Parallel I/O Optimization on Lustre
Best Practices and the Future

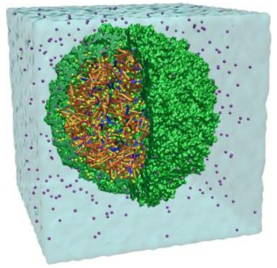
Dr. Sarah M. Neuwirth
Heidelberg University, Germany

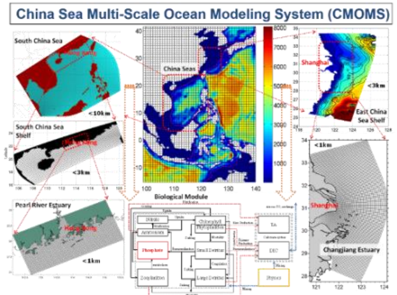
LAD’20, October 2020

Introduction to Parallel I/O
Why is Scientific I/O so difficult?

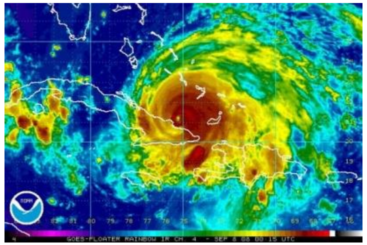
- Scientists think about data in terms of their science problem: molecules, atoms, grid cells, particles.
- Ultimately, physical disks store bytes of data.
- Layers in between, the application and physical disks are at various levels of sophistication.

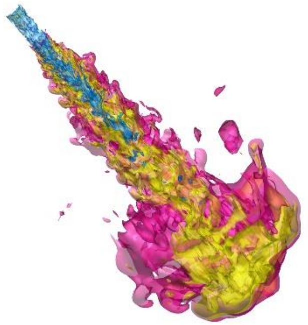
Source: *Rob Latham et al., "Parallel I/O in Practice", SC Tutorial.*


Molecular dynamics modeling of viruses.

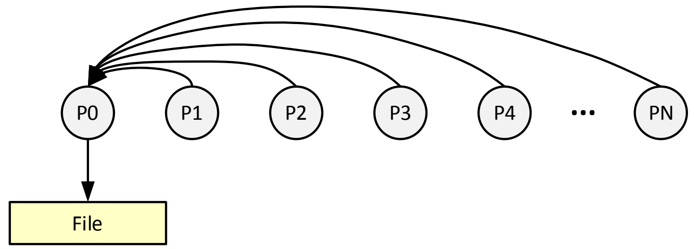


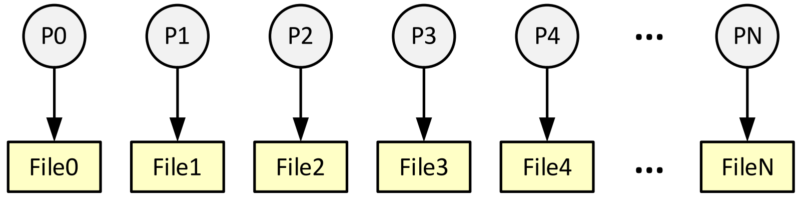
Ocean modeling (HKUST).


Weather forecasting (NOAA).


Injection process in combustion engines (ORNL).

Introduction to Parallel I/O
Sharing Patterns (I)





Single Writer (Serial) I/O:

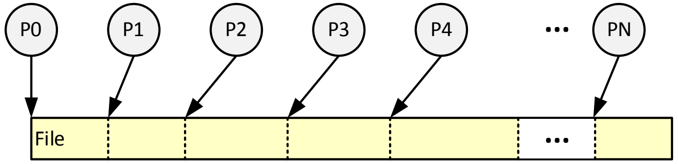
- Each task sends its data to a master that writes the data.
- Advantages
  - ✓ Simple to implement and easy to manage
- Disadvantages
  - × Scales poorly
  - × May not fit into memory on task P0
  - × Bandwidth is very limited

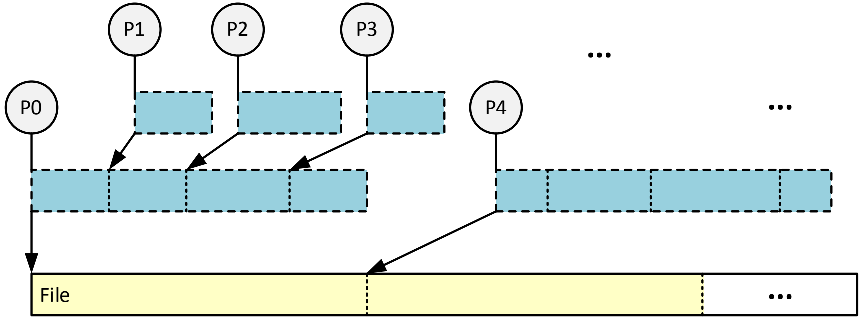
File-Per-Process I/O:

- Each task writes its data to a separate file.
- Advantages
  - ✓ Simple to program
  - ✓ Can be fast (up to a point)
- Disadvantages
  - × Can quickly accumulate many files
  - × With Lustre, MDS limit hit very fast
  - × Difficult to manage a huge number of files

Introduction to Parallel I/O
Sharing Patterns (II)

Source: *Richard Gerber, "Introduction to Parallel I/O", NERSC, August 2013.*





Single Shared File I/O:

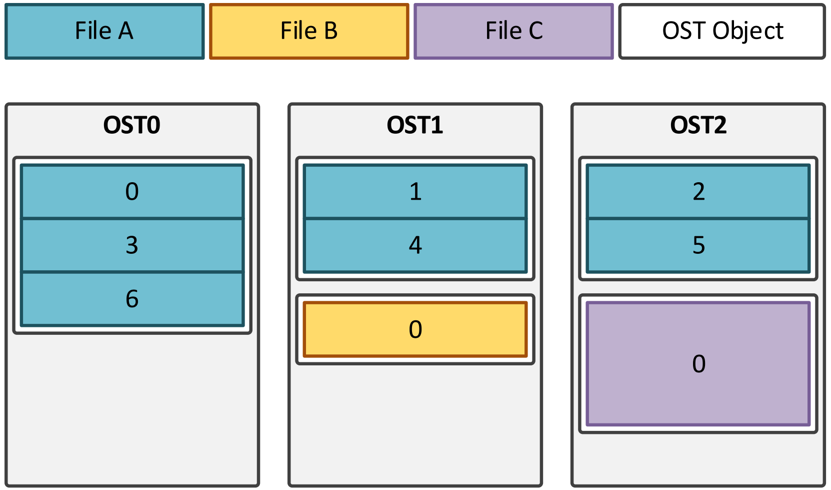
- Each task writes its own data to the same file using MPI-IO mapping.
- Advantages
  - ✓ Single file makes data manageable
- Disadvantages
  - × Lower performance than file-per-process for some concurrencies

Collective Buffering I/O:

- Groups of tasks perform parallel I/O on the same file or different files.
- Advantages
  - ✓ Better performance than single shared file
  - ✓ Fewer files than file-per-process
- Disadvantages
  - × Algorithmically complex

Lustre File System Striping
Basics

- Ability to stripe data across multiple OSTs
- Striping offers two benefits:
  - ***Large File Sizes:***
    - The ability to store large files by placing chuncks of a file on multiple OSTs. A file's size is not limited to the space available on a single OST.
  - ***Bandwidth***:
    - An increase in bandwidth because multiple processes can simultaneously access the same file. A file's I/O bandwidth is not limited to a single OST.
- The file layout is selected by the client, either
  - By policy (default settings, inherit from parent)
  - By the administrator, user or application



File striping, RAID-0 pattern.

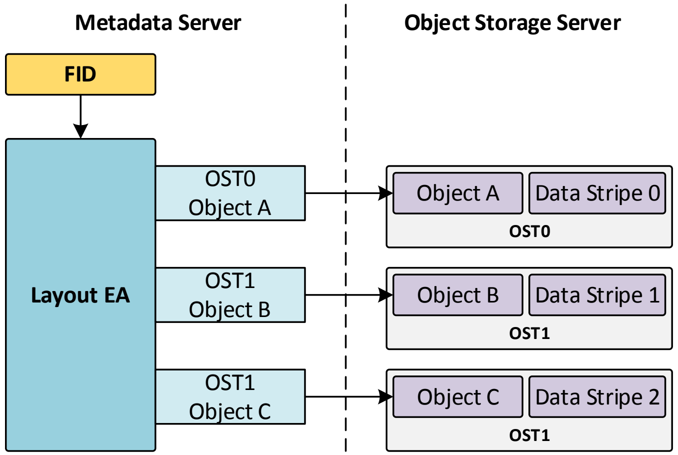
Source: *NICS, "Lustre Striping Guide", online: https://www.nics.tennessee.edu/computing-resources/file-systems/lustre-striping-guide*

Lustre File System Striping
File Layouts

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

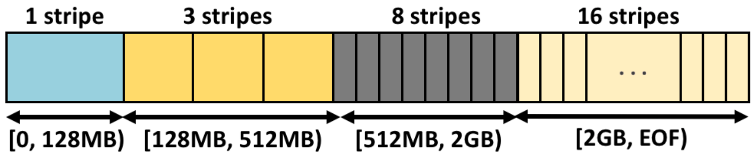
Plain File Layout:

- Information about where file data is located on the OST(s) is stored as an extended attribute called the ***Layout Extended Attribute (EA)***.



Progressive File Layout (PFL):

- Uses composite layouts to allow different RAID-0 layouts to describe different extents of a file.
- Basically an array of sub-layout components, with each sub-layout component being a plain layout covering different and non-overlapped extents.
- Reasonable performance for a variety of I/O patterns expected.
- Simplifies Lustre usage for novice users.
- Stripe layout changes as file grows.

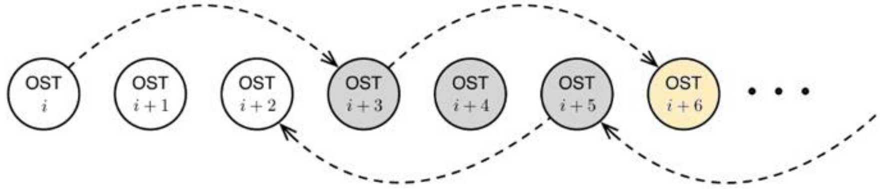


Lustre File System Striping
Standard Stripe Allocation Methods

- ***Round-robin allocator –*** When the OSTs have approximately the same amount of free space, the round-robin allocator alternates stripes between OSTs on different OSSs, so the OST used for stripe 0 of each file is evenly distributed among OSTs, regardless of the stripe count.

Simple example with eight OSTs and one OSS:

```
File 1: OST1, OST2, OST3, OST4
File 2: OST5, OST6, OST7
File 3: OST0, OST1, OST2, OST3, OST4, OST5
File 4: OST6, OST7, OST0
```



- ***Weighted allocator –*** When the free space difference between the OSTs becomes significant, the weighted algorithm is used to influence OST ordering based on size (amount of free space available on each OST) and location (stripes are evenly distributed across OSTs).

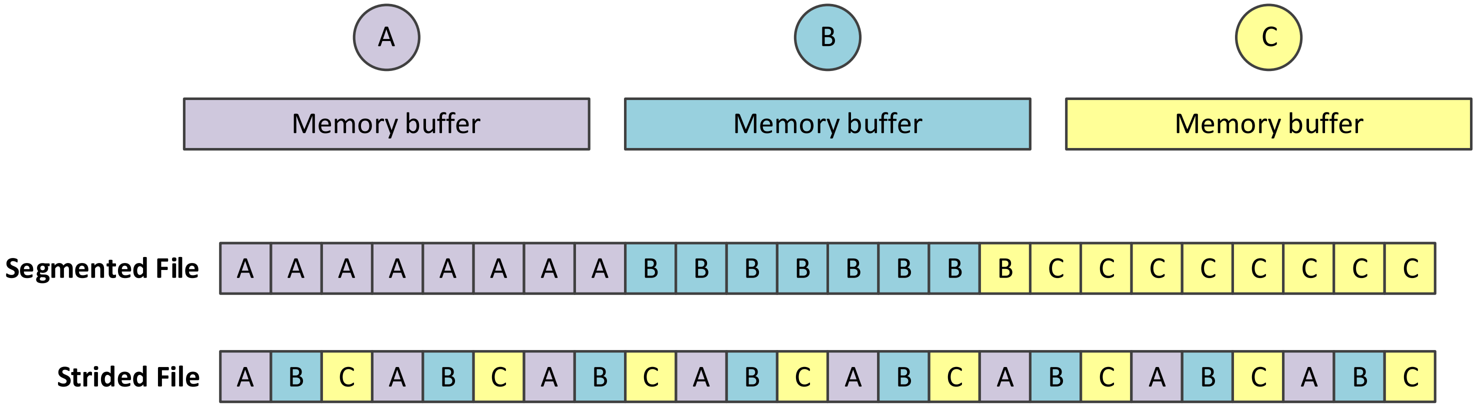
Source: *Lustre Operations Manual, Chapter 19.*

Lustre File System Striping
User Tools for Striping

- ***lfs setstripe*** – set striping pattern of a new file or for an existing directory over the command line
  - Create a new plain file or directory with one single layout:
    - `lfs setstripe [OPTIONS] {directory|filename}`
  - Create a new composite file, add one or more layout components to an existing composite file, or set or extend the default template on an existing directory:
    - `lfs setstripe {--component-end|-E end1} [component1_OPTIONS] [{--component-end|-E end2} [component2_OPTIONS] ...] {directory|filename}`
    - Example: `lfs setstripe -E 128M -c 1 -E 512M -c 3 -E 2G -c 8 -E -1 -c 16 <filename>`
- ***llapi*** – setting Lustre properties in a C program
  - `llapi_file_create()` – can be used to set the striping pattern for a new plain file
  - No API for creating or extending composite files!

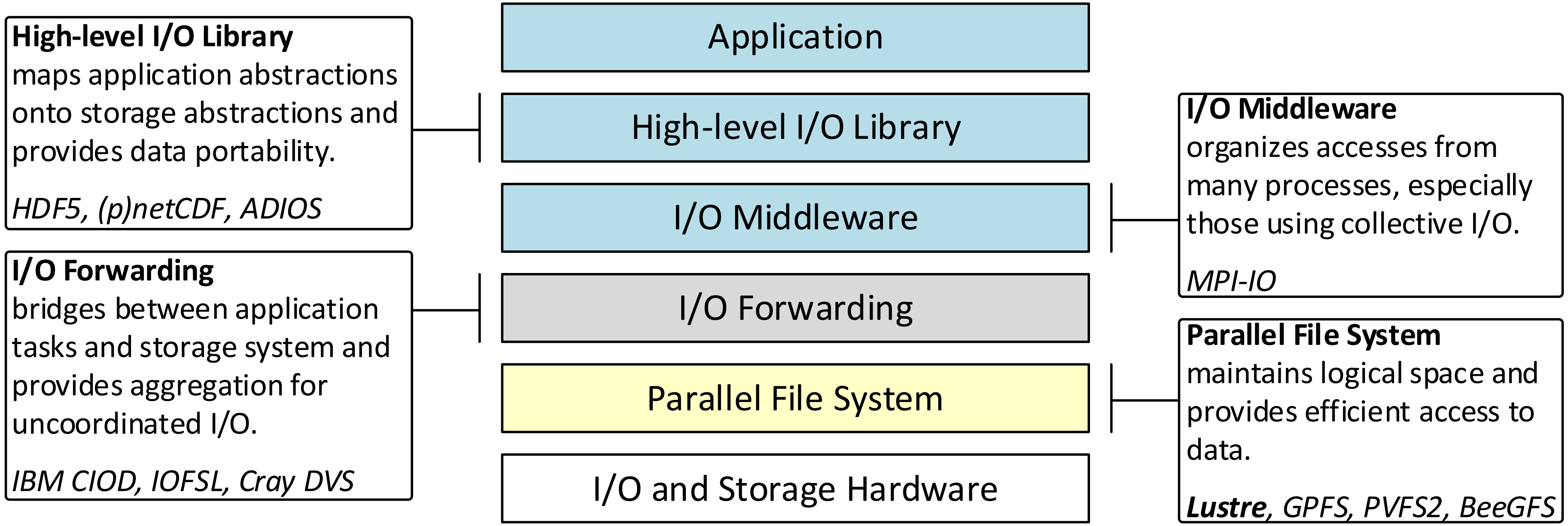
Source: *Lustre Operations Manual, Chapters 19 and 42.*

Best Practices for File I/O
Alignment of Data Access


A
B
C
Memory buffer
Memory buffer
Memory buffer
Segmented File
A A A A A A A A B B B B B B B B C C C C C C C C
Strided File
A B C A B C A B C A B C A B C A B C A B C A B C

- <u>Alignment of data access can be critical!</u>
  - Sub-block, non-aligned accesses require pre-fetching and buffering, at the minimum
  - Additional file locking overhead between threads or processes can add further overhead
- Try to minimize both the overhead associated with
  - Splitting an operation between storage targets <u>*and*</u>
  - Contention between writing processes over a single storage target

Best Practices for File I/O
High-level I/O Libraries, Middleware, and MPI



High-level I/O Library
maps application abstractions onto storage abstractions and provides data portability.

HDF5, (p)netCDF, ADIOS

I/O Forwarding
bridges between application tasks and storage system and provides aggregation for uncoordinated I/O.

IBM CIOD, IOFSL, Cray DVS

I/O Middleware
organizes accesses from many processes, especially those using collective I/O.

MPI-IO

Parallel File System
maintains logical space and provides efficient access to data.

***Lustre**, GPFS, PVFS2, BeeGFS*

Best Practices for File I/O
Recommendations at a Glance

1) The *stripe alignment* of data can be critical.
2) The *size and location of I/O operations* should be carefully managed to minimize the *file locking contention.*
3) The *I/O request size* (i.e., the transfer size) should be large: write fewer big chunks of data (i.e., a full stripe width or greater) rather than small bursty I/O.
4) It is recommended to *use high-level I/O libraries or middleware* to write flexible, portable programs. They map application abstractions onto storage abstractions and organize accesses from many processes.
5) Perform *parallel I/O*: single writer I/O can not take advantage of the system's parallel capabilities.
6) The user should have a *good understanding of how and how much the application outputs* before tuning the striping pattern!

Best Practices for File I/O
Striping Recommendations

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Plain File Layout

- $\#files \geq \#OSTs$: Reduce the Lustre contention and OST file locking

$$stripe\ count = 1$$

- $\#files < \#OSTs$: Utilize as many OSTs as possible:

$$stripe\ count = \left\lfloor \frac{\#OSTs}{\#files} \right\rfloor$$

- $\#files = 1$: Maximize the parallel access performance:

$$stripe\ count = \begin{cases} \#Aggregators, & \text{if } \#Aggr. \leq \#OSTs \\ \#OSTs, & \text{otherwise.} \end{cases}$$

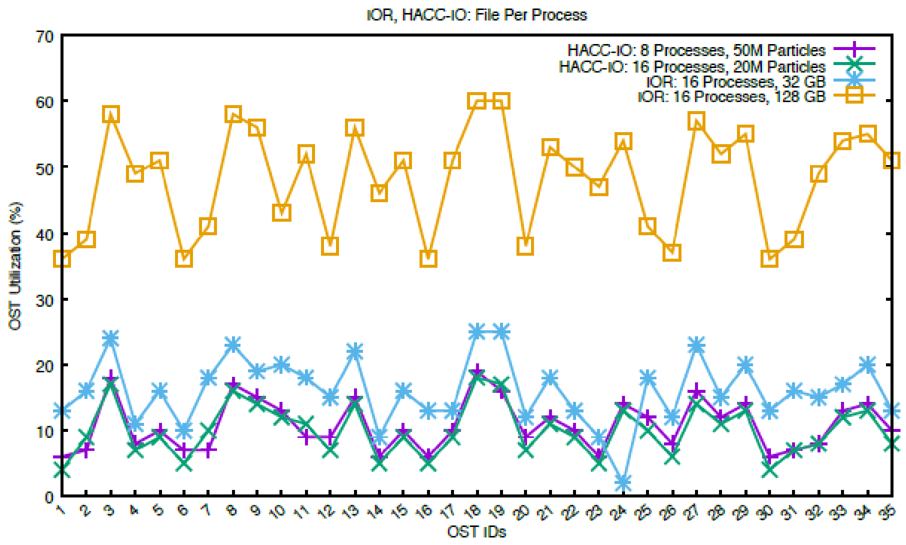
Progressive File Layout

- The progressive layout should stop growing at the point where the total number of stripes would equal or exceed the number of OSTs.
- To avoid oversubscribing OST bandwidth, OSTs used at the beginning of the file should not normally be re-used for objects allocated later in the file.

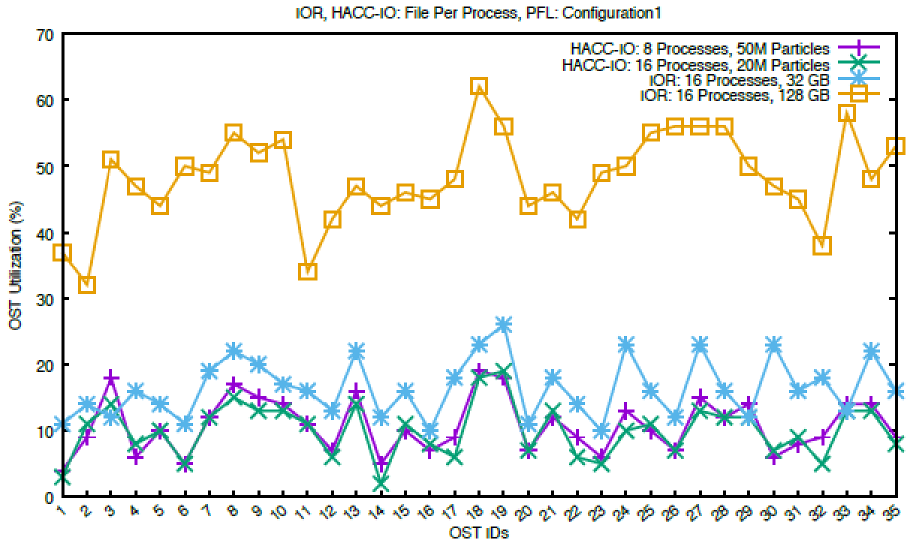
Sources:
(1) Sarah Neuwirth, "Accelerating Network Communication and I/O in Scientific High Performance Computing Environments", Ph.D. Thesis, 2018.
(2) Lustre Wiki, "Progressive File Layouts", available online: https://wiki.lustre.org/Progressive_File_Layouts

Best Practices for File I/O
OST Utilization with Standard Allocation Methods

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386



Non-PFL setup in FPP mode (stripe count = 8).



PFL setup in FPP mode with 4 extents.

Recent Research Developments
Client- and Server-side Approaches

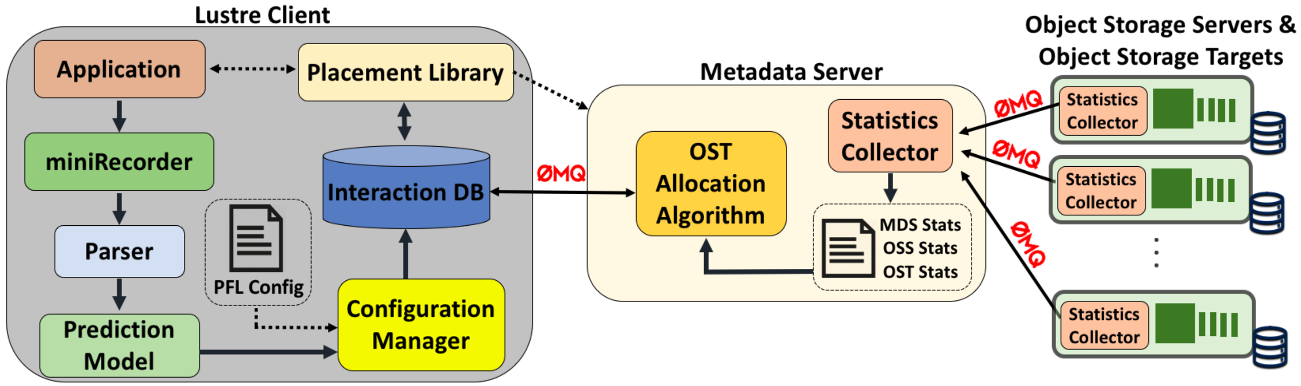
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Client-side approaches…
  - ✓ Address load imbalance on a per job basis
  - ✓ Can provide transparent implementations
  - × Do not consider requirements of other applications
  - × Lack a global view on storage stack components

So how about combining them?

- Server-side approaches…
  - ✓ Have a global view of storage servers
  - ✓ Simultaneously allocate resources to all concurrent applications
  - × Require modification of application source code
  - × Do not consider different file layouts

Recent Research Developments
iez – End-to-End Control Plane Architecture



***Source:** Arnab K. Paul, Sarah Neuwirth et al., "iez: Resource Contention Aware Load Balancing in Large-Scale Parallel File Systems", IPDPS 2019. Extended version (including PFL) submitted to IEEE TPDS.*

`iez` Client-side components:

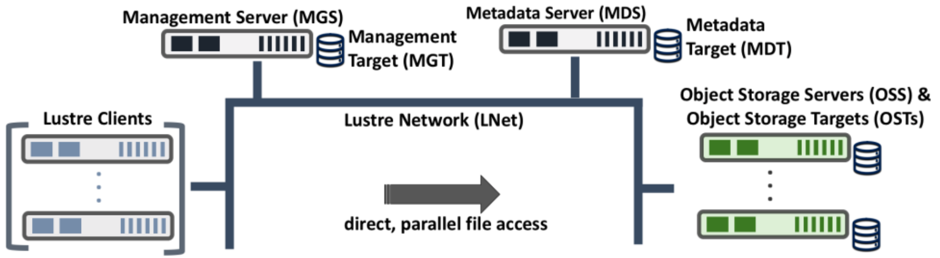
- **Capture** applications' I/O characteristics
- **Predict** future requests based on the traced I/O
- **Store** the predicted requests to be used by server-side
- **Place** the applications' I/O workload on the allocated set of OSTs

`iez` Server-side components:

- **Collect** real time statistics from OSSs, OSTs, and MDS
- **Allocate** a set of OSTs to application data, yielding a balanced load
- **Store** the load-balanced set of OSTs to be allocated for every request

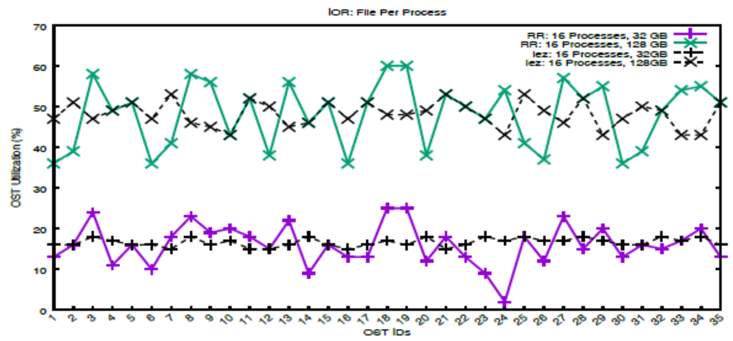
Performance Evaluation
Lustre Testbed Deployment

- **10 Node Cluster with:**
  - 1 MDS, 7 OSSs, 2 Clients
  - CentOS 7, Lustre 2.10
  - 8 cores / node, 3.2 GHz AMD Processor
  - 16 GB Memory
  - 5 OSTs per OSS (10 GB storage per OST)
- **File Layouts:**
  - Simple File Layout (Non-PFL)
  - Progressive File Layout (PFL)
- **Sharing Patterns:**
  - File Per Process (FPP)
  - Single Shared File (SSF)

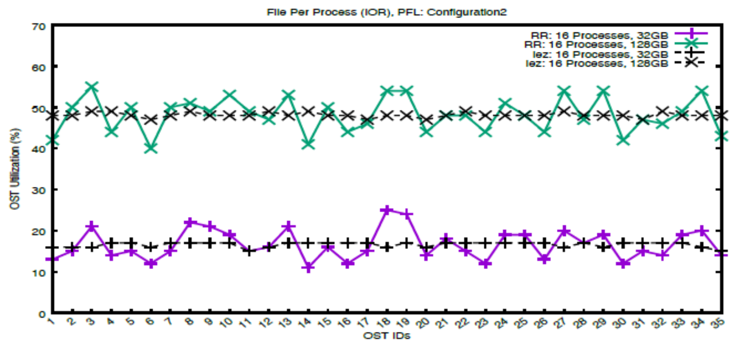


PFL Configuration 1		PFL Configuration 2	
Extent Range	***Stripe Count***	***Extent Range***	***Stripe Count***
[0, 128 MB)	1	[0, 128 MB)	1
[128 MB, 512 MB)	3	[128 MB, 2 GB)	12
[512 MB, 2 GB)	8	[2 GB, EOF)	32
[2 GB, EOF)	16		

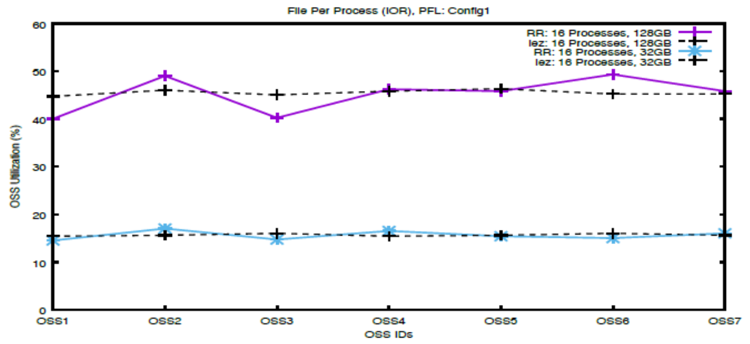
Performance Evaluation
Utilization Comparison – File-Per-Process



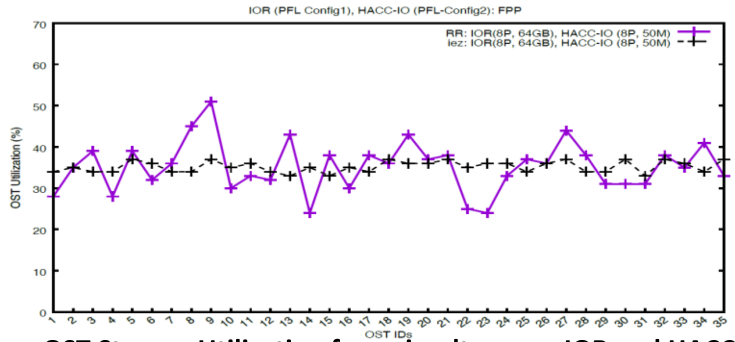
OST utilization: IOR in FPP mode and non-PFL layout (stripe count 8).



OST utilization: IOR in FPP mode and PFL Configuration 2.



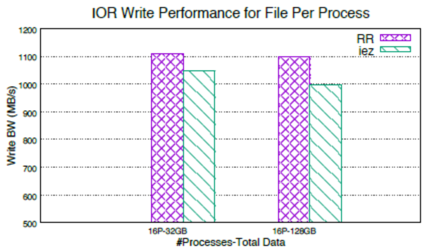
OSS utilization: IOR in FPP mode and PFL Configuration 1.

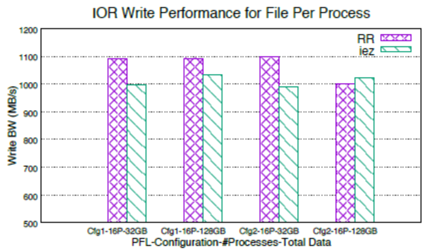


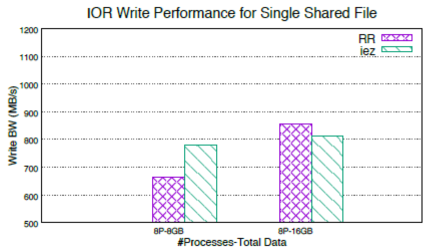
OST Storage Utilization for a simultaneous IOR and HACC-IO execution in FPP mode in PFL Configuration 1 and PFL Configuration 2, respectively.

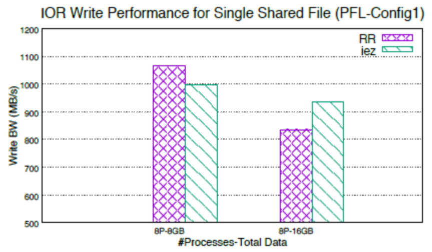
Performance Evaluation

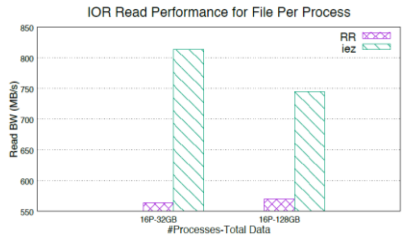
Read and Write Performance Comparison

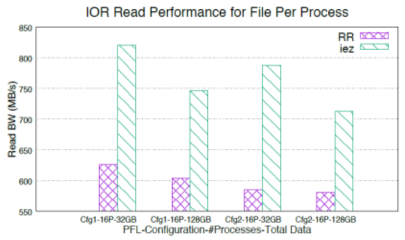


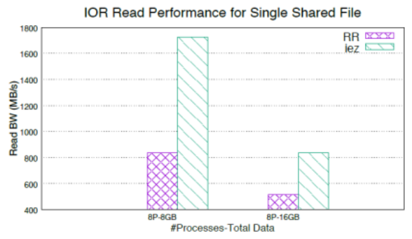


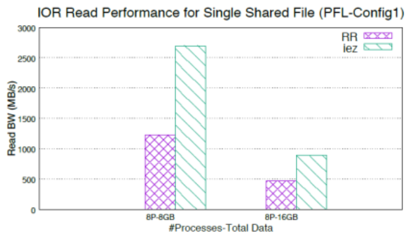


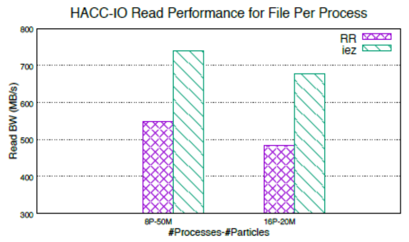


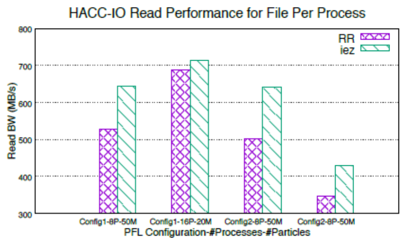


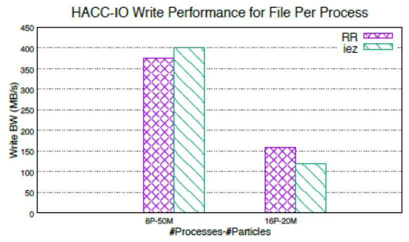


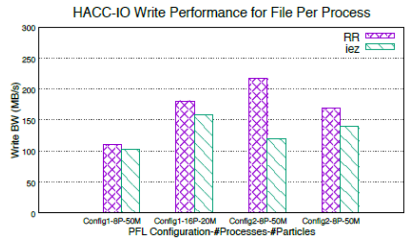












Conclusion

- ***Inconvenient truth:***
  - Every time code is ported to a new machine or underlying file system is changed or upgraded, users are required to make changes to maintain performance.
  - Users need a good understanding of how and how much their application outputs before tuning the striping pattern.
  - Users also need to be familiar with the recommended best practices for parallel I/O and need to be able to apply them efficiently.
- ***Current status of Lustre has potential for further improvement:***
  - Standard stripe allocators do not provide true load balancing yet.
  - Lustre could benefit from a global statistics collection and feedback system.
  - Write and read performance, but also the resource utilization could be improved transparently for user applications.

Thank you for your attention.

Questions?