Parallel Distributed File Systems

Parallel Storage Systems

2023-05-22

Jun.-Prof. Dr. Michael Kuhn

michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

- What is a limitation of traditional storage architectures?
    1. Performance problems due to missing knowledge about other layers
    2. Long restoration times of storage arrays
    3. Having to deal with RAID, volume management and file systems separately

- What is a Merkle tree?
  1. A tree with at most two children per node
  2. A tree with at most four children per node
  3. A tree with cryptographic hashes within nodes
  4. A tree with values stored separately from nodes

- What is an advantage of copy on write?
    1. Journaling can be performed faster
    2. Journaling is not necessary anymore
    3. Updates can happen atomically

- Where would you keep deduplication tables?
    1. RAM
    2. SSD
    3. HDD
    4. No need to store, can be computed at runtime

Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

Definition

- Parallel file systems
  - Allow parallel access to shared resources such as files and directories
  - Access should be performed as efficiently as possible
- Distributed file systems
  - Data and metadata are distributed across multiple servers
  - Individual servers typically do not have a complete view of the system
- Naming is often inconsistent
  - Often just called "parallel file system" or "cluster file system"

Definition...

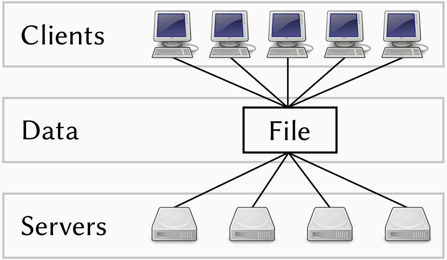
- Parallel distributed file systems are typically used in high performance computing
  - They are especially important for parallel I/O performed by applications
  - Home directories etc. might be handled by NFS
- Local file systems also offer parallel access
  - Locks can be realized using, for example, `flock` or `lockf`
  - Relatively easy to realize since all accesses pass the VFS
- Distribution requires an appropriate architecture and causes some overhead
  - Depending on the I/O interface, algorithms for distributed locks might be necessary

Definition...

- Storage Area Network (SAN)
  - Provides access to block devices via the network
  - Block devices can then be used with arbitrary file systems
    - Parallel distributed file system can also make use of SANs
- Network Attached Storage (NAS)
  - Abstracts from the underlying storage devices for more convenient access
  - Typically provides file system or similar interfaces
    - Examples are NFS or SMB

Architecture

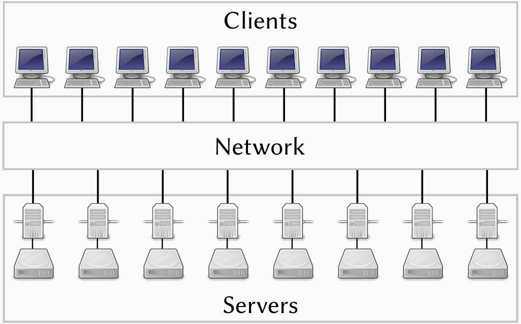
- Clients access a shared file in parallel
  - Enables efficient parallel I/O
- Data is managed within a file
- File is distributed across multiple servers by the file system
  - Offers high capacity and throughput



- Across how many servers would you distribute a 1 MB file?
  1. One server
  2. Eight servers
  3. 64 servers
  4. As many servers as possible

Architecture...

- Separation into clients and servers
  - Specialization for functionality
  - Minimize potential interference
- Clients execute parallel applications
  - Local storage for OS or caching
  - Access to file system via network
    - No direct access to storage devices
- Separate data and metadata servers
  - Different access patterns
    - Data vs. request throughput
  - Typically full-fledged servers



- Clients have to communicate with servers
- Different communication schemes are possible
  1. Clients know which servers to communicate with (more common)
  2. Clients communicate with a random server
     - Servers tell the client which server is responsible
     - Servers forward the request transparently
- Both schemes have advantages and disadvantages

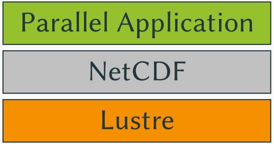
1. Clients know which servers to communicate with
   - Advantages: Communication protocol is easier, no communication between servers
   - Disadvantages: Distribution logic has to be implemented by clients, additional client-side information necessary
2. Clients communicate with a random server
   - Advantages: Clients do not need to know about data/metadata distribution, load balancing and replication are easier to realize
   - Disadvantages: Higher latency due to additional messages, more complex and error-prone communication protocol

Interface

- File system is accessed using an I/O interface
  - Typically standardized for portability
  - Proprietary interfaces might offer more functionality or performance
- Interfaces comprise syntax and semantics
  - Syntax defines available operations and their parameters
  - Semantics defines how operations behave
- POSIX I/O interface is often supported
  - Standardized and portable, even across computer types

Interface…

- Applications use highly abstracted interfaces
  - NetCDF offers a self-describing data format
  - Users only interact with NetCDF interface
- Parallel distributed file system realizes efficient access
  - Optimally, no knowledge about file system's implementation necessary
  - Users should not have to worry about file system



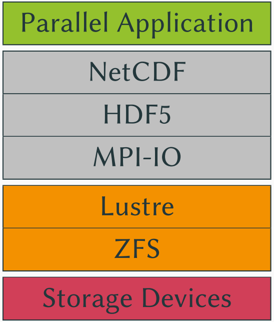
Semantics

- POSIX has strict consistency and coherence requirements
  - Changes have to be visible globally after a `write`
  - I/O should be performed atomically
- POSIX has been designed for local file systems
  - Requirements are relatively easy to support locally
  - All accesses are handled by the VFS, which can enforce semantics
- Small modifications and relaxations are possible
  - `strictatime`, `relatime` and `noatime` modify behavior regarding timestamps
  - `posix_fadvise` allows announcing access pattern

Semantics…

- Network File System (NFS) has very different semantics
  - It still offers the same interface, offering partial portability
- NFS implements the so-called session semantics
  - Changes are not directly visible for other clients
    - Changes are first performed within a client's own session
  - Other clients can see modifications after the session has ended
    - `close` writes back changes to the file and returns potential errors
- MPI-IO offers a third option for semantics
  - Less strict than POSIX but stricter than NFS
  - Its goal is to support highly-scalable parallel I/O

- Complex interplay of layers
  - Optimizations and workaround per layer
  - Requires knowledge about other layers
- Data is transformed
  - Data has to be transported through layers
  - Loss of structural information
- Convenience vs. performance
  - Structured data within application
  - Byte stream in POSIX file system



Reality...

Abstraction		Interface	Data Types	Control
High	⇑	NetCDF	Structures	Coarse-Grained
	⇕	MPI-IO	Elements	
Low	⇓	POSIX	Bytes	Fine-Grained

- High level of abstraction offers convenience but little control
  - "Write matrix `m`"
- Low level of abstraction allows fine-tuning I/O
  - "Write `n` bytes at offset `m` synchronously"

Examples

- Spectrum Scale (IBM, formerly GPFS)
- Lustre (DDN)
- OrangeFS (ANL, formerly PVFS)
- CephFS (Red Hat)
- BeeGFS (Fraunhofer, formerly FhGFS)
- GlusterFS (Red Hat)

Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

- I/O is expensive in relation to computation
    - Context switches, slow storage devices, additional latency etc.
    - Should happen asynchronously so the CPU does not have to wait
- Parallel distributed I/O has to be performed via the network
    - Additional restrictions regarding throughput and latency
- Novel concepts like burst buffers help alleviate problems
    - Data is temporarily stored and then forwarded to the file system
    - For example, nodes equipped with SSDs or node-local NVRAM/SSDs
- Both data and metadata performance can be problematic

Data…

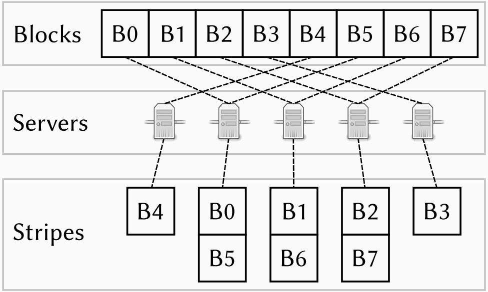
- Data is potentially being accessed by multiple clients concurrently
  - Read accesses are usually not problematic since there are no conflicts
  - Write accesses can be separated into overlapping or non-overlapping
- Overlapping write accesses
  - Typically require locks and therefore distributed lock management
  - Achievable performance depends on I/O semantics
    - POSIX guarantees correct handling, MPI-IO leaves behavior undefined
- Non-overlapping write accesses
  - Might suffer from performance problems due to coarse-grained locks

Data…

- Data distribution can be relevant for performance
  - Especially for non-overlapping accesses, which can be performed without locks
    - Different clients should write to different servers
- Realized using distribution functions
  - Typically round-robin but often configurable by the user
  - Potential support for heterogeneous access patterns
- Parallelism also determines number of data servers to contact
  - Not too few to allow high throughput
  - Not too many to keep overhead manageable

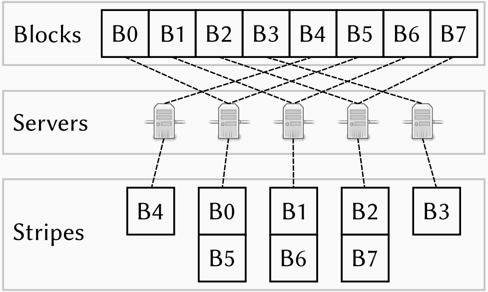
Data...

- File is split up into blocks
  - Blocks are distributed across servers
  - Here, eight blocks across five servers
  - Blocks typically have static size
- Round-robin distribution often used
  - Start at first server after last
- Does not have to start at first server
  - Typically randomly chosen server



Quiz

- Why is the starting server chosen randomly?
  - Easy implementation
  - Even load distribution
  - Fault tolerance



Metadata

- Metadata is usually accessed by multiple clients
  - Read accesses are typically not problematic again
  - Parallel updates to sizes, timestamps etc. lead to conflicts
- Metadata of one file or directory is managed by one server
  - Updates have to be serialized
  - Metadata cannot be distributed in a meaningful way
- Potentially several millions of clients
  - Distributed denial of service by clients on metadata servers

Metadata. . .

- Several approaches for efficient metadata handling
  - Some metadata is updated regularly
    - For instance, sizes and timestamps
  - Instead of storing those centralized, compute at runtime
    - Clients contacts all relevant data servers and determines global value
    - Improves update performance at the cost of query performance
- Metadata distribution can be done similarly to data distribution
  - Determine the responsible server by, for example, hashing the path
    - Hashing must be deterministic to find metadata again
    - Clients have to be able to determine server autonomously
  - One metadata object is typically managed by one server
    - Recently, directories have been split up across multiple servers

Metadata...

- Many metadata operations like path resolution are inherently serial
  - According to POSIX, it is necessary to check each path component

1. `/foo/bar`
   1.1 Read inode of the root directory
   1.2 Check access permissions and return an error if necessary
   1.3 Read root directory and look up `foo`

Metadata...

- Many metadata operations like path resolution are inherently serial
  - According to POSIX, it is necessary to check each path component

1. `/foo/bar`
   1.1 Read inode of the root directory
   1.2 Check access permissions and return an error if necessary
   1.3 Read root directory and look up `foo`
2. `/foo/bar`
   2.1 Read inode of the directory
   2.2 Check access permissions and return an error if necessary
   2.3 Read directory and look up bar

- Many metadata operations like path resolution are inherently serial
  - According to POSIX, it is necessary to check each path component

1. `/foo/bar`
   1.1 Read inode of the root directory
   1.2 Check access permissions and return an error if necessary
   1.3 Read root directory and look up `foo`
2. `/foo/bar`
   2.1 Read inode of the directory
   2.2 Check access permissions and return an error if necessary
   2.3 Read directory and look up bar
3. `/foo/bar`
   3.1 Read inode of the file
   3.2 Check access permissions and return an error if necessary
   3.3 Access file

- Separated servers allow optimization
  - HDDs for data, SSDs for metadata
  - Different prices (up to a factor of 10)
  - Metadata make up roughly 5 % of overall volume
- Potential problem: Software has to be able to utilize performance

Tech.	Device	IOPS
HDD	7,200 RPM	≈ 80
	10,000 RPM	≈ 150
	15,000 RPM	≈ 200
SSD	OCZ Vertex 4	≈ 120,000
	Samsung SSD 960 EVO	≈ 380,000
	Fusion-io ioDrive Octal	≈ 1,200,000

[Wikipedia, 2021]

Performance

- 2009: Blizzard (GPFS)
  - Computation: 158 TFLOPS
  - Capacity: 7 PB
  - Throughput: 30 GB/s
- 2015: Mistral (Lustre)
  - Computation: 3.6 PFLOPS
  - Capacity: 60 PB
  - Throughput: 450 GB/s (5.9 GB/s per node)
  - IOPS: 400,000 operations/s
- 2022: Levante (Lustre)
  - Computation: 14 PFLOPS
  - Capacity: 130 PB
- 2012: Titan (ORNL, Lustre)
  - Computation: 17.6 PFLOPS
  - Capacity: 40 PB
  - Throughput: 1.4 TB/s
- 2019: Summit (ORNL, Spectrum Scale)
  - Computation: 148.6 PFLOPS
  - Capacity: 250 PB
  - Throughput: 2.5 TB/s

Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

Overview

- One of the most popular parallel distributed file systems
  - Used on more than half of the TOP100
  - Used on more than a third of the TOP500
- Open source (GPLv2)
  - More than 550,000 lines of code
- Supports Linux
  - Name is a combination of Linux and cluster
  - Core functionality is implemented within kernel modules

History

- 1999: Start of development
  - Research project at Carnegie Mellon University, lead by Peter Braam
- 2001: Founding of Cluster File Systems
- 2007: Acquisition by Sun
  - Integration with Sun's HPC hardware, combination with ZFS
- 2010: Acquisition by Oracle
  - End of development, further development by the community
- 2012: Acquisition by Intel
- 2018: Acquisition by DDN
  - Separate division called Whamcloud

History… [OpenSFS and EOFS, 2021]

- Version 2.3 (October 2012)
  - Experimental support for ZFS
- Version 2.4 (May 2013)
  - Distributed Namespace (DNE) and ZFS for data and metadata
- Version 2.5 (October 2013)
  - Hierarchical Storage Management (HSM)
- Version 2.6 (July 2014)
  - Experimental support for striped directories
- Version 2.8 (March 2016)
  - Support for striped directories and cross-MDT metadata operations

History… [OpenSFS and EOFS, 2021]

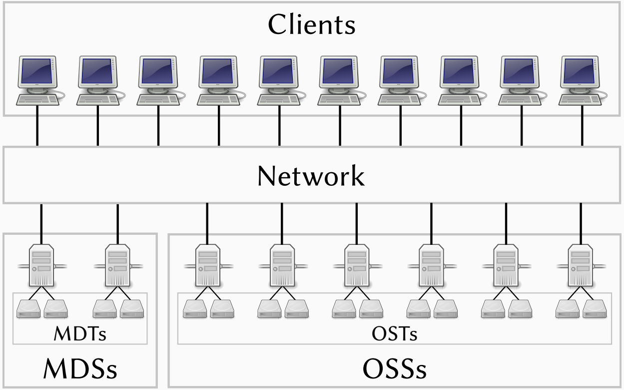
- Version 2.9 (December 2016)
  - Support for Kerberos (authentication, encryption), 16 MiB RPCs and `ladvise`
- Version 2.10 (June 2017)
  - File system snapshots, project quota and progressive file layouts
- Version 2.11 (April 2018)
  - Data on metadata servers, file-based redundancy and lock ahead
- Version 2.12 (December 2018)
  - Lazy Size on MDT
- Version 2.13 (December 2019)
  - Persistent Client Cache (NVRAM, NVMe)
- Version 2.14 (February 2021)
  - Client Data Encryption

- DNE allows distributing directories
  - Servers can be tuned for different workloads
  - Directories might be too large for one server
- Responsible servers can be specified
  - scratch for large, home for small files
  - Static approach, distribution has to be specified manually
- Support for striped directories
  - striped is striped across three servers

```
$ lfs mkdir --index 0 /l/home
$ lfs mkdir --index 1 /l/scratch
$ lfs mkdir --count 3 /l/striped
```

Architecture

- Applications on compute nodes
  - Compute nodes are clients
- Communication via network
  - Often separate for performance
- Two different server types
  - OS means Object Storage
  - MD means MetaData



Architecture…

- Object Storage Servers (OSSs)
  - Manage data in the form of object with byte-level access
  - Data is stored on Object Storage Targets (OSTs)
    - A target can be a RAID array or a single device
- Metadata Servers (MDSs)
  - Manage metadata but are not involved in the actual I/O
  - Metadata is stored on Metadata Targets (MDTs)
    - Metdata access patterns can make SSD RAIDs feasible
- Distribution happens across targets, not servers
  - A server can manage multiple targets
  - Multiple targets might not have performance benefits if on the same server

Architecture…

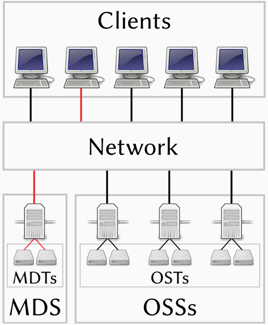
- Data and metadata servers use local file systems
  - Traditionally, ldiskfs (ext4 fork)
    - Redundant functionality causes overhead
    - No support for checksums, compression, snapshots etc.
  - Alternatively, ZFS
    - Allows direct access to Data Management Unit and Adaptive Replacement Cache
- Clients cannot access storage devices directly
  1. Client sends request to server
  2. Server executes operations
  3. Server sends reply to client

Architecture...

- Lustre uses a direct communication protocol
  - Clients contact the management server and query available servers
- Servers typically do not have to communicate with each other
  - Servers have to know about the management server and (un)register there
  - Metadata servers might have to communicate with each other

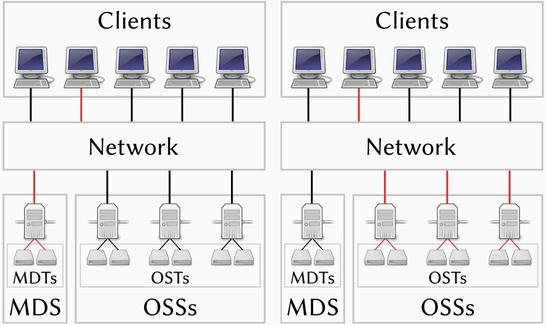
1. Clients contact management server
   - Configured via `/etc/fstab` or the `mount` command
   - Management server returns information about available servers etc.
2. Clients contact the responsible metadata server
   - Metadata server returns information about distribution and data servers
3. Clients contact the responsible data servers

- Metadata access for initial opening
  - Metadata server returns distribution



Architecture...

- Metadata access for initial opening
  - Metadata server returns distribution
- Direct parallel access via data servers
  - Provided no metadata is changed
  - Metadata server would be bottleneck for parallel access



Support

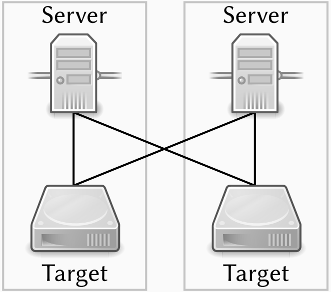
- Lustre is a kernel file system
  - Both client and server run in kernel space
- Client supports relatively new kernels
  - Support for new kernels sometimes takes a while
  - Requires using recent Lustre versions without long time support
- Server only supports selected enterprise kernels
  - Red Hat Enterprise Linux (or CentOS) or Ubuntu
  - Mainly due to ldiskfs
    - A “patchless” variant has been made available
    - Can also be used with standard kernel when using ZFS

Functionality

- Distributed lock management for parallel access
  - Used for both data and metadata
  - Overlapping read locks and non-overlapping write locks with byte granularity
  - Locks can be disabled with the mount options `nolock`
- Supports explicit locks
  - Mount options `noflock`, `localflock` and `flock` can influence them
- Lustre is mostly POSIX-compliant
  - POSIX interface is provided by the VFS kernel module
  - No native support for MPI-IO available
    - The popular ROMIO implementation provides a special Lustre module

Functionality. . .

- Hierarchical storage management
  - Important requirement for large-scale storage systems
  - Lustre supports multiple so-called tiers
    - SSDs, HDDs, tapes etc.
  - Metadata is always managed by Lustre
    - Data is moved to different tiers transparently
- High availability
  - Lustre supports failover mechanisms
    - Active/passive and **active/active** configurations
  - Servers can take over each others' duties
    - A servers might management two targets temporarily
  - Can also be used for seamless upgrades



Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

Overview

- Popular research file system
  - Developed by Clemson University, Argonne National Laboratory and Omnibond
- Open source (LGPL)
  - More than 250,000 lines of code
- Successor of PVFS (Parallel Virtual File System)
  - 2007: Start as a new development branch
  - 2010: Replaces PVFS as the main version

Functionality [OrangeFS Development Team, 2021]

- Basic functionality of a parallel distributed file system
  - Including distributed data, metadata and directories
- Runs completely in user space
  - Less maintenance overhead than kernel code
- Very good MPI-IO support
  - Native backend within ROMIO
- Also supports the POSIX interface
  - POSIX libraries or a FUSE file system are available
  - Alternatively, an optional kernel module can be used

Functionality…

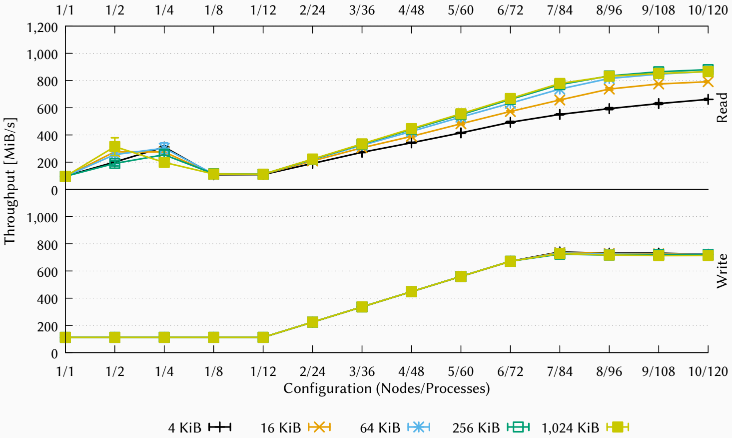
- OrangeFS is not POSIX-compliant
  - Supports atomic non-contiguous and non-overlapping accesses
  - Consequently, supports (non-atomic) MPI-IO
- Sufficient for many use cases, including many parallel applications
  - Stricter semantics is not supported
  - MPI-IO’s atomic mode is not available due to missing locks

Outline

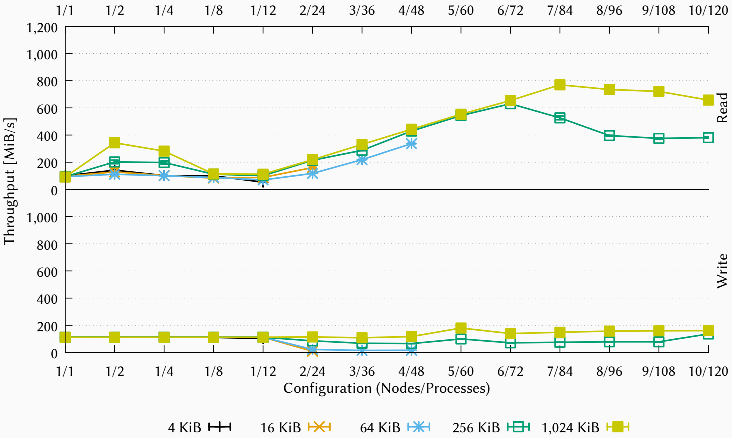
Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

- Highly optimized for parallel access
  - No performance degradation with individual accesses
- Optimizations
  - Aggregates write operations in RAM
  - Performs read ahead and lock ahead



- Shared access is problematic
  - Caused by strict POSIX semantics
- High performance requires alignment
  - 1-to-1 client-server accesses if possible



Outline

Parallel Distributed File Systems

- Review
- Concepts
- Performance Considerations
- Example: Lustre
- Example: OrangeFS
- Performance Assessment
- Summary

Summary

- Parallel distributed file systems offer efficient parallel access
  - Distribution of data and metadata increases throughput and capacity
- Data and metadata servers are usually separated
  - Different access patterns require different optimizations
- Access is performed via I/O interfaces
  - Often used with POSIX or MPI-IO, which have very different semantics
- Efficient use of parallel distributed file system can be complex
  - In-depth knowledge about behavior is required
  - I/O libraries offer optimizations and convenience functionality

References

[OpenSFS and EOFS, 2021] OpenSFS and EOFS (2021). **Lustre.** https://www.lustre.org/.

[OrangeFS Development Team, 2021] OrangeFS Development Team (2021). **OrangeFS.** https://www.orangefs.org/.

[Wikipedia, 2021] Wikipedia (2021). **IOPS.** http://en.wikipedia.org/wiki/IOPS.