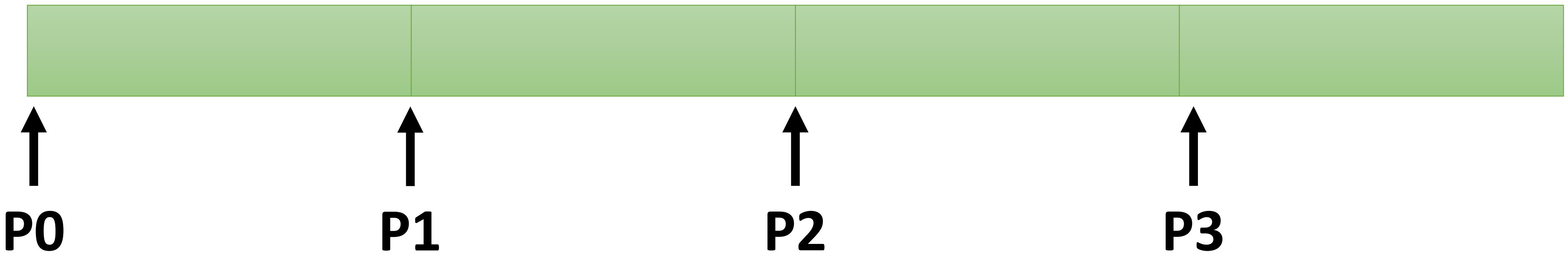
Parallel I/O – III

Apr 20, 2021

Recap – Parallel I/O

Uniform access pattern



Each process reads a big chunk of data (one-fourth of the file) from a common file

Independent I/O

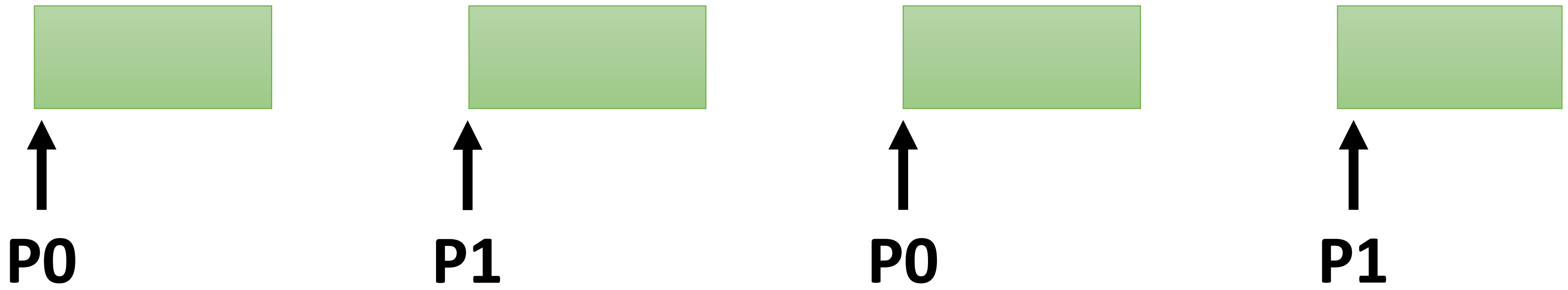
```
MPI_File_set_view (fh, rank*file_size_per_proc, …)
MPI_File_read (fh, data, datacount, MPI_INT, status)
```

fh: individual file pointer

Recap – File View

MPI_File_set_view (	
fh,	file handle
rank*file_size_per_proc,	displacement (bytes to be skipped from the start of the file)
MPI_INT,	etype (any basic or derive MPI datatype)
MPI_INT,	filetype (constructed out of etype, specifies visible portions)
“native”,	data representation (native → same as memory)
info	file info hints
)	

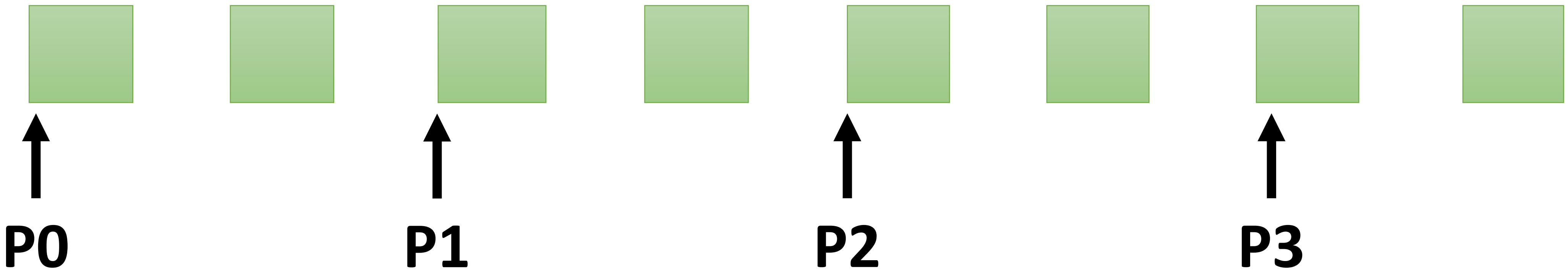
Using File Views



Each process reads a small chunk of data from a common file

```
MPI_File_set_view (fh, displacement, etype, filetype, “native”, info)
MPI_File_read_all (fh, data, datacount, MPI_INT, status)
```

Discontiguous Access Pattern



Each process reads small discontiguous chunks of data from a common file

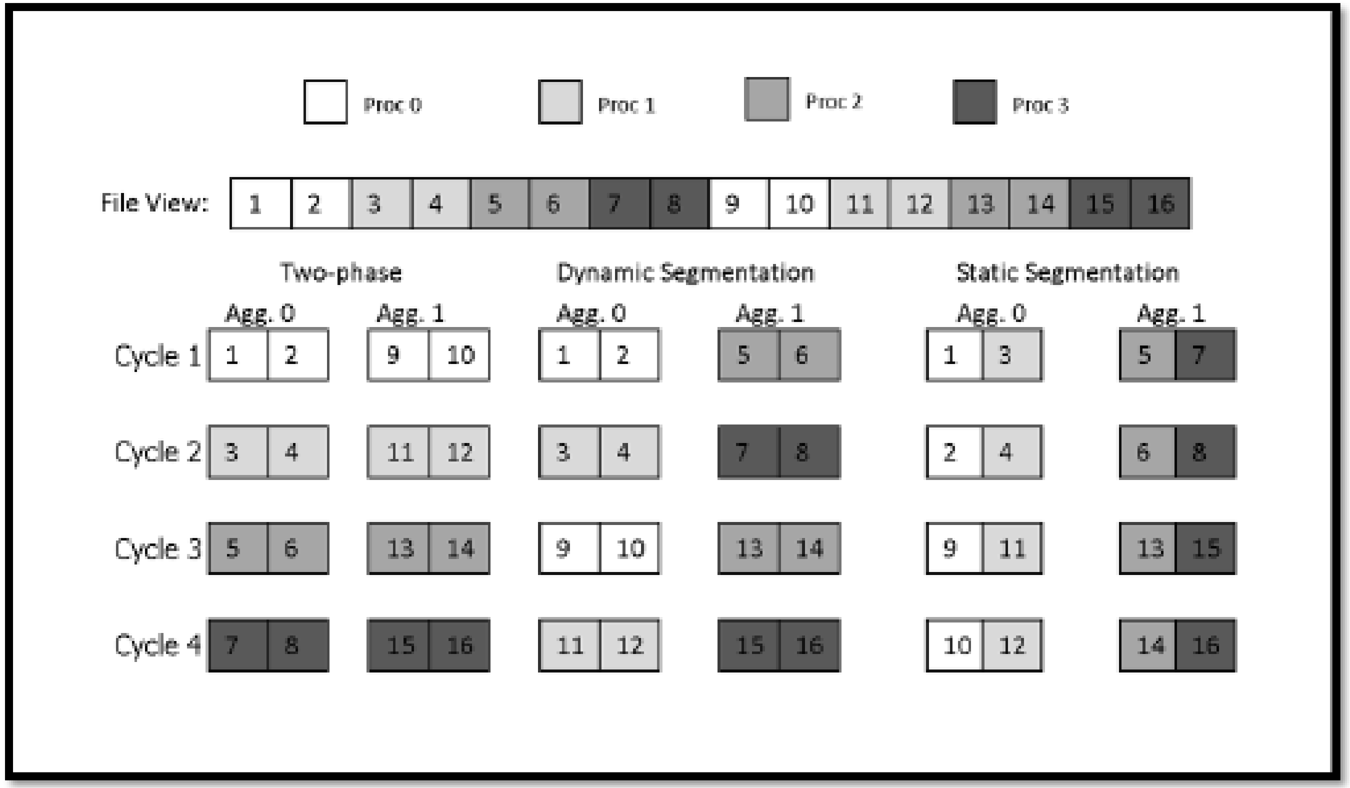
Collective I/O – Selecting Aggregators

Mohamad Chaarawi and Edgar Gabriel, Automatically Selecting the Number of Aggregators for Collective I/O Operations, IEEE Cluster 2011

Three approaches for collective I/O aggregation

- Two-phase
  - Shuffle phase
  - I/O phase
- Dynamic segmentation
  - Fixed #processes per aggregator
- Static segmentation
  - Extension of dynamic segmentation
  - Fixed #bytes per cycle per process per aggregator

Collective I/O – Selecting Aggregators

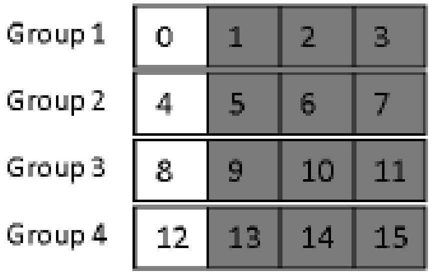


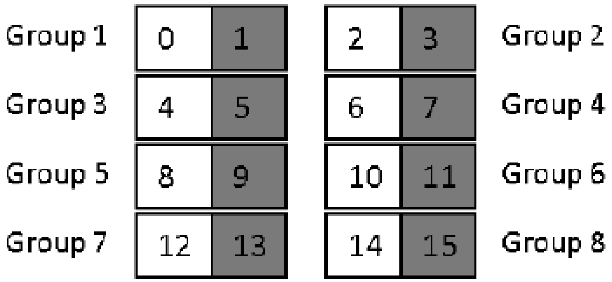
Source: Chaarawi et al., Automatically Selecting the Number of Aggregators for Collective I/O Operations, Cluster'11

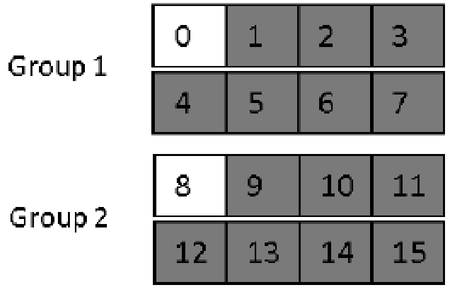
Collective I/O – Selecting Aggregators

Select

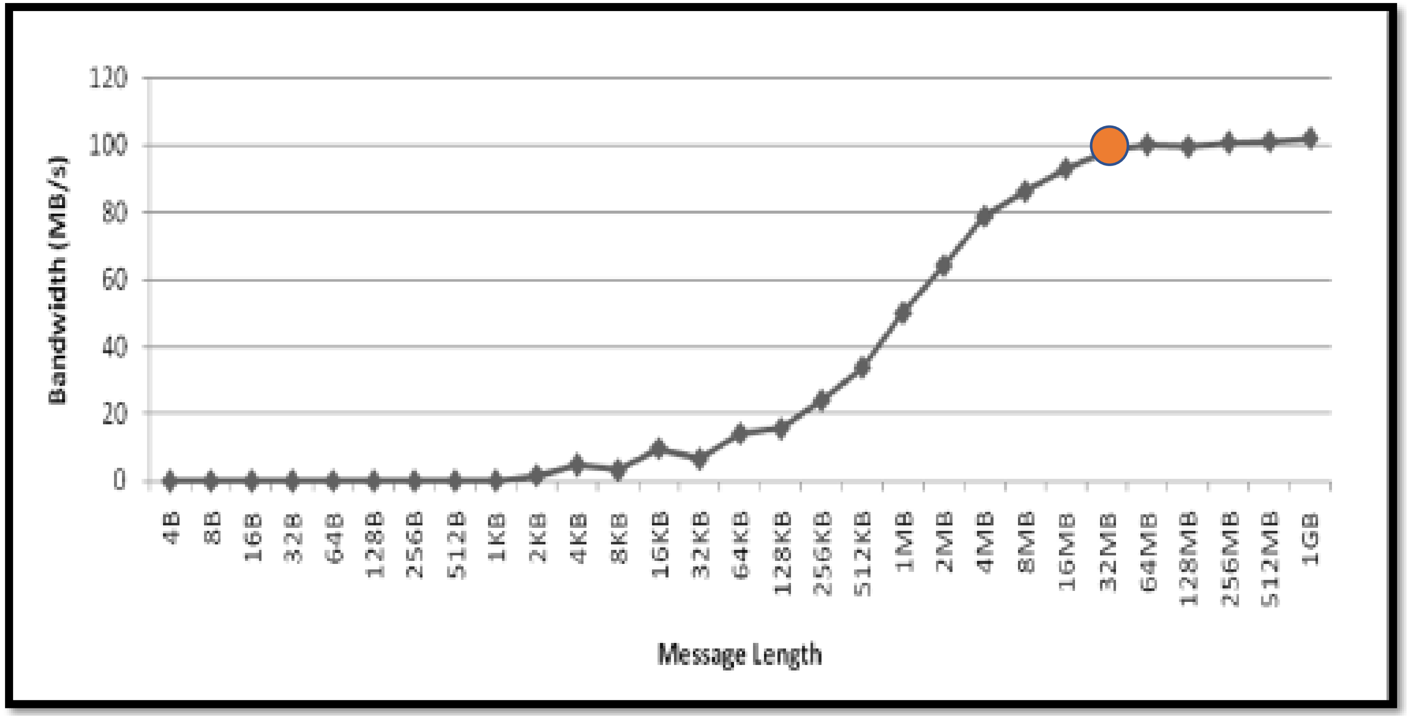
- Data size k (write saturation point)
  - Using a benchmark
  - Determined by network topology, I/O network, parallel file system, etc.
- Initial #aggregators
- Refine #aggregators
  - If total bytes per group > k, then split
  - If total bytes per group < k, then join







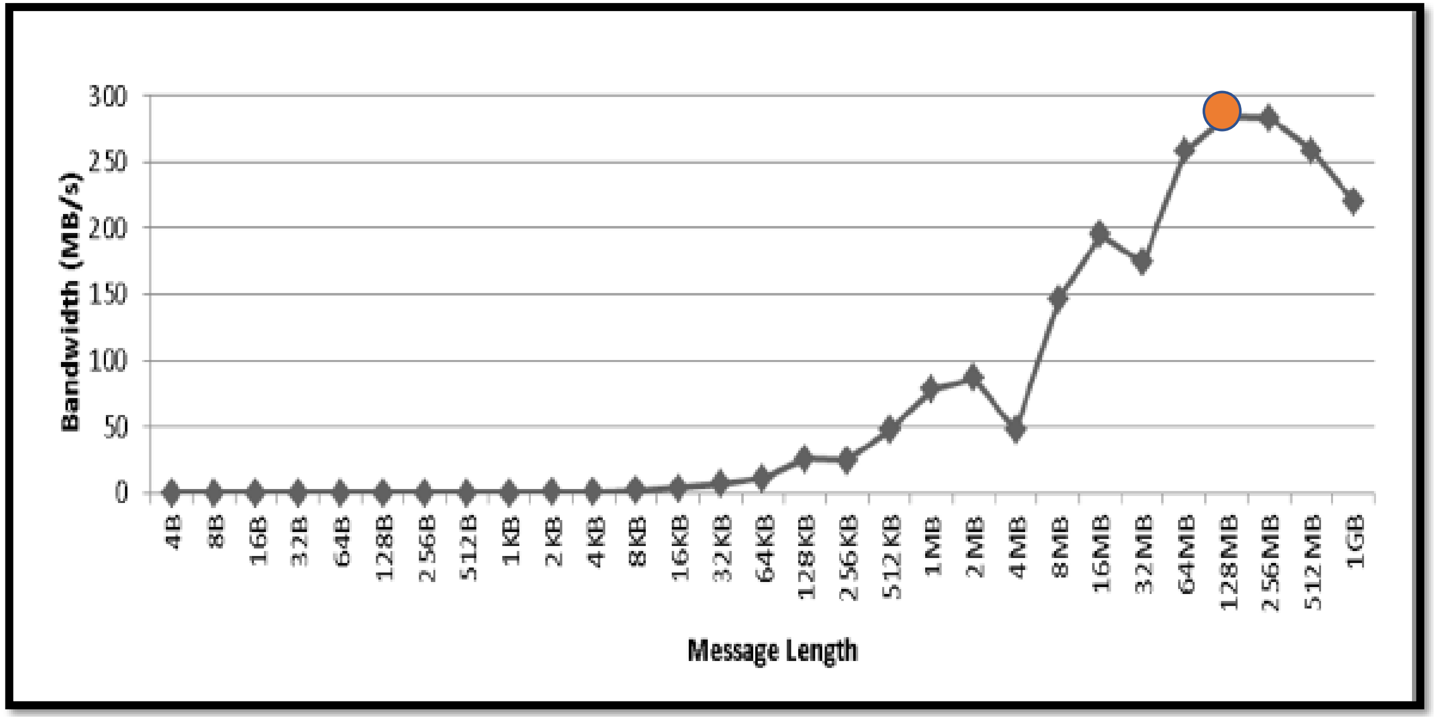
Tuning for Bandwidth Saturation



Shark

- 29 nodes
- SDR Inifiniband and Gigabit Ethernet
- PVFS2

Tuning for Bandwidth Saturation

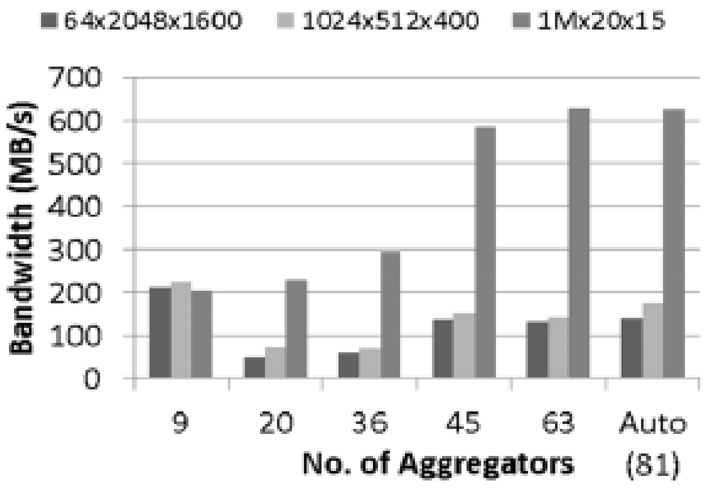


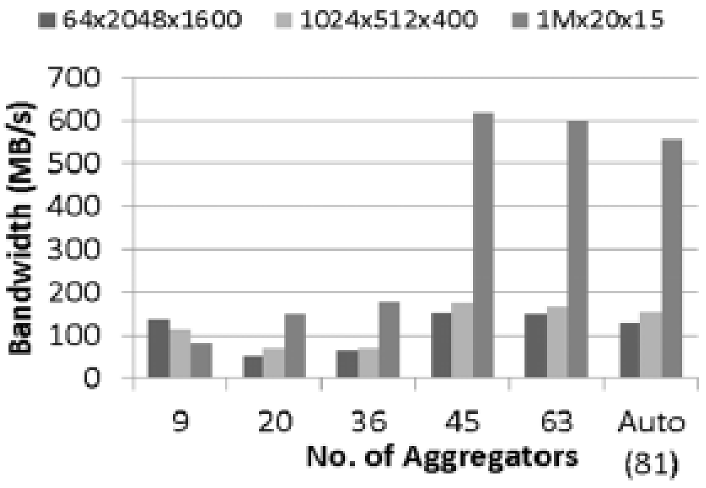
Deimos
- 724 nodes
- 11 I/O servers via SDR Inifiniband
- Lustre

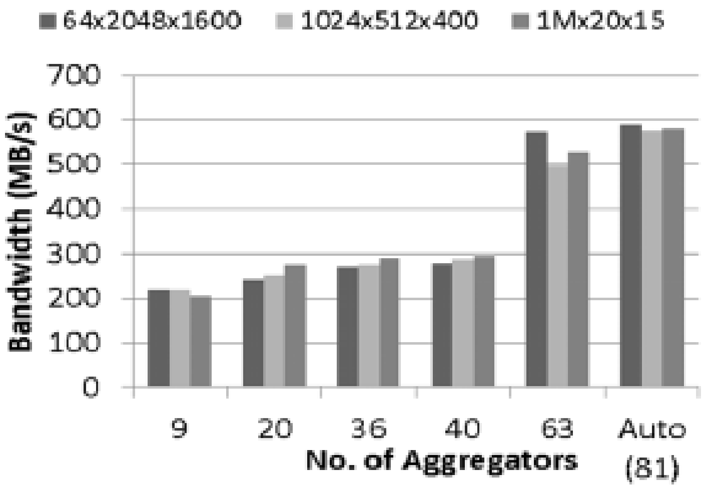
Dynamic vs. Static vs. Two-phase on Shark

The number of aggregators chosen is equal to the total number of processes, because each process is requesting to write data larger than the value of *k* per function call.

64 B, 1 KB, 1 MB





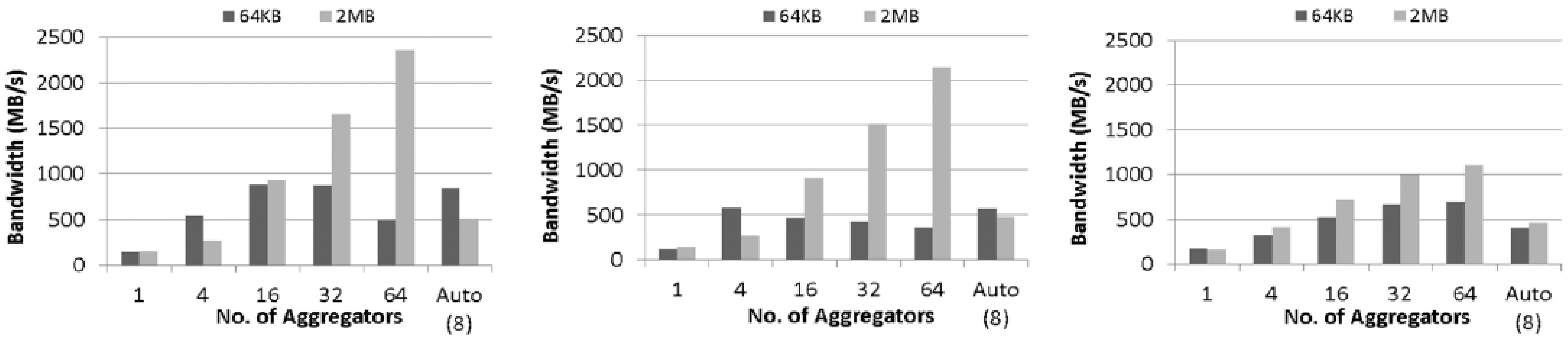


MPI Tile IO benchmark with 81 (9 X 9) processes

Dynamic vs. Static vs. Two-phase on Deimos

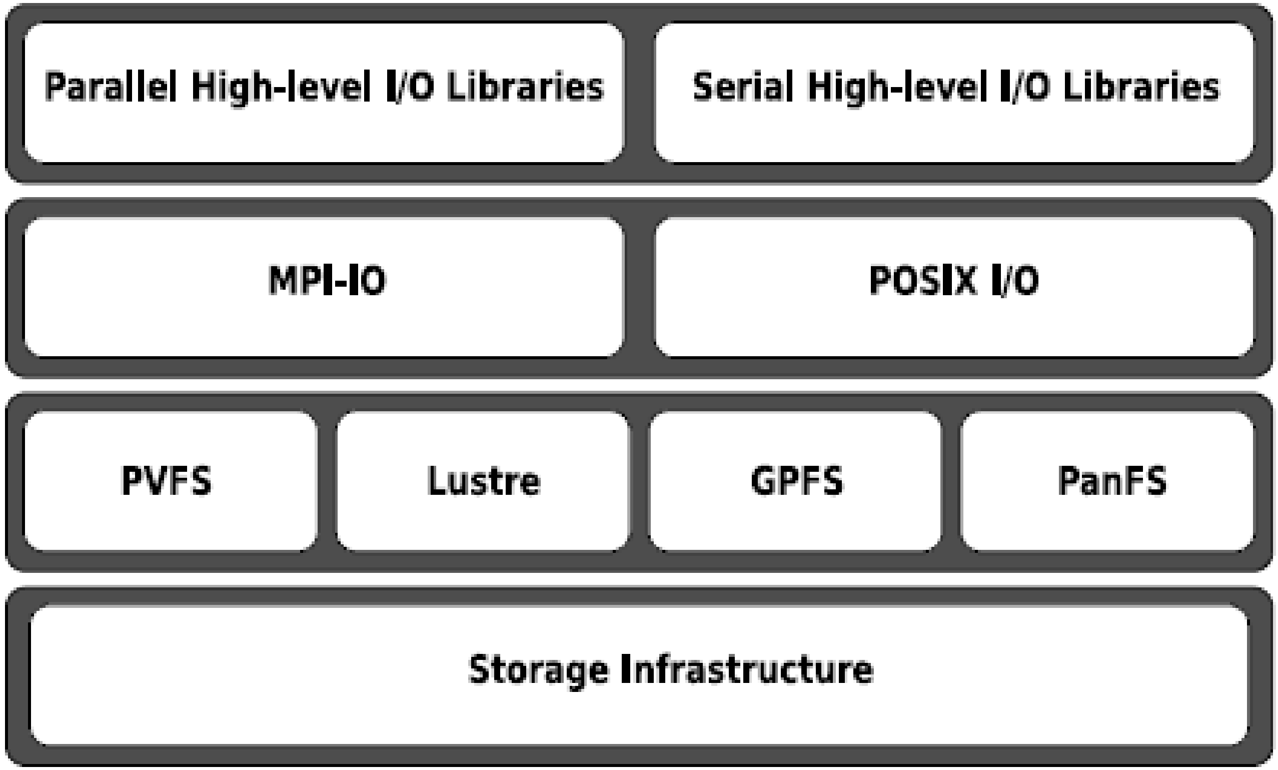
- The number of aggregators chosen was good enough for small size, but communication costs dominated the cost of large segment size.
- 2 MB being a multiple of stripe size did not affect performance

Size of a segment



Latency IO benchmark with 64 processes

I/O Stack

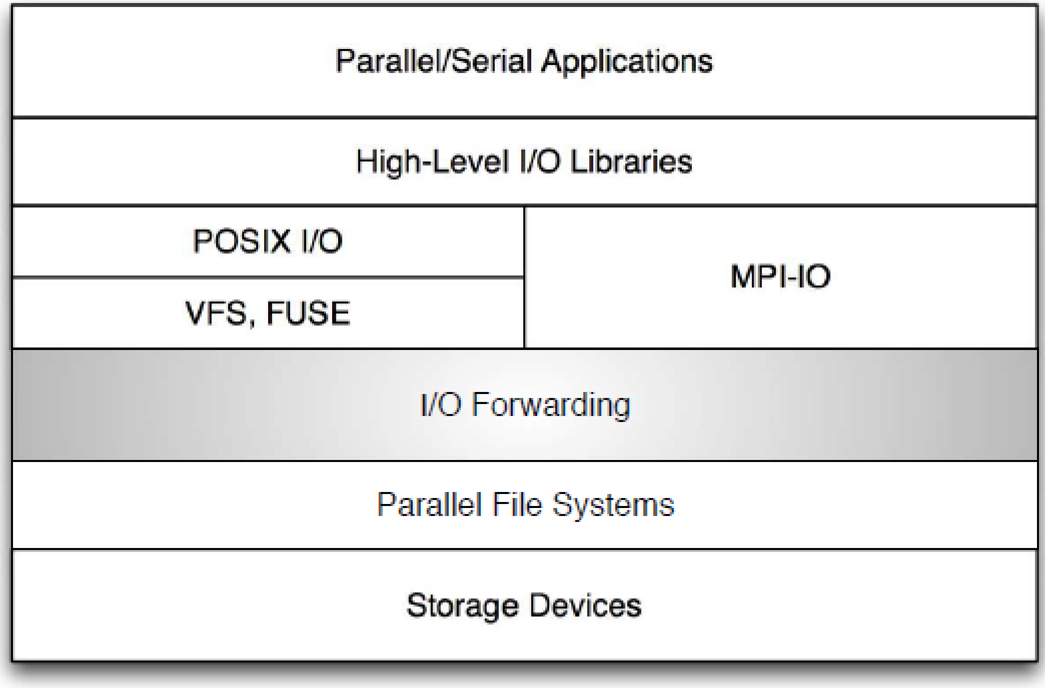

Parallel High-level I/O Libraries
Serial High-level I/O Libraries
MPI-IO
POSIX I/O
PVFS
Lustre
GPFS
PanFS
Storage Infrastructure

High Data Throughput

How?

- I/O forwarding from compute to I/O nodes
- Multiple I/O servers to manage the data storage
- A large file may be striped across several disks

I/O Forwarding

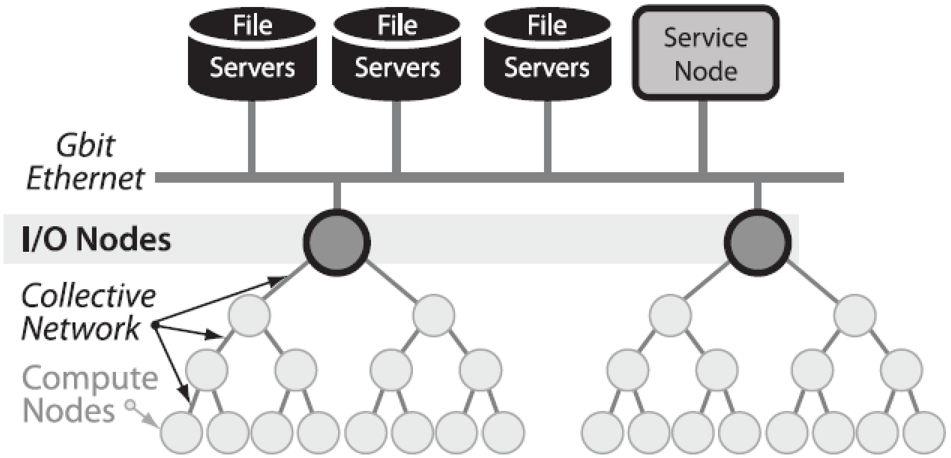


Source: Ohta et al., "Optimization Techniques at the I/O Forwarding Layer"

- I/O requests forwarded to dedicated I/O nodes by compute node kernels
- I/O nodes redirect I/O requests to the backend parallel file systems
- Reduces the number of clients accessing the file systems
- Can reduce the file system traffic by aggregating and reordering I/O requests
- I/O forwarding scheduler can exploit the global view of parallel applications to sort and merge I/O requests more effectively

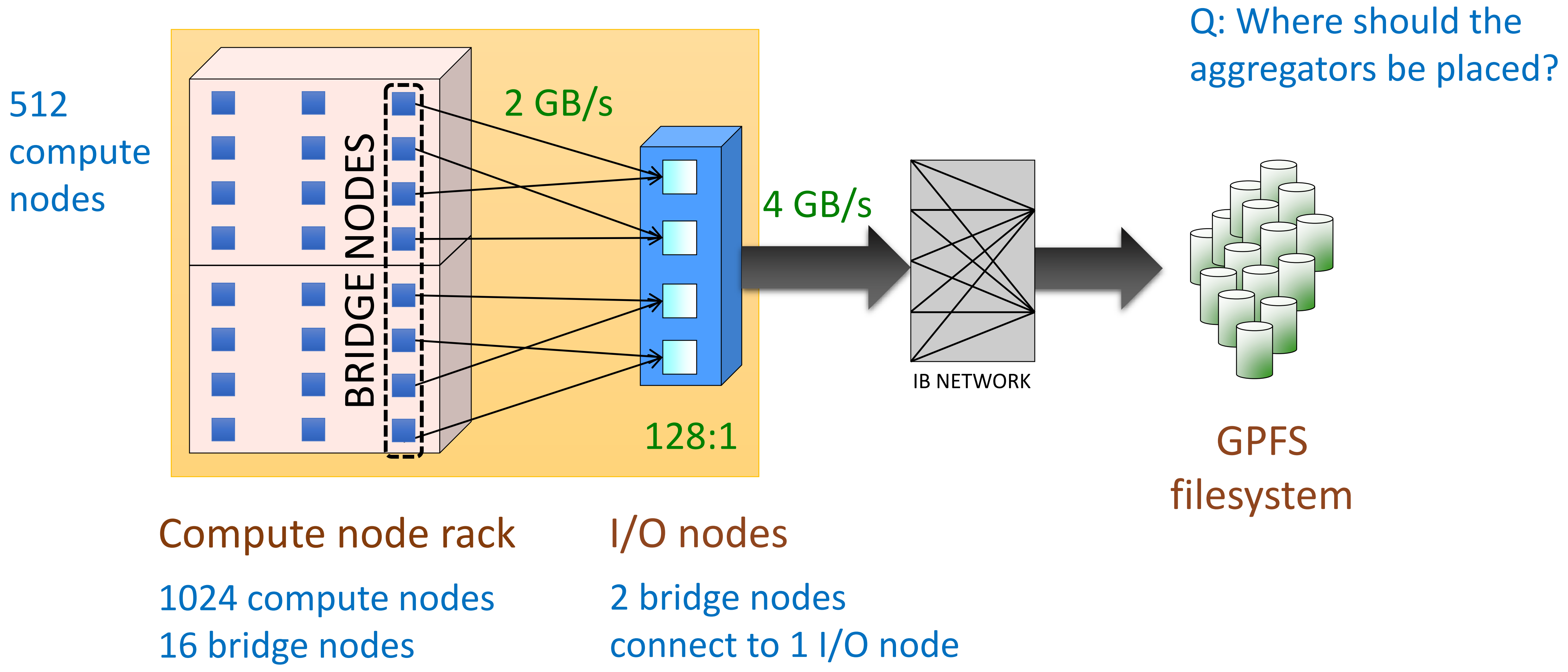
IBM Blue Gene/P

- Compute nodes are partitioned into subsets that map to an I/O node
- Compute node kernel forwards all I/O and socket requests to the I/O node
- A dedicated control and I/O daemon running on the I/O node performs I/O on behalf of the compute nodes
- I/O forwarding can potentially reduce the file system traffic by aggregating, caching the I/O requests at the I/O nodes



Source: Ali et al., “Scalable I/O Forwarding Framework for High-Performance Computing Systems”

BG/Q – I/O Node Architecture

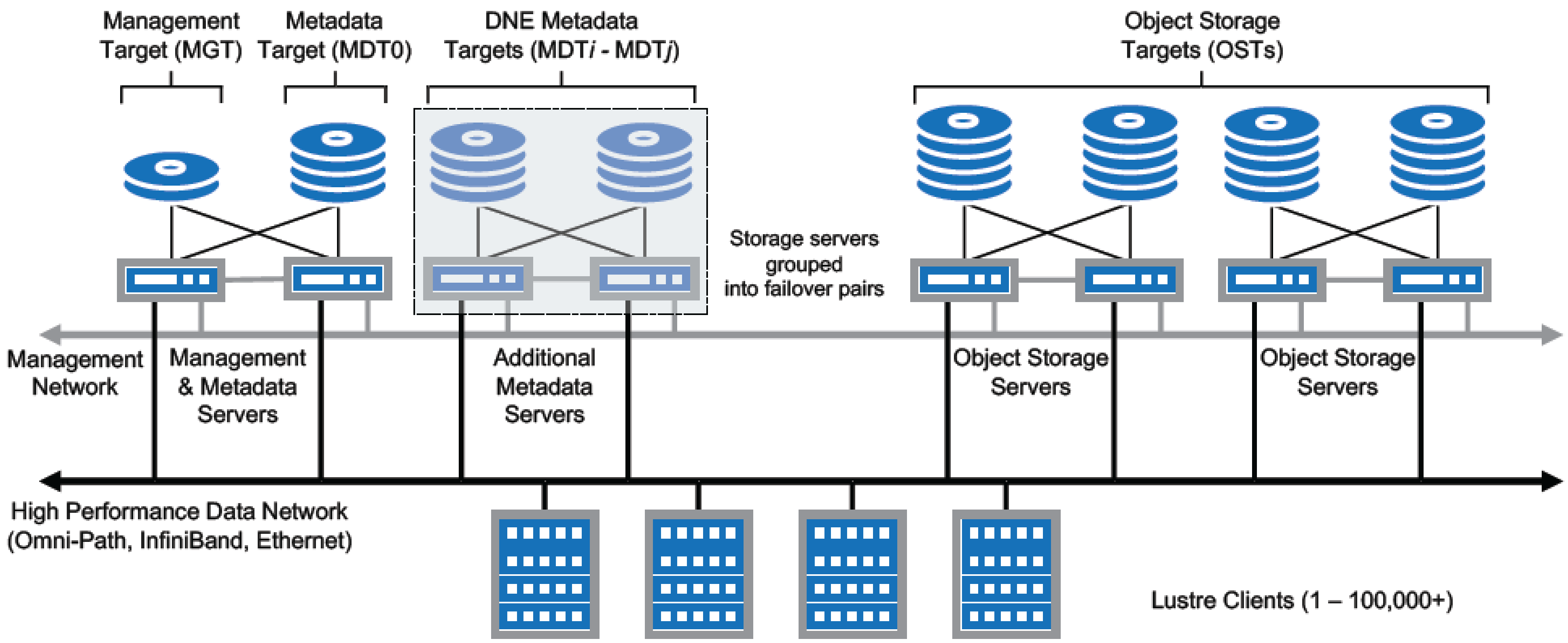


Lustre File System

- Parallel file system
- Used in 15/30 top 500 supercomputers
- POSIX-compliant file system
  - presents a unified file system interface to the user
- Object-based filesystem
  - "A storage object is a logical collection of bytes on a storage device" [1]
  - Composed of data, attributes, metadata
  - Files distributed across multiple objects
- Scalability due to object storage and division of labor
- No file server bottleneck

[1] Mesnier et al., Object-Based Storage, IEEE Communications Magazine, 2003

Lustre Scalable Storage Architecture



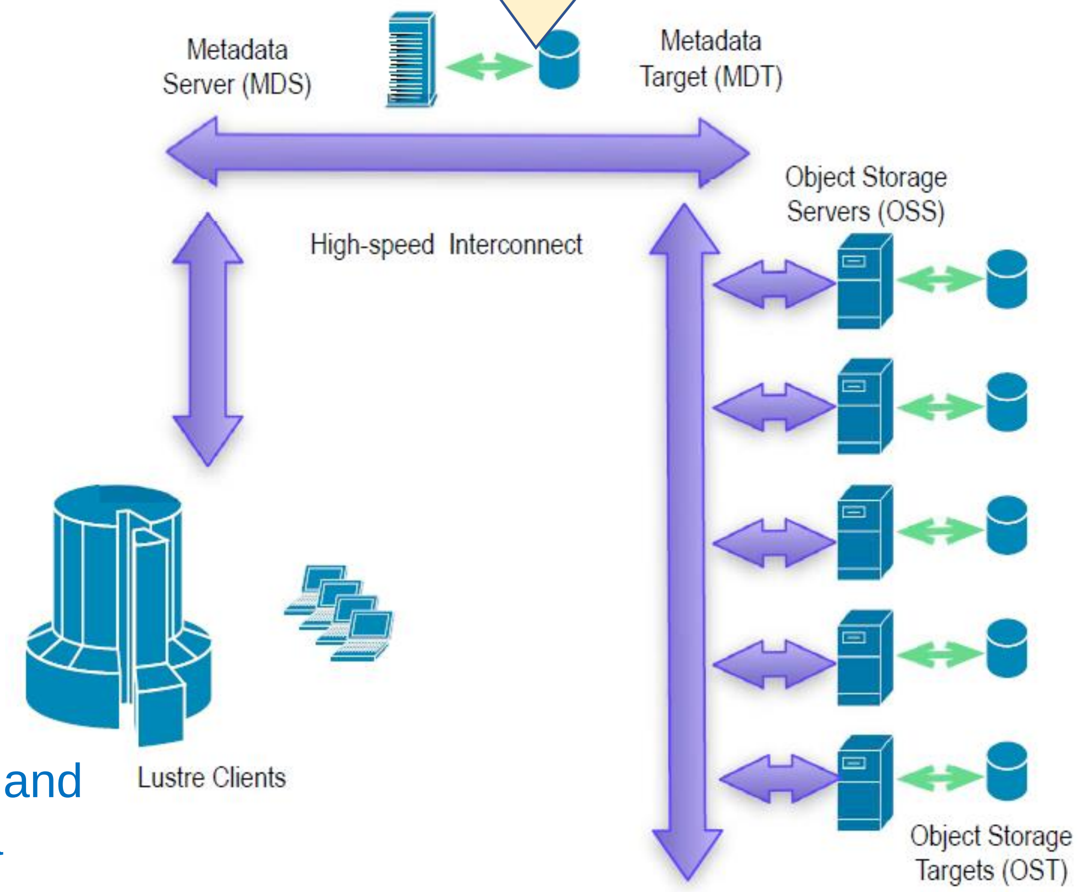
"Lustre can deliver more than a terabyte-per-second of combined throughput." -- http://wiki.lustre.org/images/6/64/LustreArchitecture-v4.pdf

Lustre

Three components

- Metadata servers (MDS)
- Object storage servers (OSS)
  - Object storage targets (OST)
- Clients

Stores file metadata, such as file names, directory structures, and access permissions



Lustre clients access and concurrently use data through the standard POSIX I/O system calls.

Source: Understanding Lustre Internals

Lustre Components

Metadata Server (MDS)

- File operations (create, open, read etc.) require metadata stored on MDS
- Handles metadata requests - file lookups, file and directory attribute manipulation
- Maintains a transactional record of file system changes

Object Storage Server (OSS) and Object Storage Targets (OST)

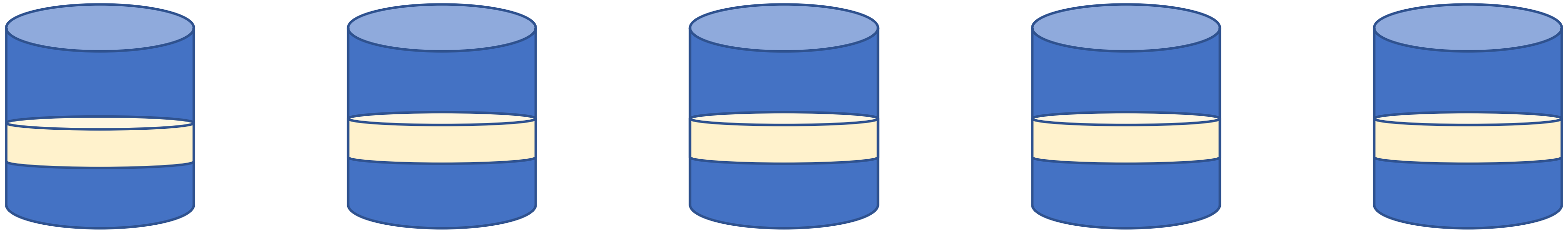
- Each file is composed of data objects striped on one or more OSTs
- Responsible for actual file system I/O
- Responsible for interfacing with storage devices

Lustre Components

Lustre Client

- Queries MDS
- Retrieves the list of OSTs
- Sends request to the OSTs

Lustre Striping



- Stripe size
- Stripe count/width

Obj 1 => OST A
Obj 2 => OST B
Obj 3 => OST C
Obj 4 => OST D
Obj 5 => OST E

Lustre Striping Example

```
[pmalakar@cn364 testq]$ lfs setstripe -c 10 testmpiio.out
[pmalakar@cn364 testq]$ lfs getstripe testmpiio.out
testmpiio.out
lmm_stripe_count:   10
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  1
        obdidx           objid           objid            group
             1         4673479        0x474fc7                0
             0         4600893        0x46343d                0
            20         4551236        0x457244                0
             3         4701254        0x47bc46                0
            21         4479152        0x4458b0                0
            19         4696884        0x47ab34                0
             5         4704057        0x47c739                0
             7         4647142        0x46e8e6                0
            16         4640736        0x46cfe0                0
            18         4595400        0x461ec8                0
```

Example: File striped across 10 OSTs. Each OST stores 1 MB objects.

Lustre Striping Parameters

lfs setstripe –S <size> -c <count> filename

```
[pmalakar@cn364 testq]$ rm testmpiio.out
[pmalakar@cn364 testq]$ time dd if=/dev/zero of=testmpiio.out bs=10M count=1000
1000+0 records in
1000+0 records out
10485760000 bytes (10 GB) copied, 18.2025 s, 576 MB/s

real    0m18.205s
user    0m0.004s
sys     0m10.042s
[pmalakar@cn364 testq]$ rm testmpiio.out
[pmalakar@cn364 testq]$ lfs setstripe -S 2M -c 10 testmpiio.out
[pmalakar@cn364 testq]$ time dd if=/dev/zero of=testmpiio.out bs=10M count=1000
1000+0 records in
1000+0 records out
10485760000 bytes (10 GB) copied, 10.4116 s, 1.0 GB/s

real    0m10.420s
user    0m0.003s
sys     0m10.406s
```

Striping Benefit

8 MB

```
Time 0.010138
Time 0.013419
Time 0.027182
Time 0.075958
Time 0.219819
Time 0.333267
```

256 MB

Stripe count = 1

8 MB

```
Time 0.020716
Time 0.025181
Time 0.035053
Time 0.063688
Time 0.220986
Time 0.223855
```

256 MB

Stripe count = 18

Striping Benefit

1.5 GB

```
Time 0.102997
Time 0.099457
Time 0.245411
Time 0.439744
Time 0.692367
Time 0.954562
```

Stripe count = 18

1.5 GB

```
Time 0.053866
Time 0.158203
Time 0.330782
Time 0.744611
Time 1.399156
Time 2.390323
```

Stripe count = 1