre Parallel File System, KSL, Bilel Hadri