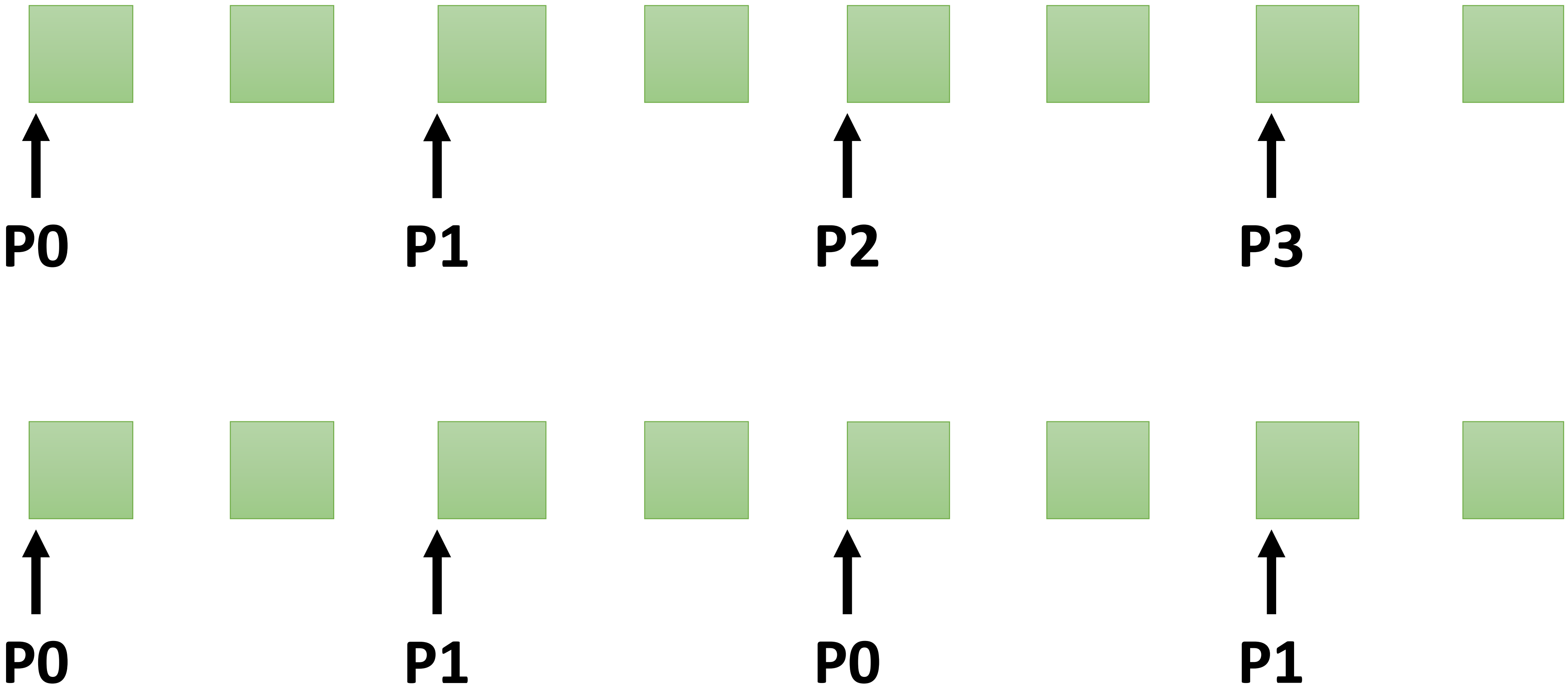
Parallel I/O – III

Lecture 17

Mar 18, 2024

Access Pattern



Collective I/O – Selecting Aggregators

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

Motivation: Default parameters lead to sub-optimal I/O performance

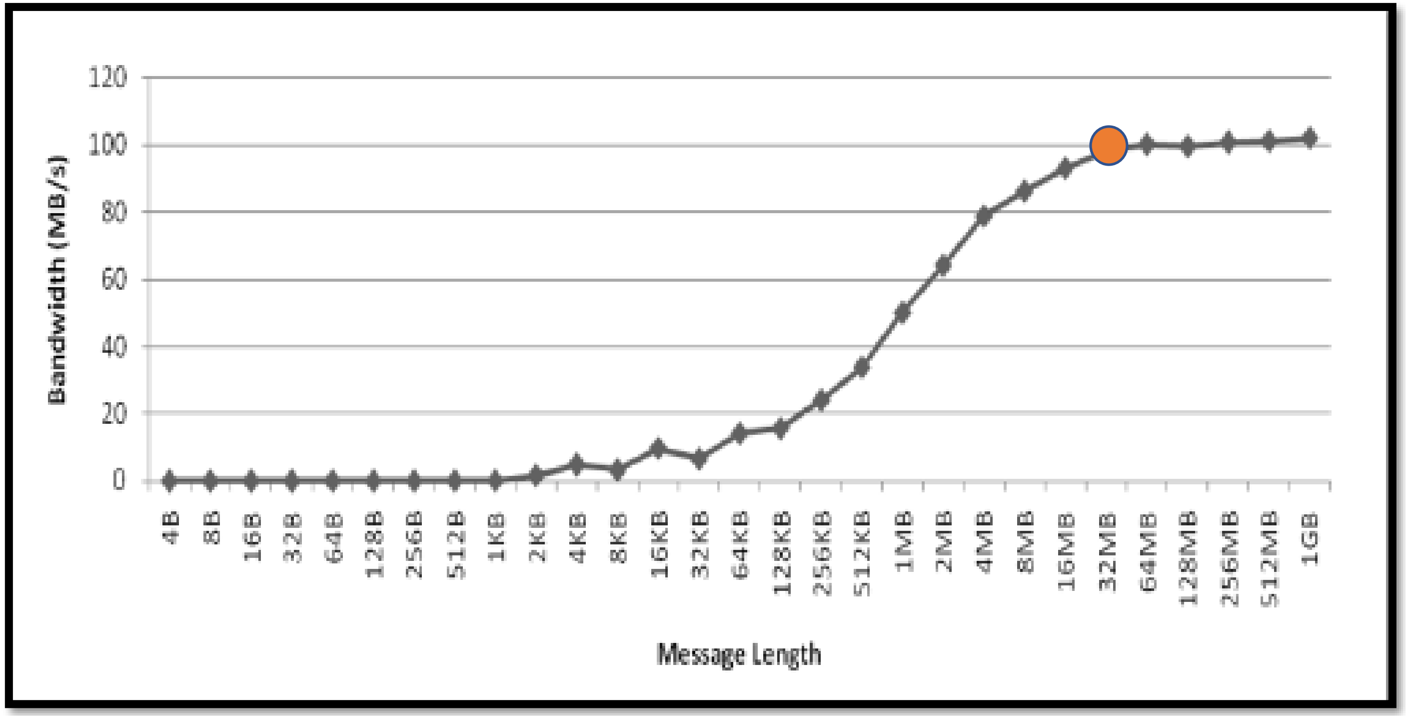
Find an optimal number of aggregators such that each aggregator is saturating the file system from the processes perspective, but not overloading the aggregator.

Collective I/O – Selecting Aggregators

Select

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

Tuning for Bandwidth Saturation



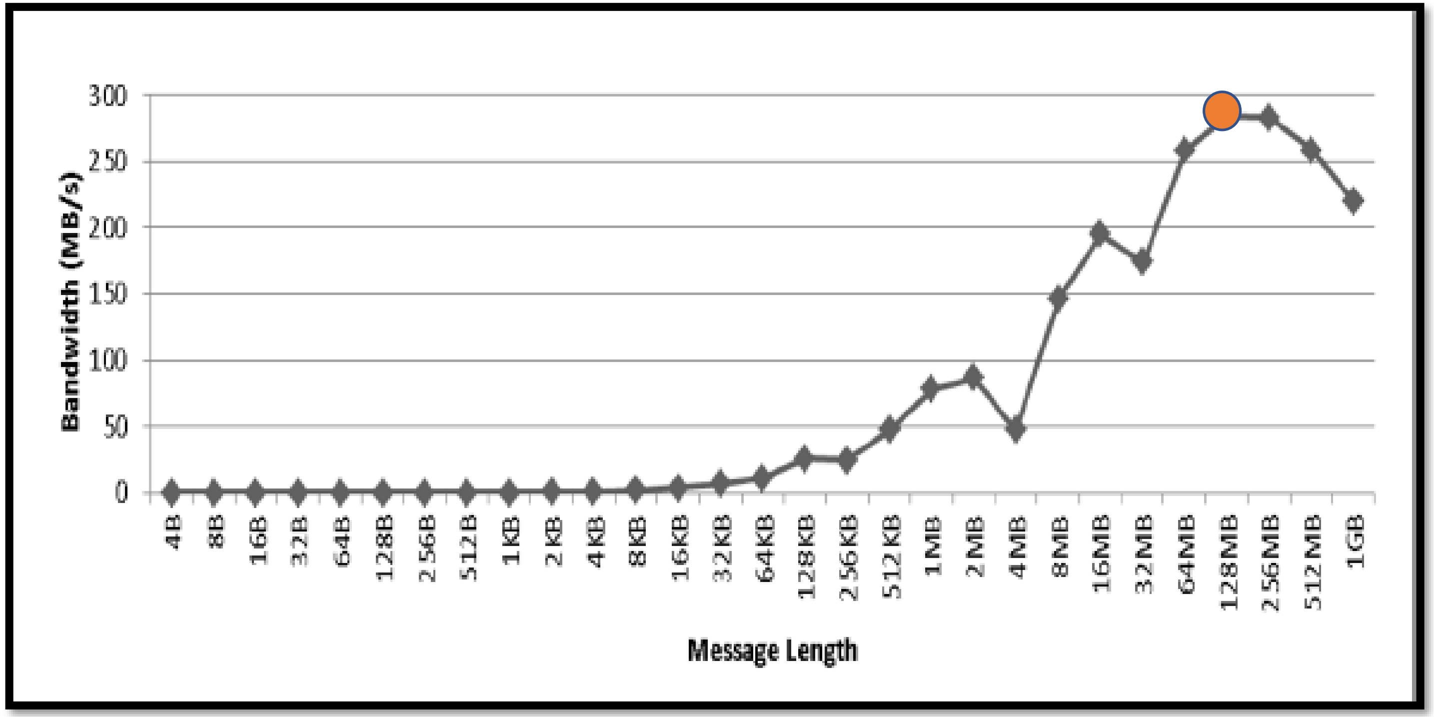
Q: What are the factors?

Shark

- 29 nodes
- SDR Inifiniband and Gigabit Ethernet
- PVFS2

- Objective: Find write saturation point k (done once)
- Motivation: Prevent underutilization of resources

Tuning for Bandwidth Saturation



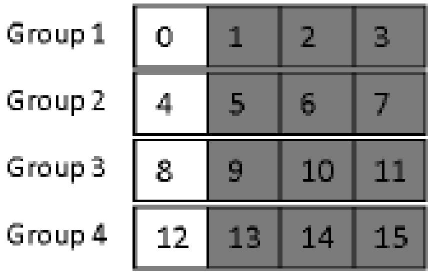
Deimos

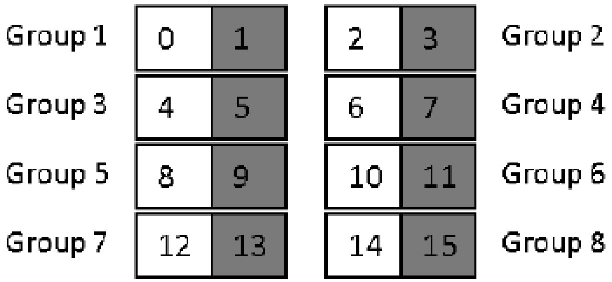
- 724 nodes
- 11 I/O servers via SDR Inifiniband
- Lustre

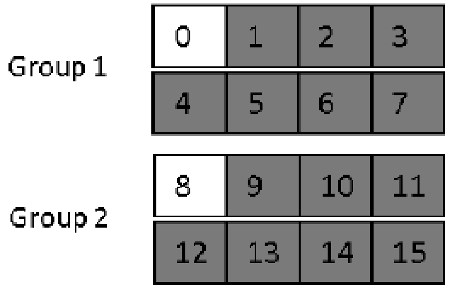
Collective I/O – Selecting Aggregators

Select

- Data size k (write saturation point)
- Initial #aggregators
  - Consider process topology
- Refine #aggregators
  - If total bytes per group > k, then split
  - If total bytes per group < k, then join







Collective I/O – Selecting Aggregators

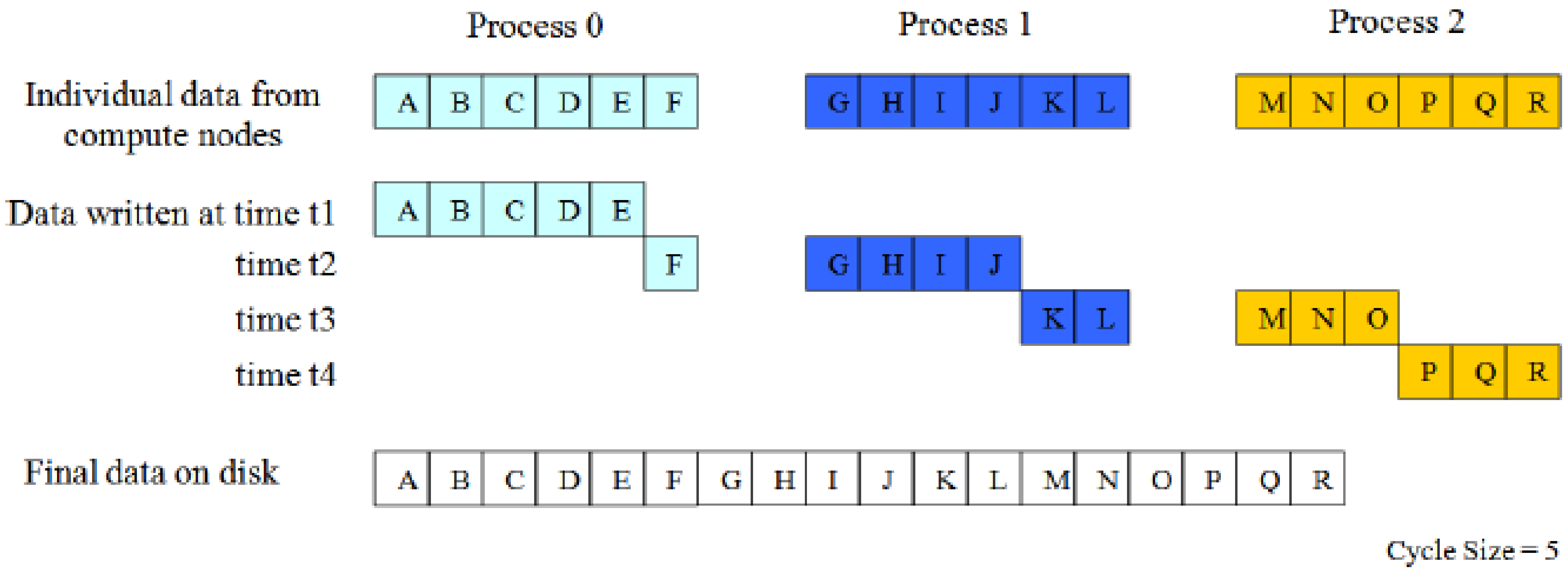
Approaches for collective I/O aggregation

- Two-phase
  - Shuffle phase
  - I/O phase

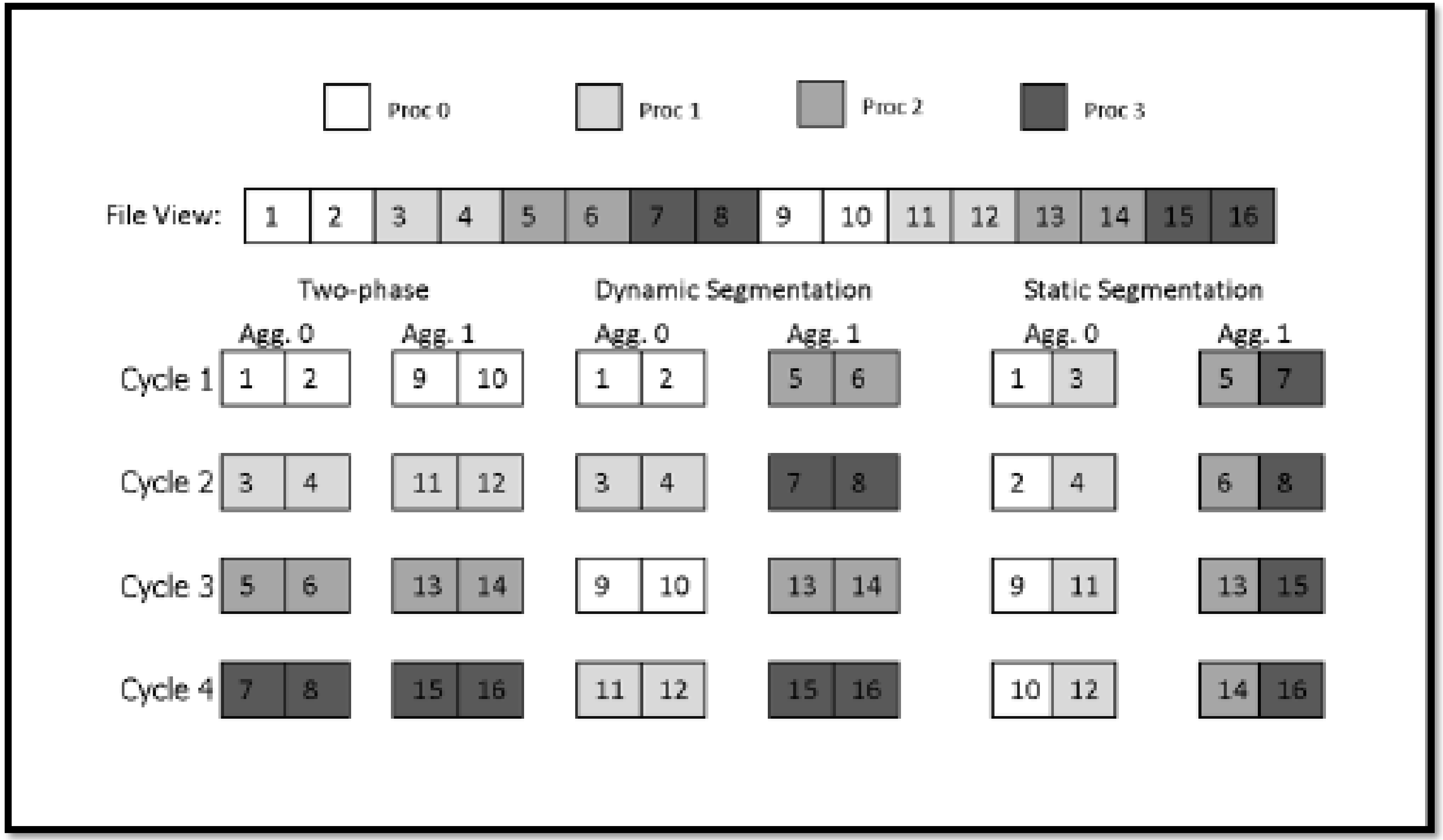
Motivation: reduce shuffle step overhead [*Performance Evaluation of Collective Write Algorithms in MPI I/O, ICCS 2009*]

- Dynamic segmentation
  - Fixed #processes per aggregator
- Static segmentation
  - Extension of dynamic segmentation
  - Fixed #bytes per cycle per process per aggregator

Dynamic Segmentation – Motivation



Collective I/O – Selecting Aggregators



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

Advantage of Dynamic and Static

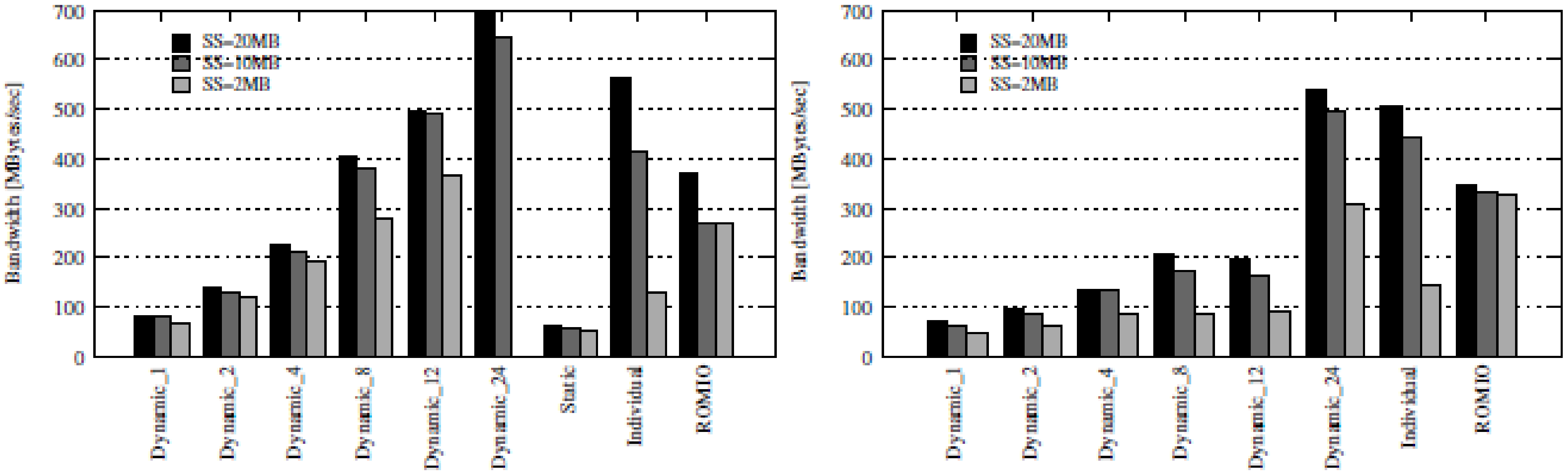
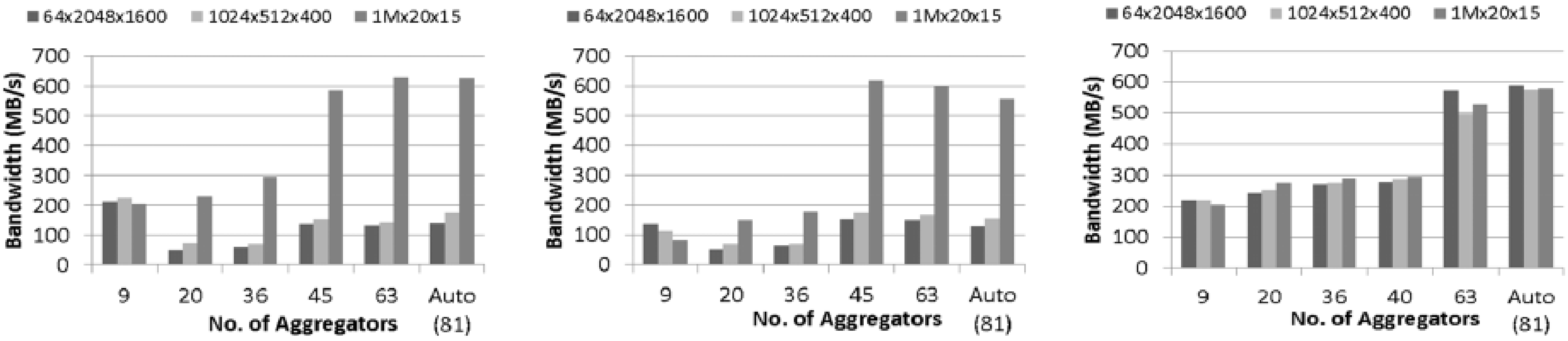


Fig. 3. Performance Comparison for 24 processes (left) and 48 processes (right) with varying the segment size and keeping the cycle buffer size constant at 20MB

Dynamic vs. Static vs. Two-phase on Shark

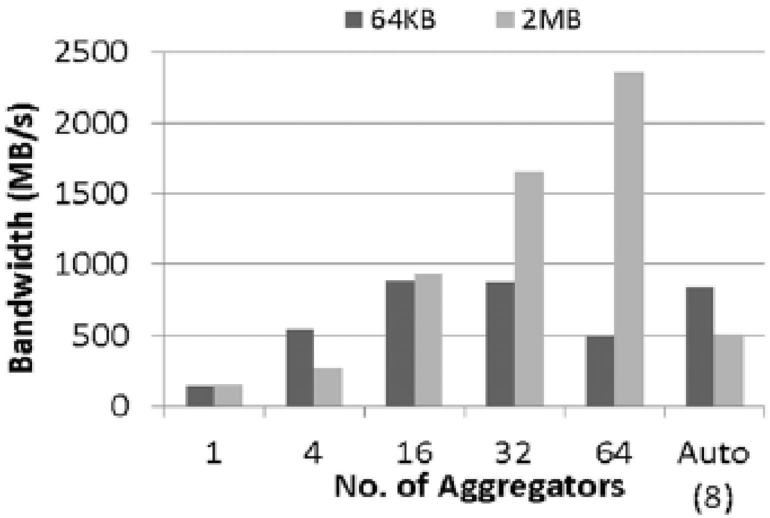
The number of aggregators chosen is equal to the total number of processes, because each process is requesting to write data larger than the value of *k* per function call.

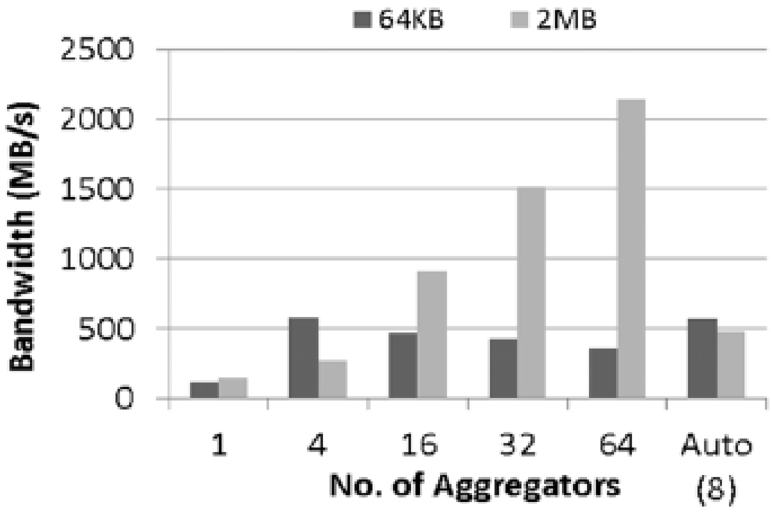


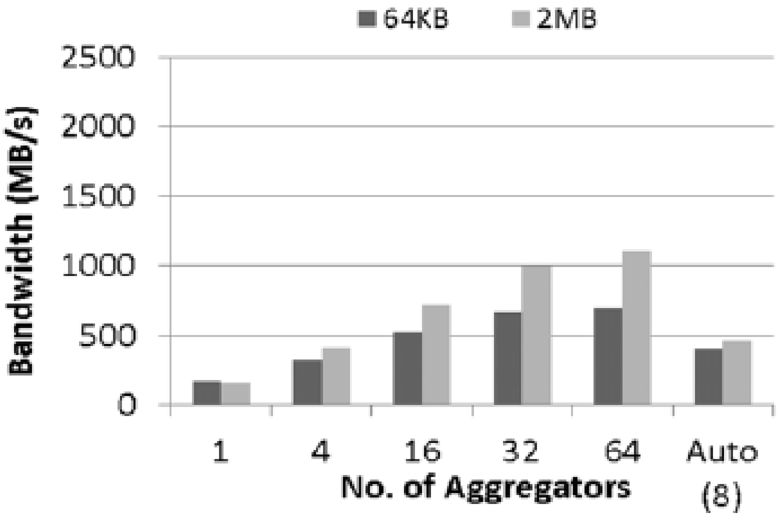
MPI Tile IO benchmark with 81 (9 X 9) processes: Dynamic | Static | Two-phase

Dynamic vs. Static vs. Two-phase on Deimos

- The number of aggregators chosen was good enough for small segment size
- 2 MB being a multiple of stripe size did not affect performance



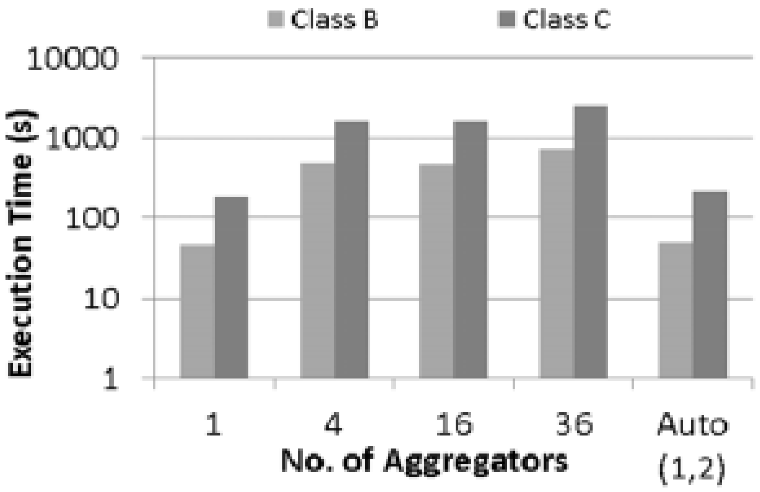


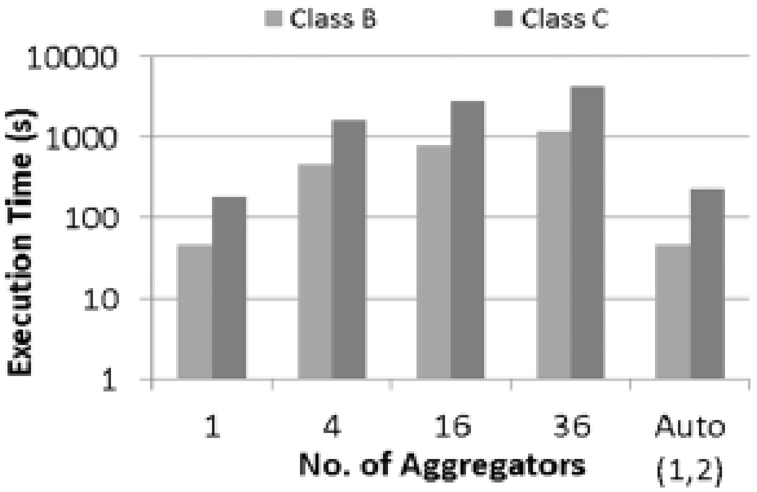


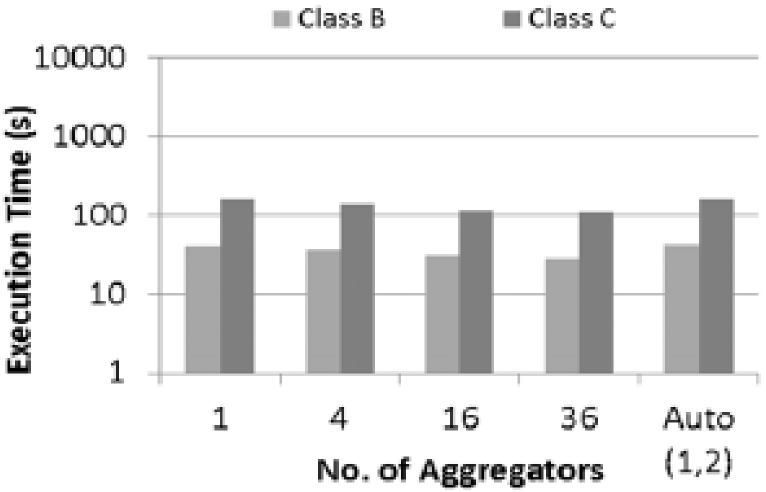
Latency IO benchmark with 64 processes (Segment sizes: 64 KB and 2 MB)

Dynamic vs. Static vs. Two-phase on Shark

The optimal number of aggregators was small in this case

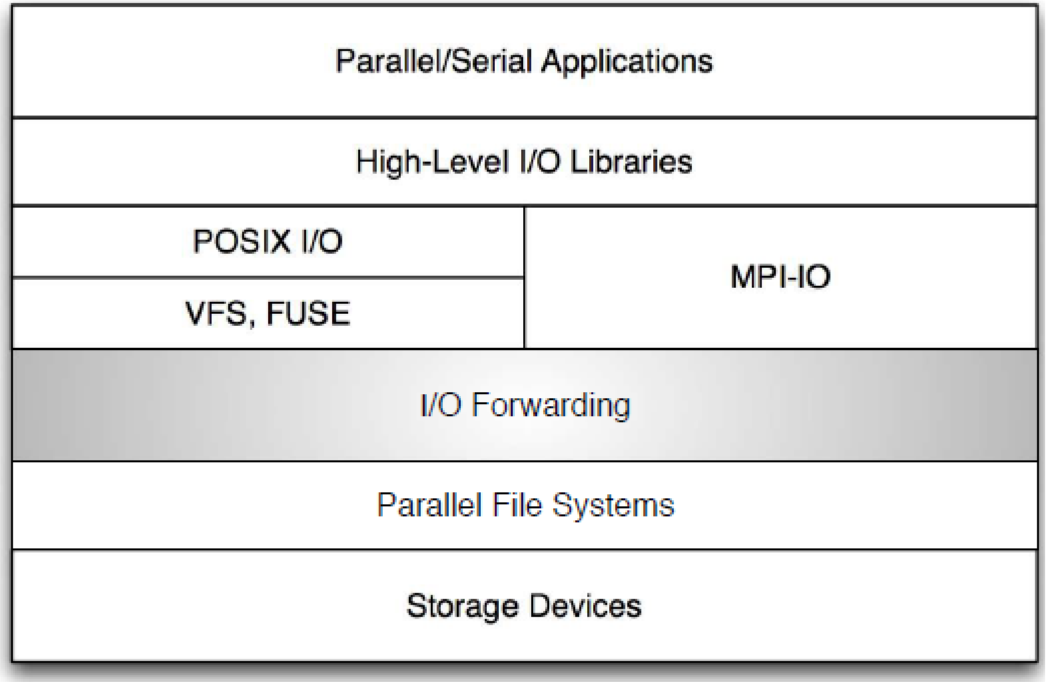






BT-IO benchmark on 36 processes

I/O Forwarding

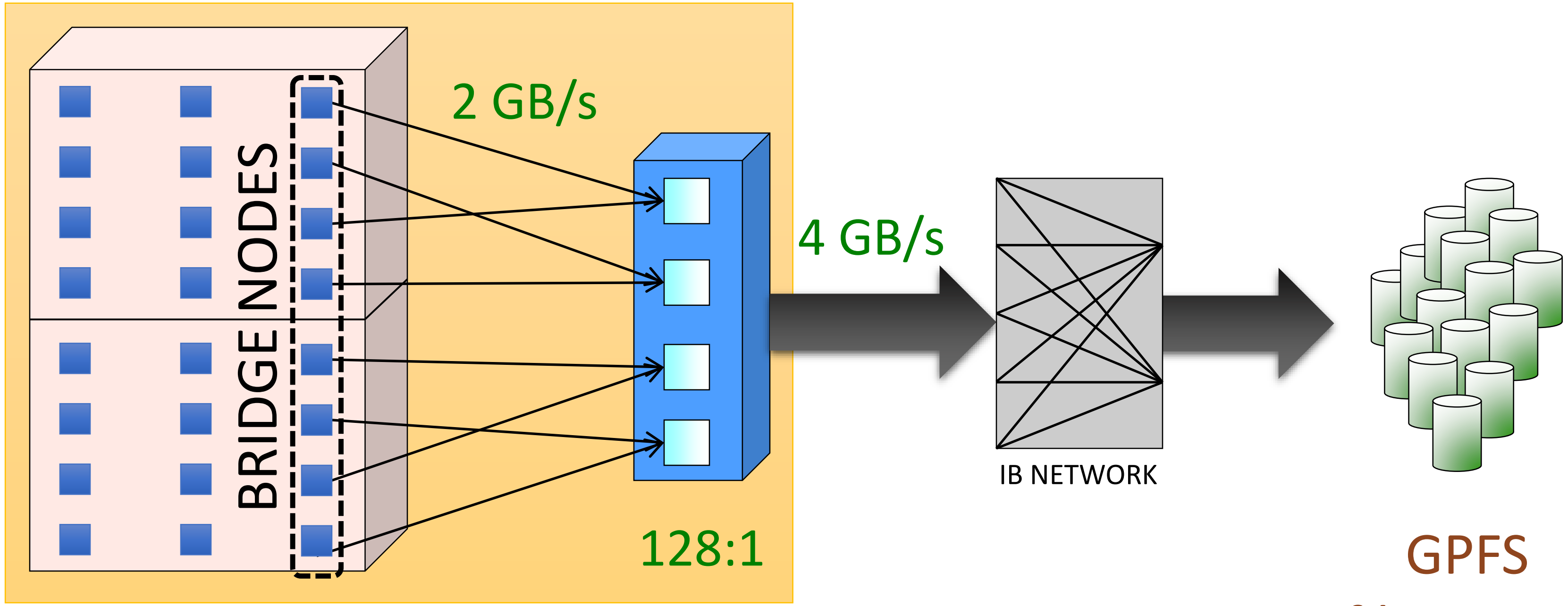


Source: Ohta et al., “Optimization Techniques at the I/O Forwarding Layer”

- I/O requests forwarded to dedicated I/O nodes by compute nodes
- 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

BG/Q – I/O Node Architecture

512
compute
nodes



GPFS
filesystem

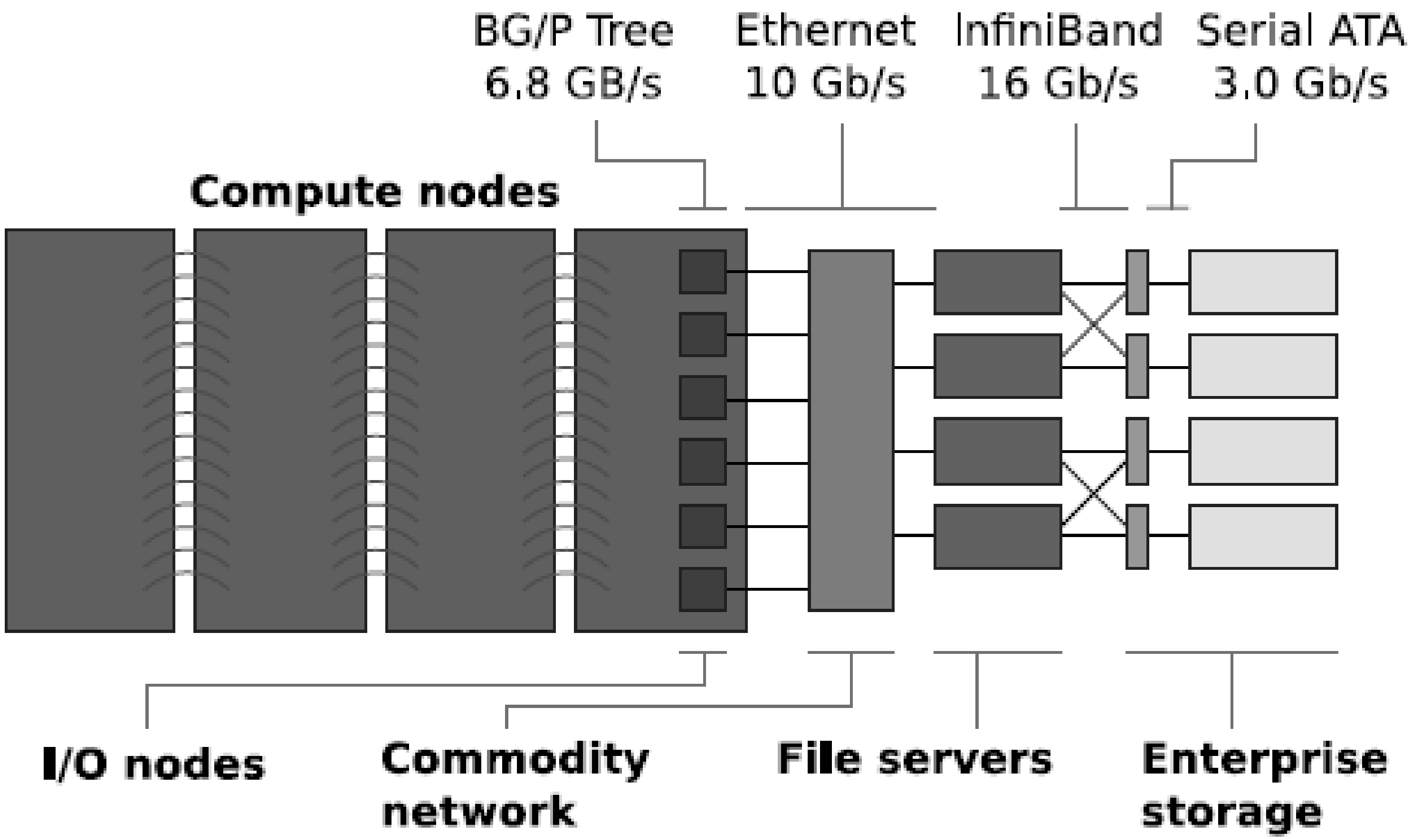
Compute node rack

1024 compute nodes
16 bridge nodes

I/O nodes

2 bridge nodes
connect to 1 I/O node

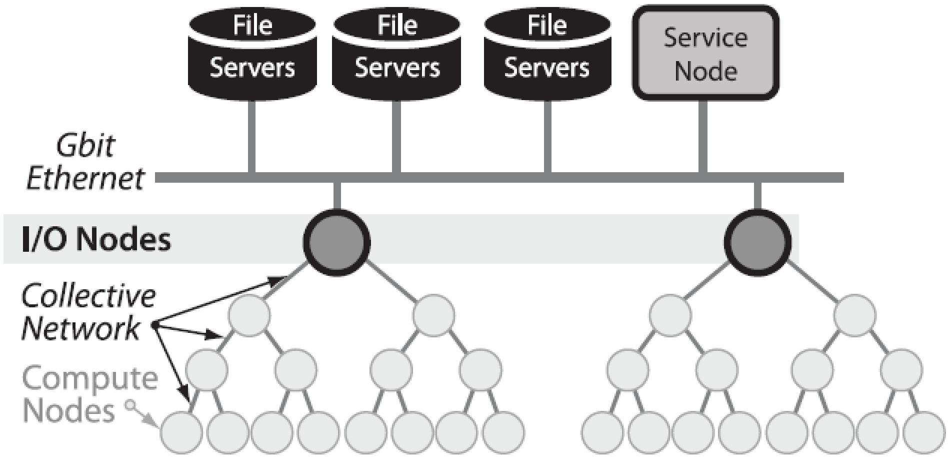
IBM Blue Gene/P Network



24/7 Characterization of Petascale I/O Workloads, Carns et al., Cluster 2009

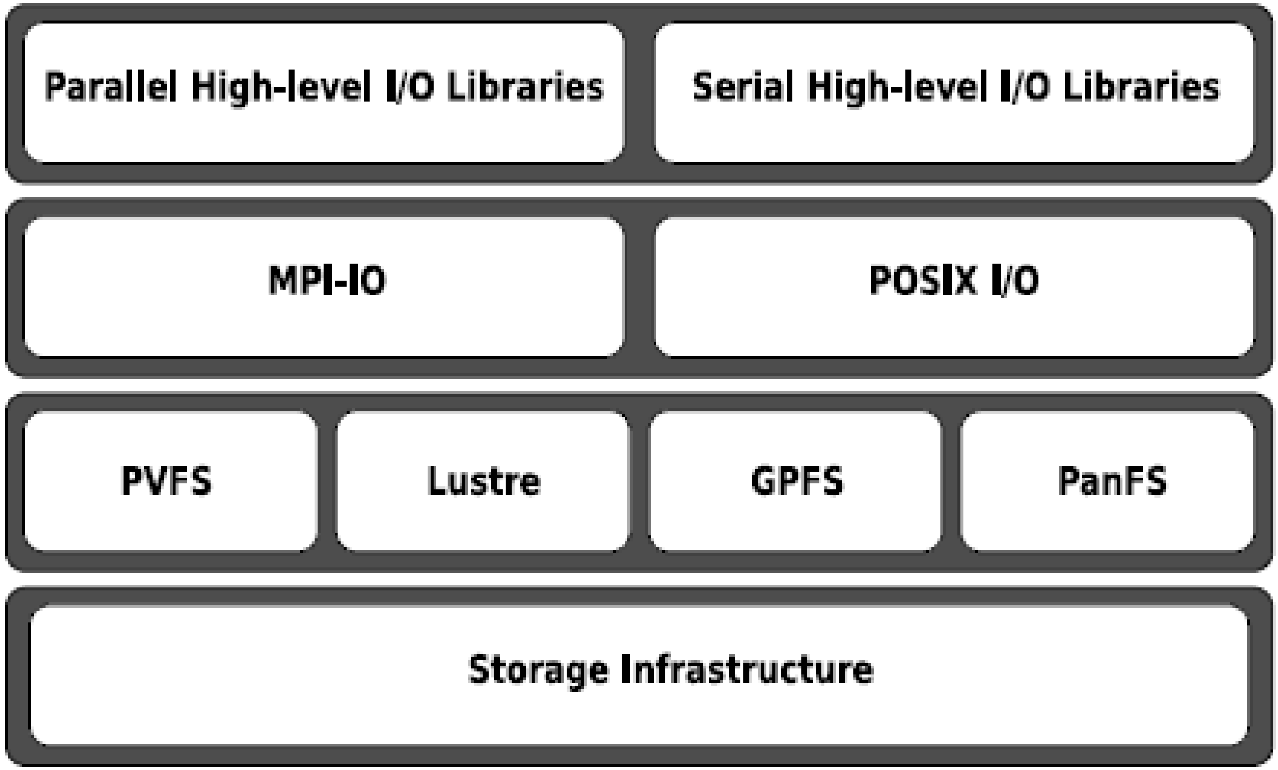
IBM Blue Gene/P

- 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”

I/O Stack

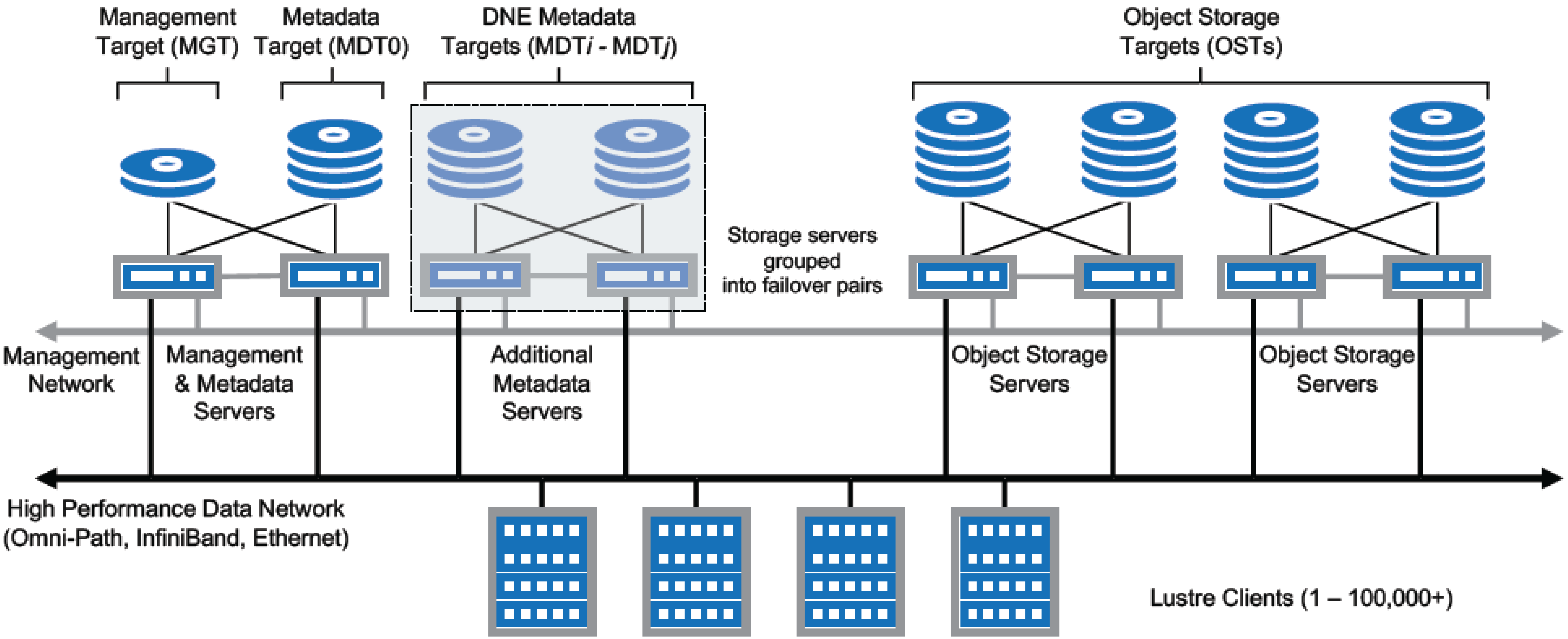

Parallel High-level I/O Libraries
Serial High-level I/O Libraries
MPI-IO
POSIX I/O
PVFS
Lustre
GPFS
PanFS
Storage Infrastructure

Lustre File System

- Parallel file system
- Used in several 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



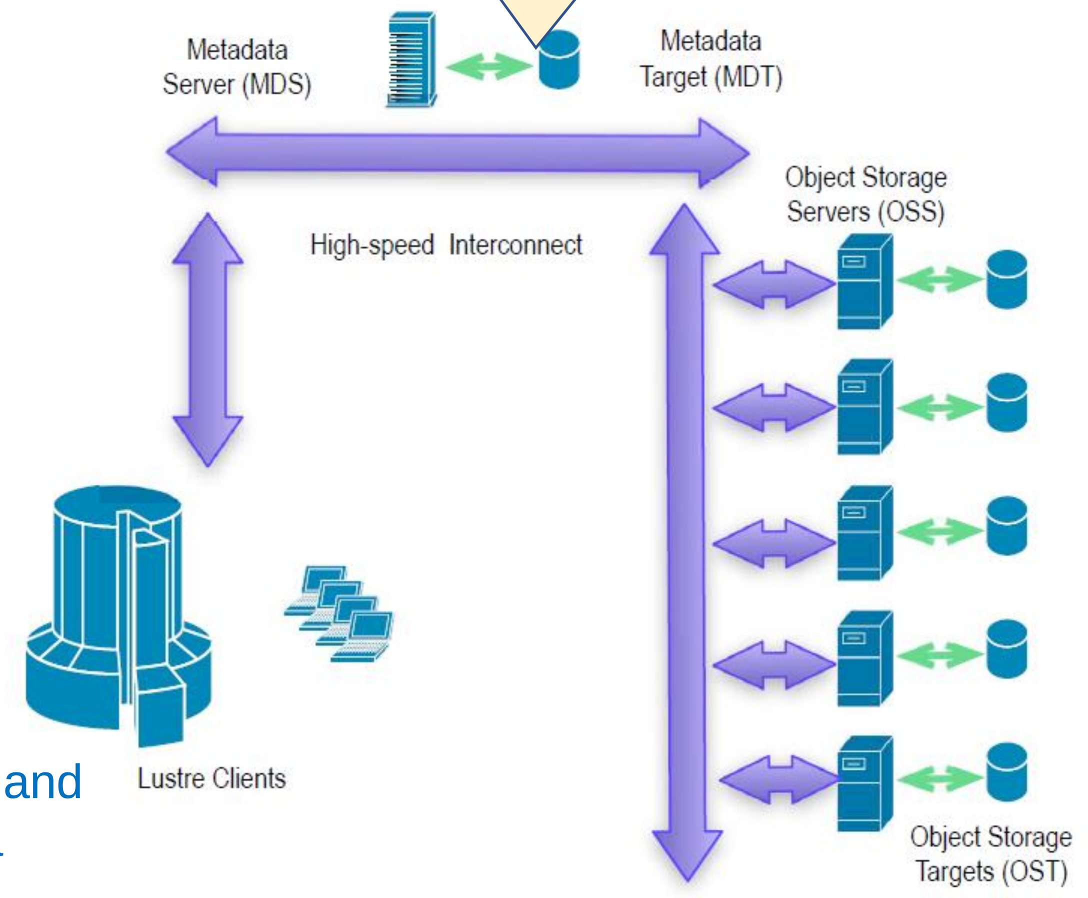
"Lustre can deliver more than a terabyte-per-second of combined throughput." -- http://wiki.lustre.org/images/6/64/LustreArchitecture-v4.pdf

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

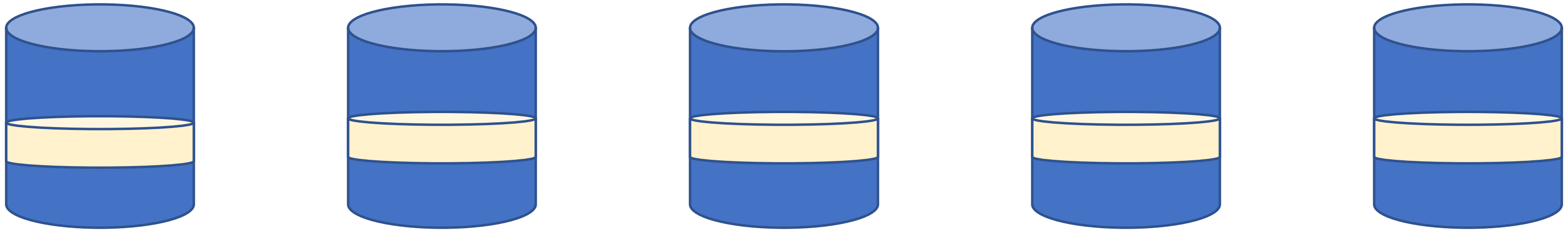
Object Storage Server (OSS) and Object Storage Targets (OST)

- 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 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

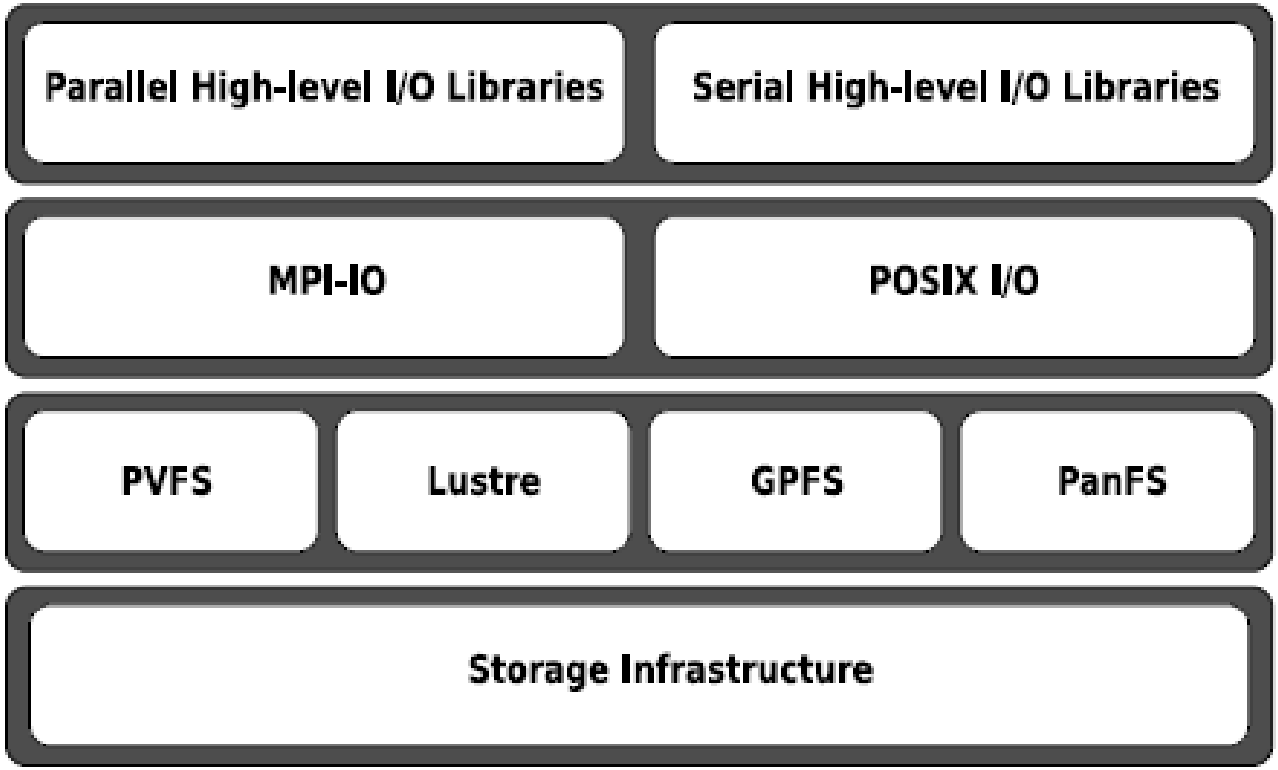
Stripe count = 1

8 MB	Time 0.010138
	Time 0.013419
	Time 0.027182
	Time 0.075958
	Time 0.219819
256 MB	Time 0.333267

Stripe count = 18

8 MB	Time 0.020716
	Time 0.025181
	Time 0.035053
	Time 0.063688
	Time 0.220986
256 MB	Time 0.223855

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

Adapted from

https://extremecomputingtraining.anl.gov/wp-content/uploads/sites/96/2019/08/ATPESC_2019_Track-3_5_8-2_1115am_Latham-Higher-level_IO_Libraries.pdf

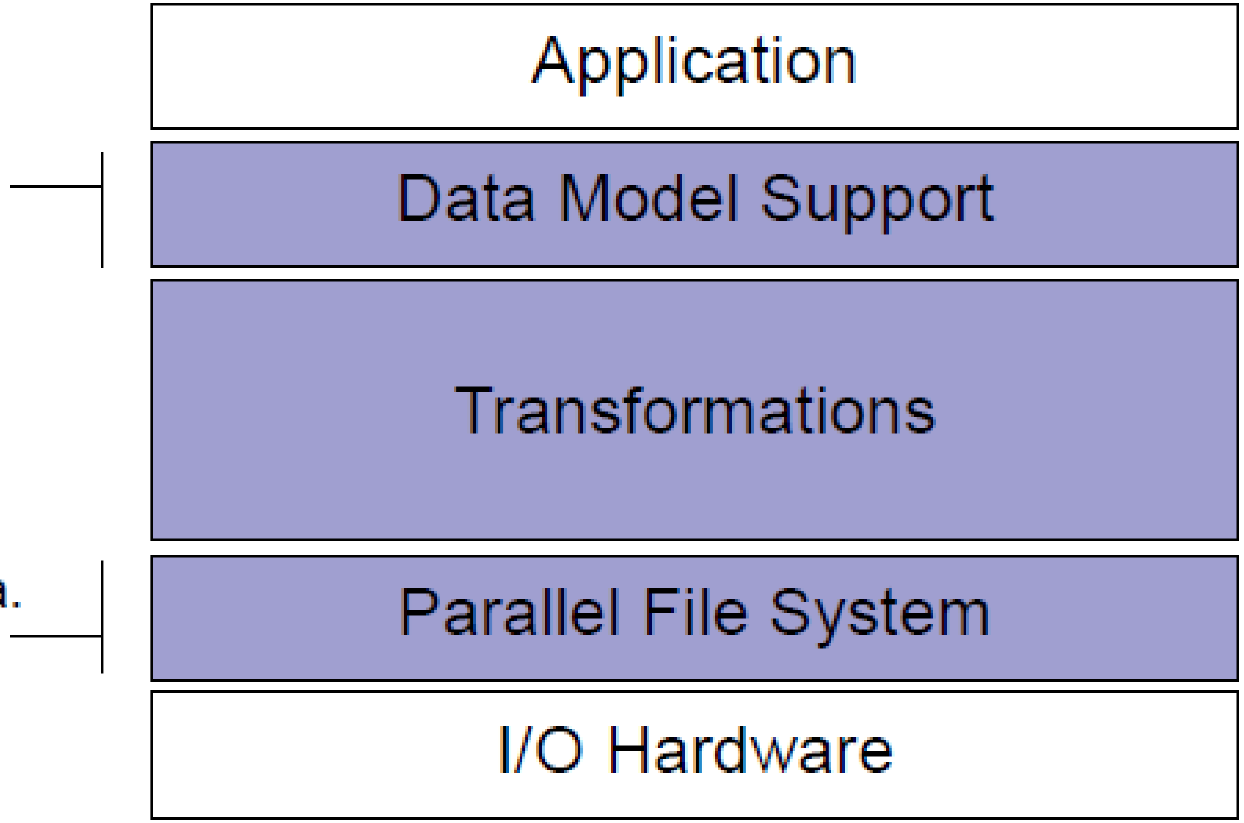
I/O Stack

Data Model Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file system maintains logical file model and provides efficient access to data.

PVFS, PanFS, GPFS, Lustre



I/O Libraries

Scientific applications work with structured data and desire more self-describing file formats

- PnetCDF and HDF5 are two popular high-level I/O libraries

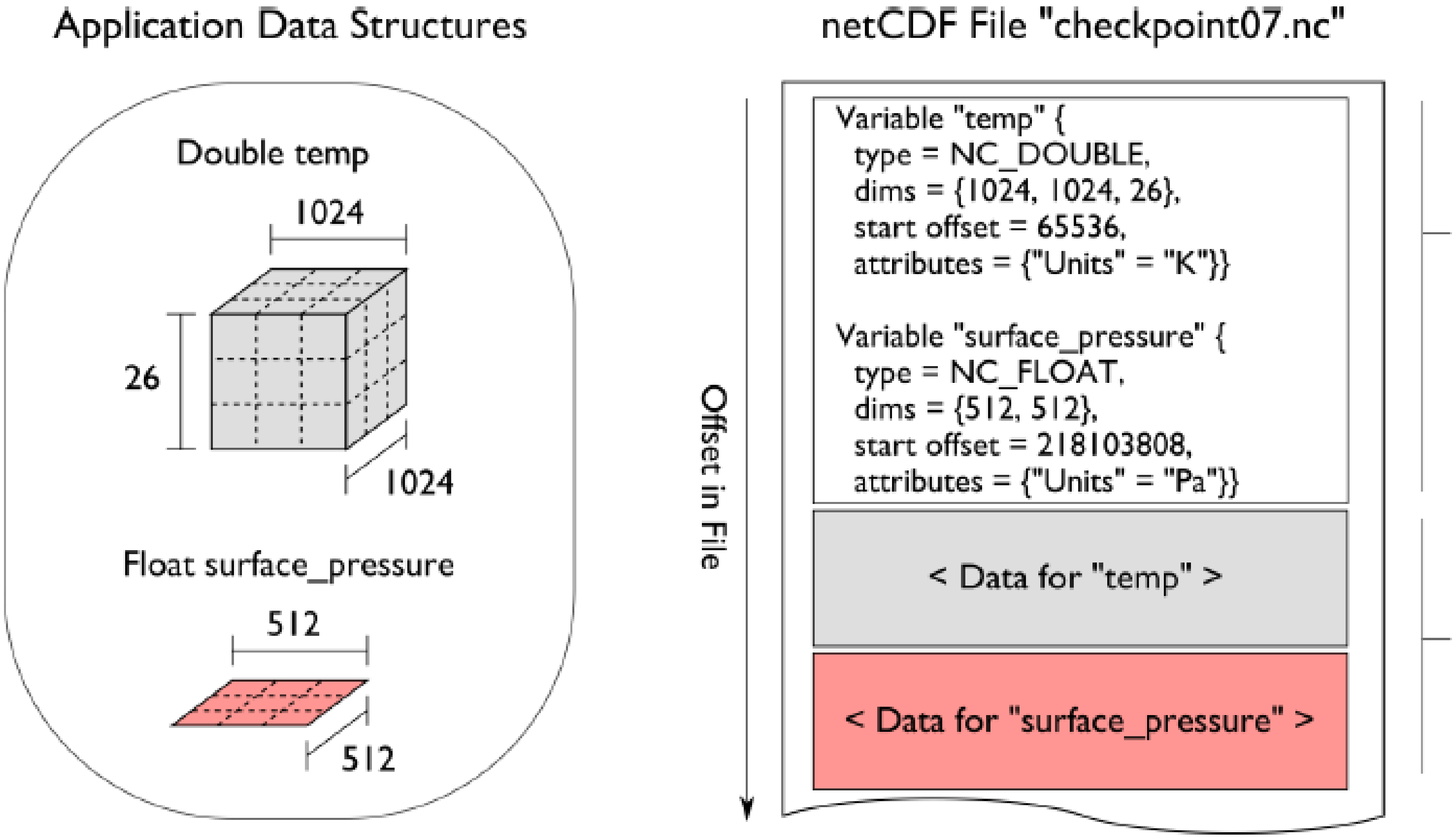
– Abstract away details of file layout

– Provide standard, portable file formats

- For parallel machines, these use MPI and probably MPI-IO

–MPI-IO implementations are sometimes poor on specific platforms, in which case libraries might directly call POSIX calls instead

NetCDF Data Model



netCDF header describes the contents of the file: typed, multi-dimensional variables and attributes on variables or the dataset itself.

Data for variables is stored in contiguous blocks, encoded in a portable binary format according to the variable's type.

NetCDF File Format

```
netcdf d01_2009-05-23_00_00_00 {
dimensions:
        Time = 1 ;
        DateStrLen = 19 ;
        west_east = 303 ;
        south_north = 216 ;
        bottom_top = 27 ;
        bottom_top_stag = 28 ;
        soil_layers_stag = 4 ;
        west_east_stag = 304 ;
        south_north_stag = 217 ;
variables:
        char Times(Time, DateStrLen) ;
        float LU_INDEX(Time, south_north, west_east) ;
                LU_INDEX:FieldType = 104 ;
                LU_INDEX:MemoryOrder = "XY " ;
                LU_INDEX:description = "LAND USE CATEGORY" ;
                LU_INDEX:units = "" ;
                LU_INDEX:stagger = "" ;
                LU_INDEX:coordinates = "XLONG XLAT" ;
```

NetCDF File Format

```
class ncdump -v PSFC d01_2009-05-23_00_00_00.nc | grep "float "
        float LU_INDEX(Time, south_north, west_east) ;
        float ZNU(Time, bottom_top) ;
        float ZNW(Time, bottom_top_stag) ;
        float ZS(Time, soil_layers_stag) ;
        float DZS(Time, soil_layers_stag) ;
        float U(Time, bottom_top, south_north, west_east_stag) ;
        float V(Time, bottom_top, south_north_stag, west_east) ;
        float W(Time, bottom_top_stag, south_north, west_east) ;
        float PH(Time, bottom_top_stag, south_north, west_east) ;
        float PHB(Time, bottom_top_stag, south_north, west_east) ;
        float T(Time, bottom_top, south_north, west_east) ;
        float MU(Time, south_north, west_east) ;
        float MUB(Time, south_north, west_east) ;
        float NEST_POS(Time, south_north, west_east) ;
        float P(Time, bottom_top, south_north, west_east) ;
        float PB(Time, bottom_top, south_north, west_east) ;
        float SR(Time, south_north, west_east) ;
        float POTEVP(Time, south_north, west_east) ;
        float SNOPCX(Time, south_north, west_east) ;
        float SOILTB(Time, south_north, west_east) ;
        float FNM(Time, bottom_top) ;
        float FNP(Time, bottom_top) ;
        float RDNW(Time, bottom_top) ;
        float RDN(Time, bottom_top) ;
        float DNW(Time, bottom_top) ;
        float DN(Time, bottom_top) ;
```

NetCDF File Format

```
class ncdump -v PSFC d01_2009-05-23_00_00_00.nc | grep -A 20 "float PSFC"
        float PSFC(Time, south_north, west_east) ;
                PSFC:FieldType = 104 ;
                PSFC:MemoryOrder = "XY " ;
                PSFC:description = "SFC PRESSURE" ;
                PSFC:units = "Pa" ;
                PSFC:stagger = "" ;
                PSFC:coordinates = "XLONG XLAT" ;
        float U10(Time, south_north, west_east) ;
                U10:FieldType = 104 ;
                U10:MemoryOrder = "XY " ;
                U10:description = "U at 10 M" ;
                U10:units = "m s-1" ;
                U10:stagger = "" ;
                U10:coordinates = "XLONG XLAT" ;
        float V10(Time, south_north, west_east) ;
                V10:FieldType = 104 ;
                V10:MemoryOrder = "XY " ;
                V10:description = "V at 10 M" ;
                V10:units = "m s-1" ;
                V10:stagger = "" ;
                V10:coordinates = "XLONG XLAT" ;
```

NetCDF Code Flow

Define Dimensions

ncmpi_def_dim()

Define Variables

ncmpi_def_var()

ncmpi_put_att_int()

Write Data

ncmpi_put_vara_int_all()