

Efficient I/O and Data Intensive Computing

Ritu Arora (Galen Arnold , Jose Nandez)

(Along With Contributions from Dan Stanzione, Doug James, John Cazes, Robert McLay)

Email: rauta@tacc.utexas.edu

Overview

- Introduction
- **I/O During Pre-Processing Stage**
 - Choose the right data transfer protocol
 - Remove unnecessary data before processing
- **I/O During Processing Stage**
 - Understand your parallel file system
 - Use parallel I/O
- **I/O During Post-Processing Stage**
 - Move your data to secondary or tertiary storage media
- **An Example of a System for Data Intensive Computing**
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Before Running Applications Involving Large Datasets

- Prepare a data management plan
 - Determine the type of data, amount of data, and the rate at which the data will be produced or consumed
 - Determine the data retention value
 - Identify the required hardware and software for storing and accessing the data
 - Be aware of any compliance needs or policies for data usage
- Learn about the usage policies associated with the systems that you would like to use
 - Know your **filesystem**
 - Know about **do's and don'ts on resources of interest**
- A sample **checklist for data management plan**:
<http://www.dcc.ac.uk/resources/data-management-plans/checklist>

I/O in Compute- or Data-Intensive Applications

- Compute- or data-intensive applications often
 - Read initial conditions or datasets for processing
 - Write numerical data from simulations
 - Saving application-level checkpoints
- In case of large distributed HPC applications, the total execution time can be broken down into the computation time, communication time, and the I/O time
- Optimizing the time spent in computation, communication and I/O can lead to overall improvement in the application performance
- However, doing efficient I/O without stressing out the HPC system is challenging and often an afterthought

Overview

- Introduction
- **I/O During Pre-Processing Stage**
 - **Choose the right data transfer protocol**
 - Remove unnecessary data before processing
- I/O During Processing Stage
 - Understand your parallel file system
 - Use parallel I/O
- I/O During Post-Processing Stage
 - Move your data to secondary or tertiary storage media
- An Example of a System for Data Intensive Computing
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Protocols for Data Transfer

- Different protocols exist for data transfer to (and between) remote sites, *e.g.*,
 1. Linux command-line utilities **scp** & **rsync**
 2. Globus' **globus-url-copy** command-line utility
 3. Globus Connect
- Check the user-guide of the resource that you are wanting to use to see the list of supported protocols

Data Transfer Using **scp** or WinSCP

- If your local computer is a Mac or a Linux laptop, you can use the `scp` commands to transfer data to and from a remote resource like Stampede

scp filename username@hostname:/path/to/directory

- If you are using a Windows computer, you can download and use the WinSCP application (GUI-based), or download and use Cygwin (command-line based, can run the aforementioned commands)
 - For small amounts of data, you may also use the “File Transfer Window” available in the SSH client – drag and drop the files across the local laptop and a remote resource

More Information on Using WinSCP

- For learning the usage of WinSCP the following slides and video might be useful for the Windows users

- Slides:

<https://drive.google.com/open?id=0B8zOSeBE0p0rUDZvVVR4aHI5b0k>

- Video:

<https://www.youtube.com/watch?v=Nn7Ofb0IYwM>

Data Transfer Using `rsync`

- The `rsync` command is another way to transfer data and to keep the data at the source and destination in sync
`rsync <path-to-source-file> <path-to-destination-file>`
- If transferring the data for the first time to a remote resource, `rsync` and `scp` might show similar performance except when the connection drops
 - If a connection drops, upon restart of the data transfer, `rsync` will automatically transfer only the remaining files to the destination, it will skip the already transferred files
- `rsync` transfers only the actual changed parts of a file (instead of transferring an entire file)
 - this selective method of data transfer can be much more efficient than `scp` because it reduces the amount of data sent over the network

Using Globus Connect

- Globus Connect provides fast, secure transport via an easy-to-use web interface using pre-defined and user-created "endpoints"
- Globus Connect makes it possible to create a transfer endpoint on any machine (including campus servers and home laptops) with few clicks
- For more information on Globus Connect:
<https://www.globus.org/globus-connect>
<http://www.cac.cornell.edu/vw/DataTransfer/globus.aspx>

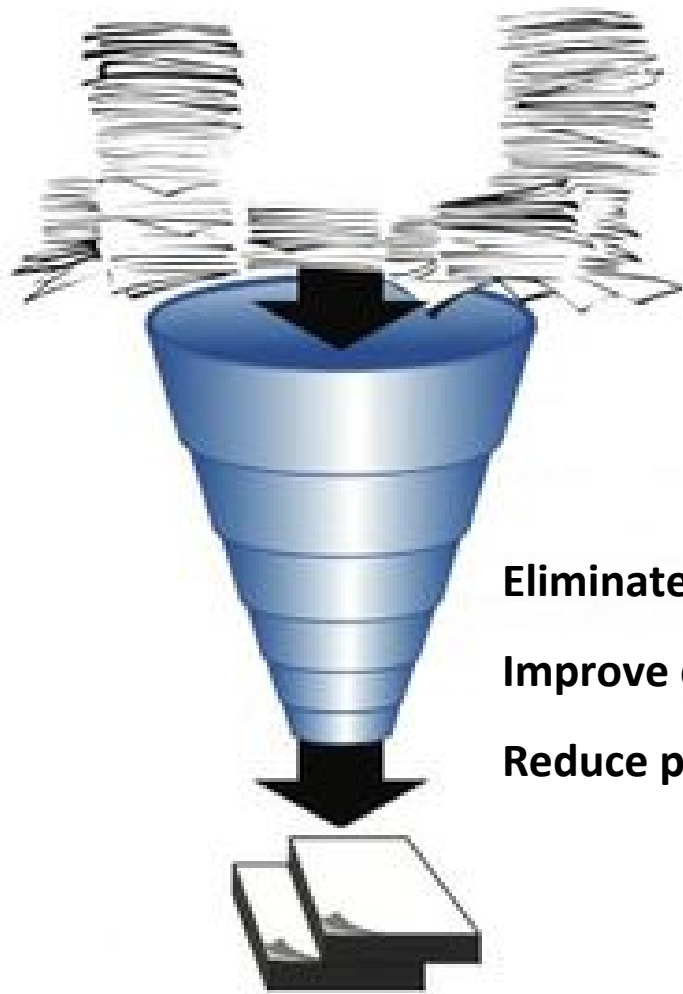
Data Transfer Issues – Real World Scenario

- During one project, transferring **4.3 TB** of data from the Stampede Supercomputer in Austin to the Gordon Supercomputer in San Diego, took **approx. 210 hours**
- The transfer was **restarted about 14 times** during June 3 to June 18, 2014 - about 15 days
- If the data transfer would have completed without any interruptions, it would have completed in about 9 days at the given speed
- Multiple reasons for interruption - sometimes maintenance on Stampede or Gordon, some other file-system issue, network traffic/available bandwidth - all are factors affecting the data transfer rate

Overview

- Introduction
- **I/O During Pre-Processing Stage**
 - Choose the right data transfer protocol
 - **Remove unnecessary data before processing**
- I/O During Processing Stage
 - Understand your parallel file system
 - Use parallel I/O
- I/O During Post-Processing Stage
 - Move your data to secondary or tertiary storage media
- An Example of a System for Data Intensive Computing
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Culling the Data Collection



Eliminate undesirable data

Improve quality

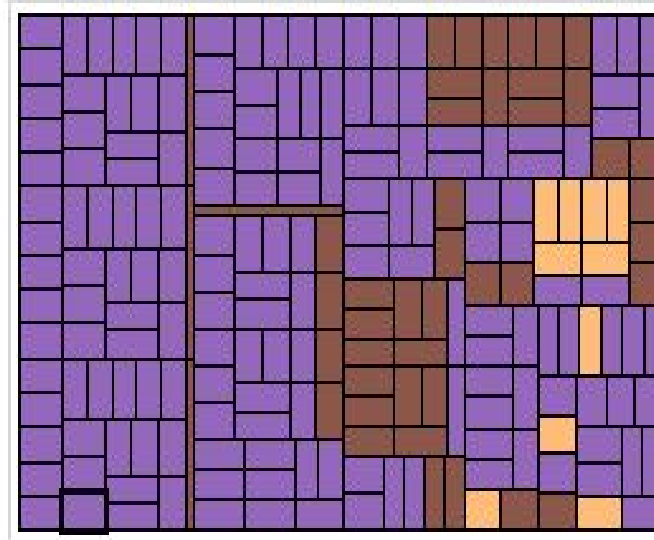
Reduce processing needs

- **Filtering**
 - By name, size, date
 - By type
 - By author, tags, metadata
- **De-duplication**
 - Exact binary match by checksum
- **Required**
 - Culling
 - Scripts for automation
 - Human intelligence

Interactive Visualization for Data Culling and Quality Control

FORMAT_NAME

- Graphics Interchange Fo...
- JPEG File Interchange Fo...
- Portable Network Graphics



FORMAT_NAME

- (All)
- Null
- Cascading Style Sheet
- Extensible Markup Langu...
- Graphics Interchange Fo...
- GZIP Format
- Hypertext Markup Langu...
- Java Archive Format
- Java Script File

✓ Keep Only

✗ Exclude

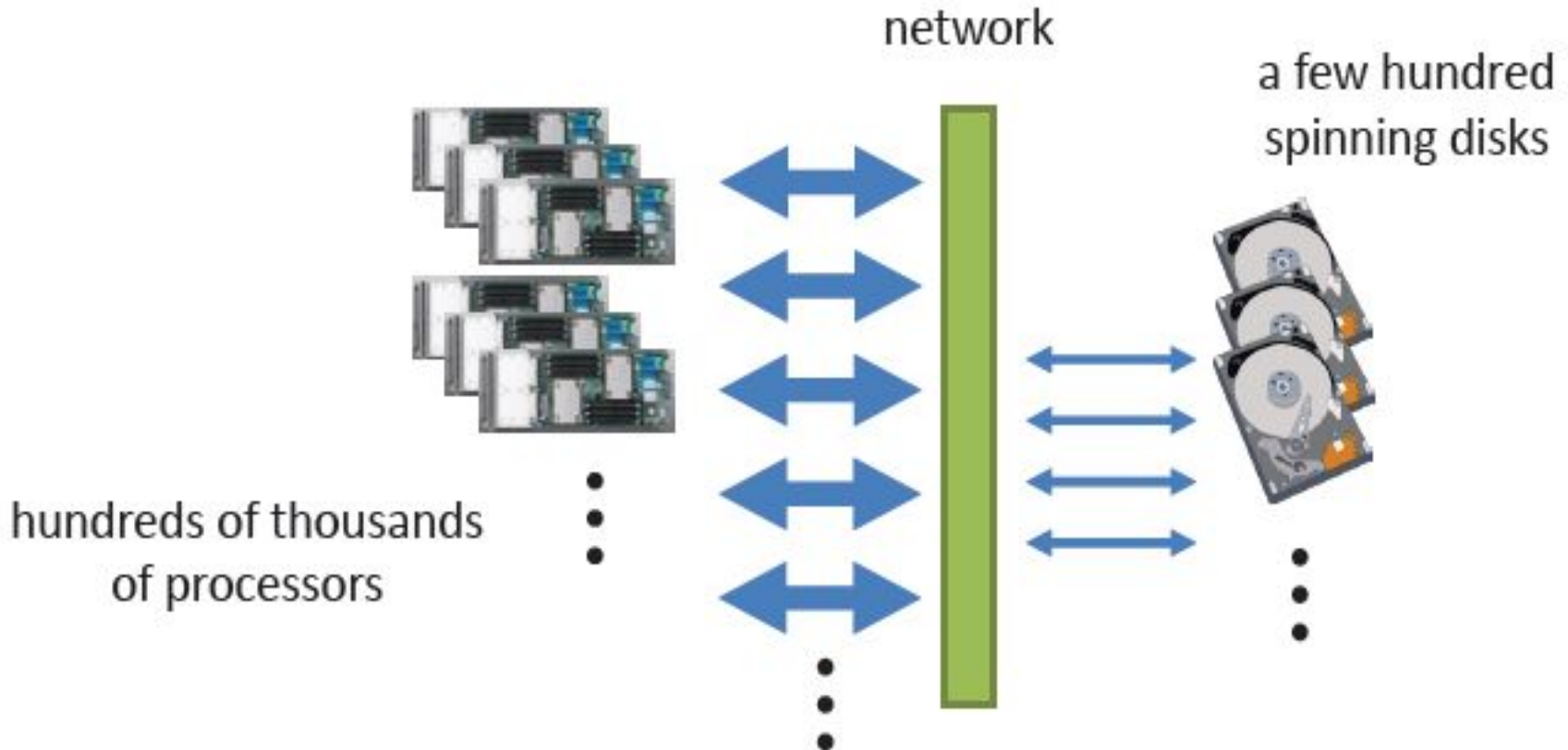


EXT: **jpg**
FILE_PATH: **/work/01698/rauta/tools/l_ics_2013.0.028/doc_icsxe/icsxe_gsg_files/Linux_itac_files/image014.jpg**
FORMAT_NAME: **JPEG File Interchange Format**
PARENT_ID: **167**
TYPE: **File**
Count of ID: **1**

Overview

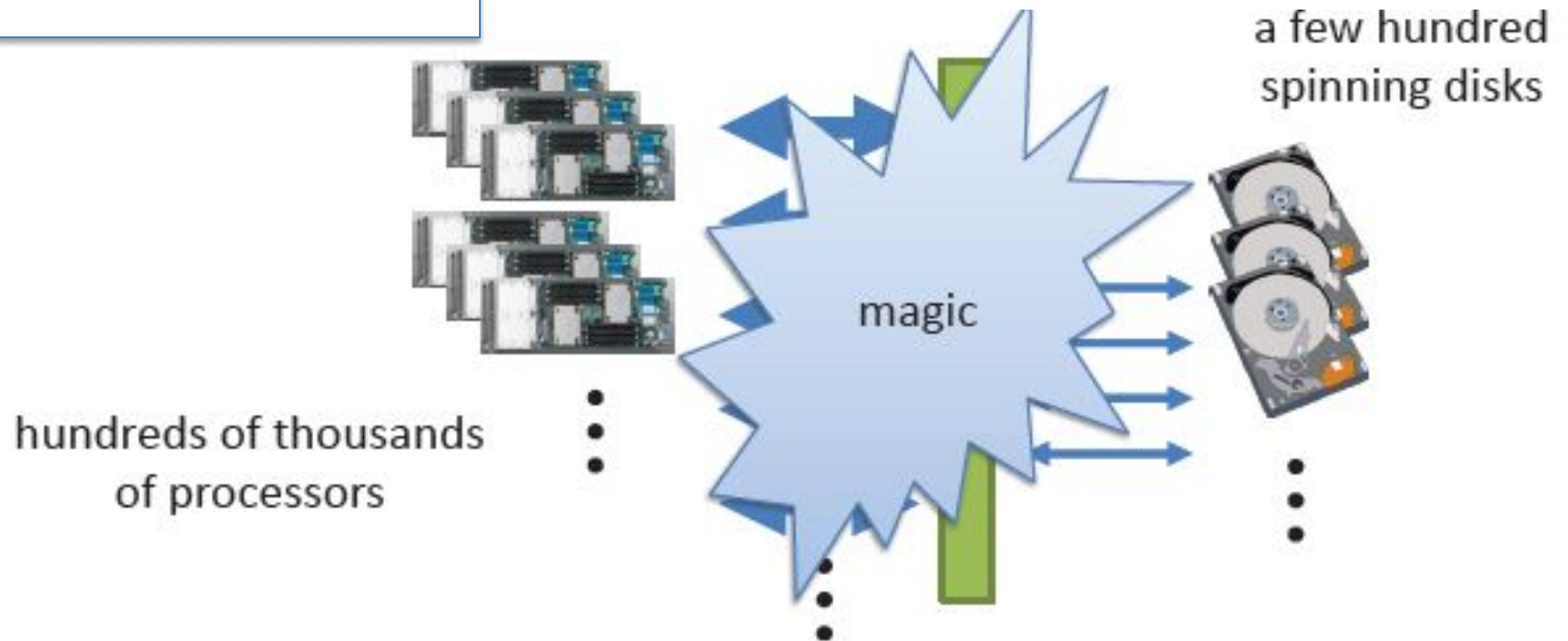
- Introduction
- I/O During Pre-Processing Stage
 - Choose the right data transfer protocol
 - Remove unnecessary data before processing
- **I/O During Processing Stage**
 - **Understand your parallel file system**
 - Use parallel I/O
- I/O During Post-Processing Stage
 - Move your data to secondary or tertiary storage media
- An Example of a System for Data Intensive Computing
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Modern HPC Cluster

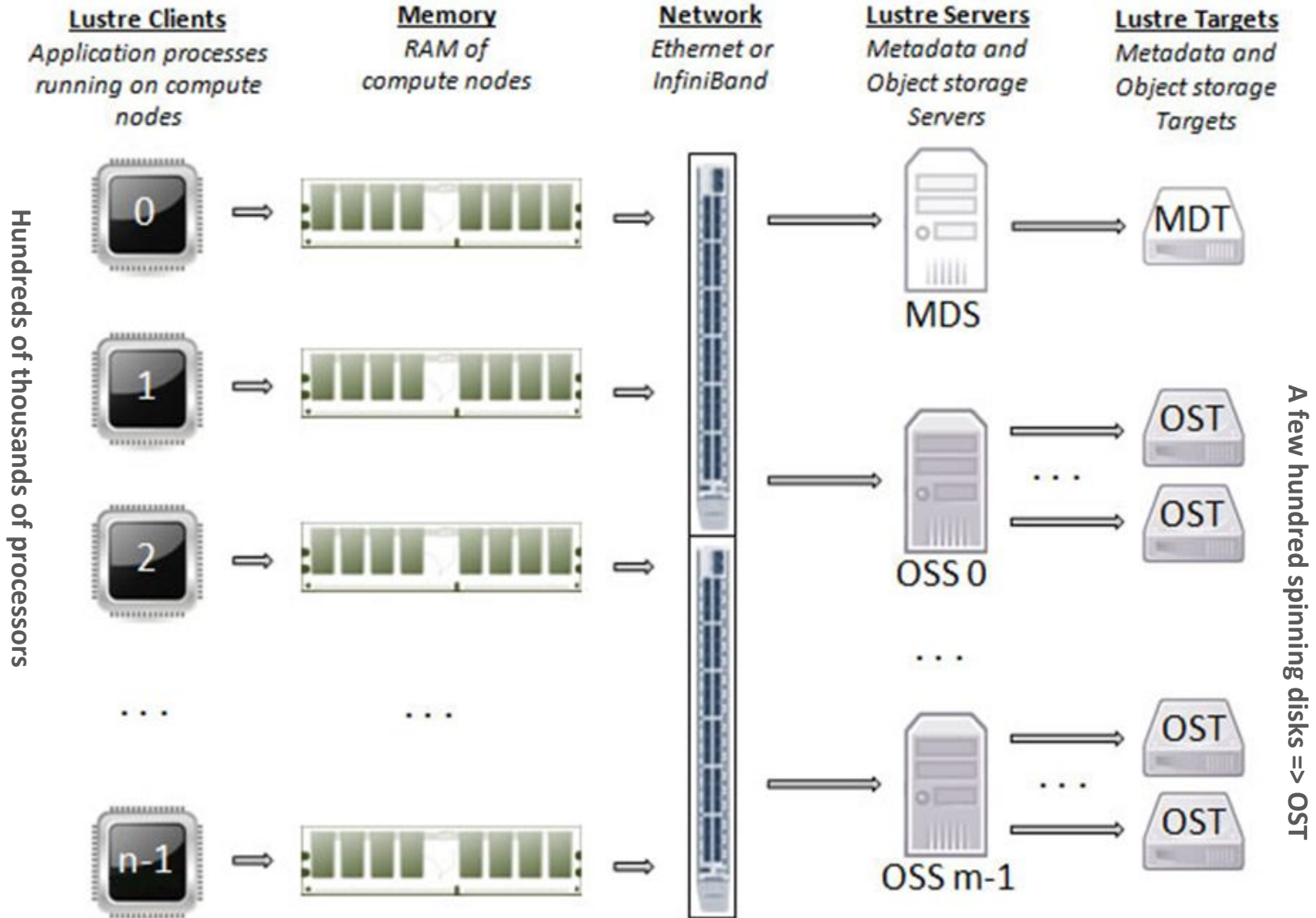


Modern HPC Cluster

...we need some magic to make the collection of spinning disks act like a single disk for the user...



...Parallel Filesystem (e.g., Lustre) Provides the Magic



Source: Reference 2, 4

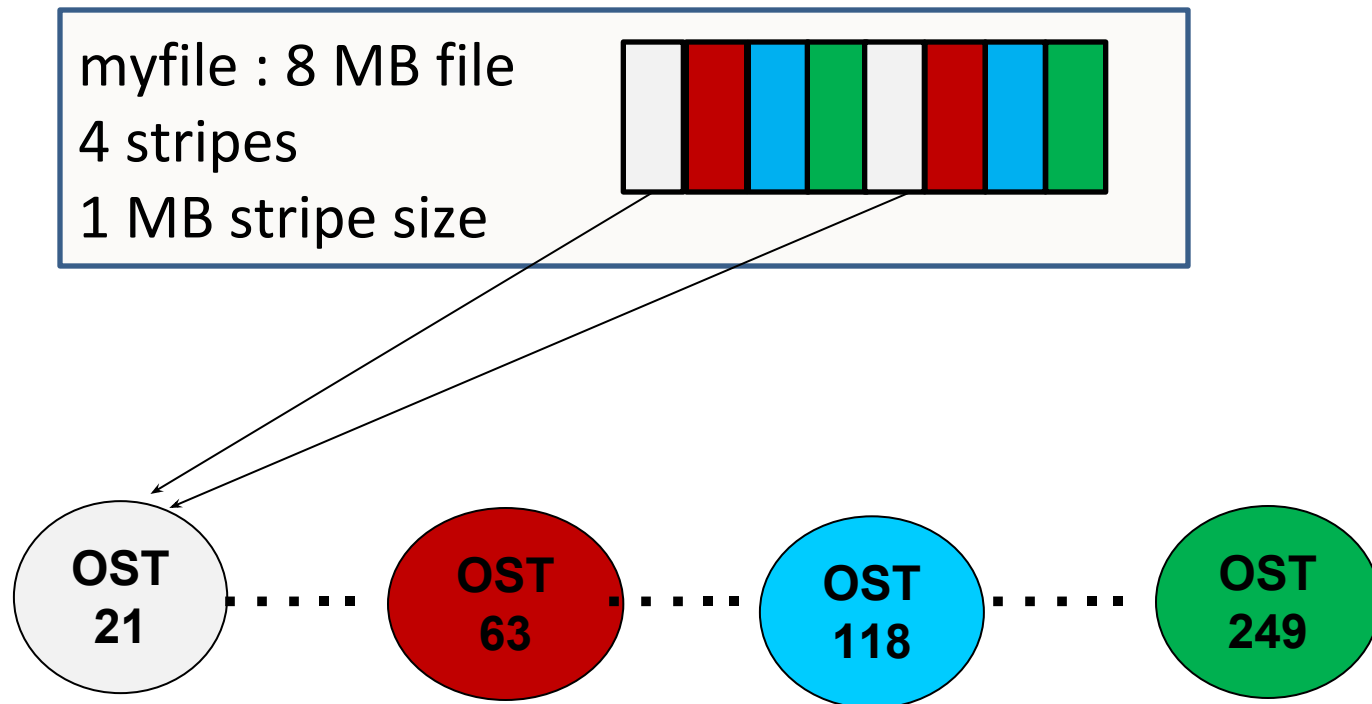
Lustre Filesystem at TACC

- Each Lustre filesystem has a different number of OSTs
- The greater the number of OSTs the better the I/O capability

	\$HOME	\$WORK	\$SCRATCH
Stampede	24	672	348
Lonestar	N/A (NFS)	30	90

Lustre File System - Striping

- Lustre supports the striping of files across several I/O servers (similar to RAID 0)
- Each stripe is a fixed size block



Lustre File System – Striping on TACC Resources

- Administrators set a default stripe count and stripe size that applies to all newly created files
 - Stampede: `$SCRATCH: 2 stripes/1MB`
`$WORK: 1 stripe /1MB`
 - Lonestar: `$SCRATCH: 2 stripes/1MB`
`$WORK: 1 stripe /1MB`
- However, users can reset the default stripe count or stripe size using the Lustre commands

Lustre Commands

- Get stripe count

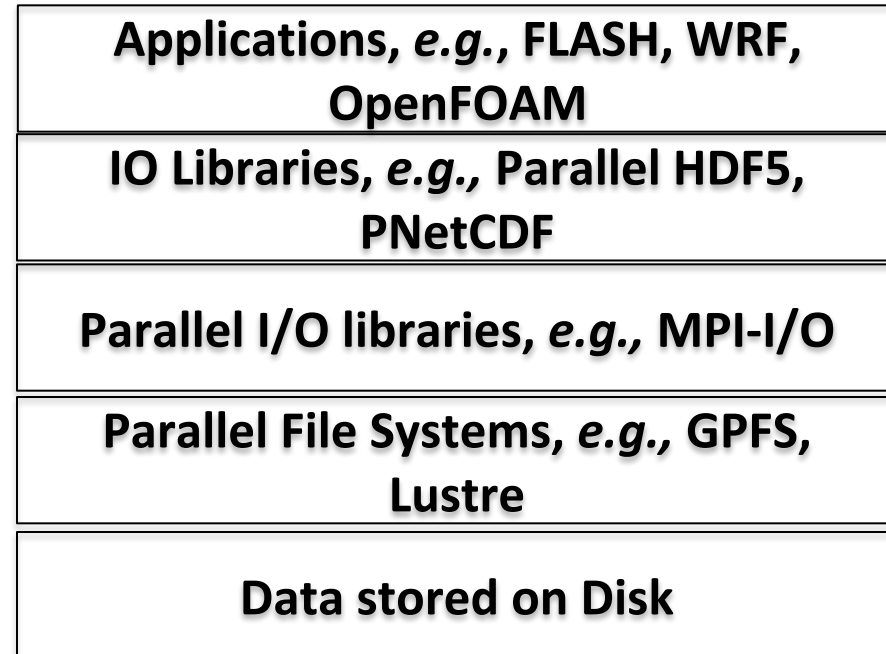
```
% lfs getstripe ./testfile
./testfile
lmm_stripe_count:    2
lmm_stripe_size:    1048576
lmm_stripe_offset:  50
      obdidx          objid          objid          group
      50             8916056        0x880c58        0
      38             8952827        0x889bfb        0
```

- Set stripe count

```
% lfs setstripe -c 4 -s 4M testfile2
% lfs getstripe ./testfile2
./testfile2
lmm_stripe_count:    4
lmm_stripe_size:    4194304
lmm_stripe_offset:  21
      obdidx          objid          objid          group
      21             8891547        0x87ac9b        0
      13             8946053        0x888185        0
      57             8906813        0x87e83d        0
      44             8945736        0x888048        0
```

Need for High-Level Support for Parallel I/O

- Parallel I/O can be hard to coordinate and optimize if working directly at the level of Lustre API
- Therefore, specialists implement a number of intermediate layers for coordination of data access and mapping from application layer to I/O layer
- Hence, application developers only have to deal with a high-level interface built on top of a software stack, that in turn sits on top of the underlying hardware
 - *e.g.*, MPI-I/O, parallel HDF5, T3PIO



Implementation Layers

See Reference # 4

You Can Stress Out Lustre Easily if You...

- Open and close the same file every few milliseconds
 - Stresses the MDS
- Too often, too many
 - Stresses the MDS and OSTs
- Write large files to `$HOME` or `$WORK`
 - `$SCRATCH` should be used
- `ls` in a crowded directory
 - `ls` is aliased to `"ls --color=tty"`
 - Every directory item incurs the overhead of an extra "stat" call to the MDS
 - Use `/bin/ls` in a crowded directory
- Create thousands of files in the same directory
 - A directory too is a file managed by the MDS

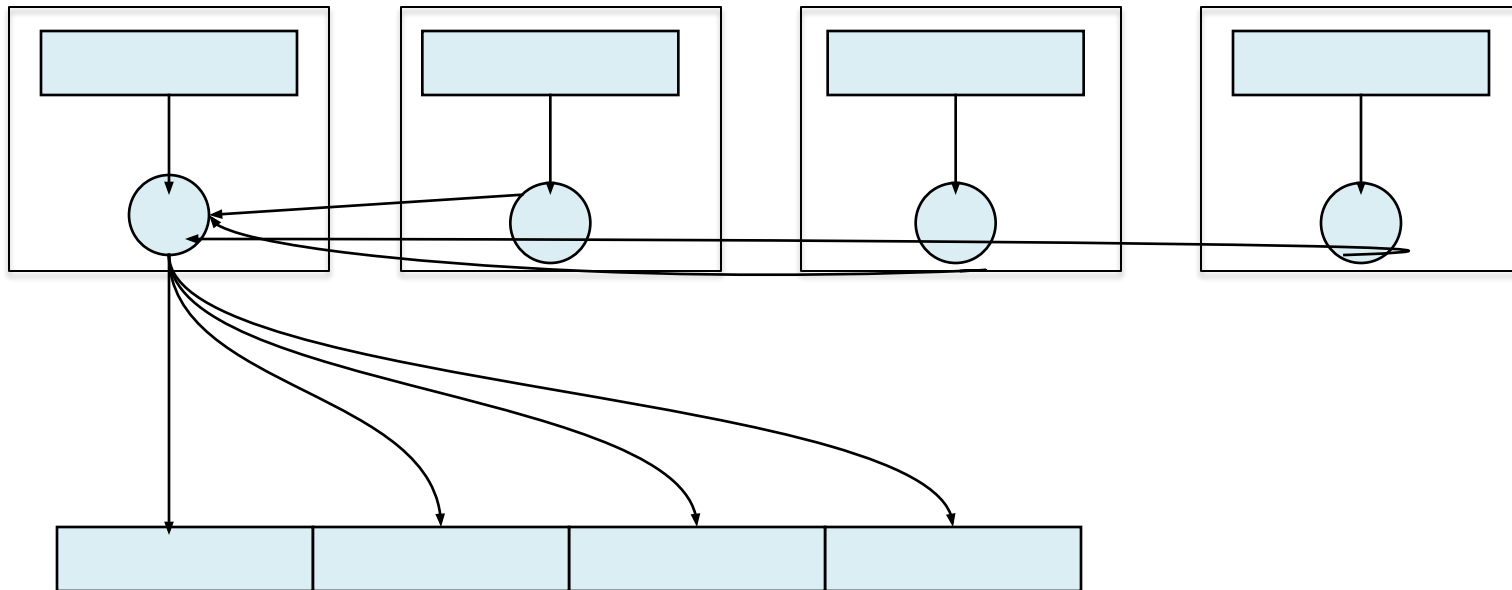
What happens when Lustre gets stressed out?

Overview

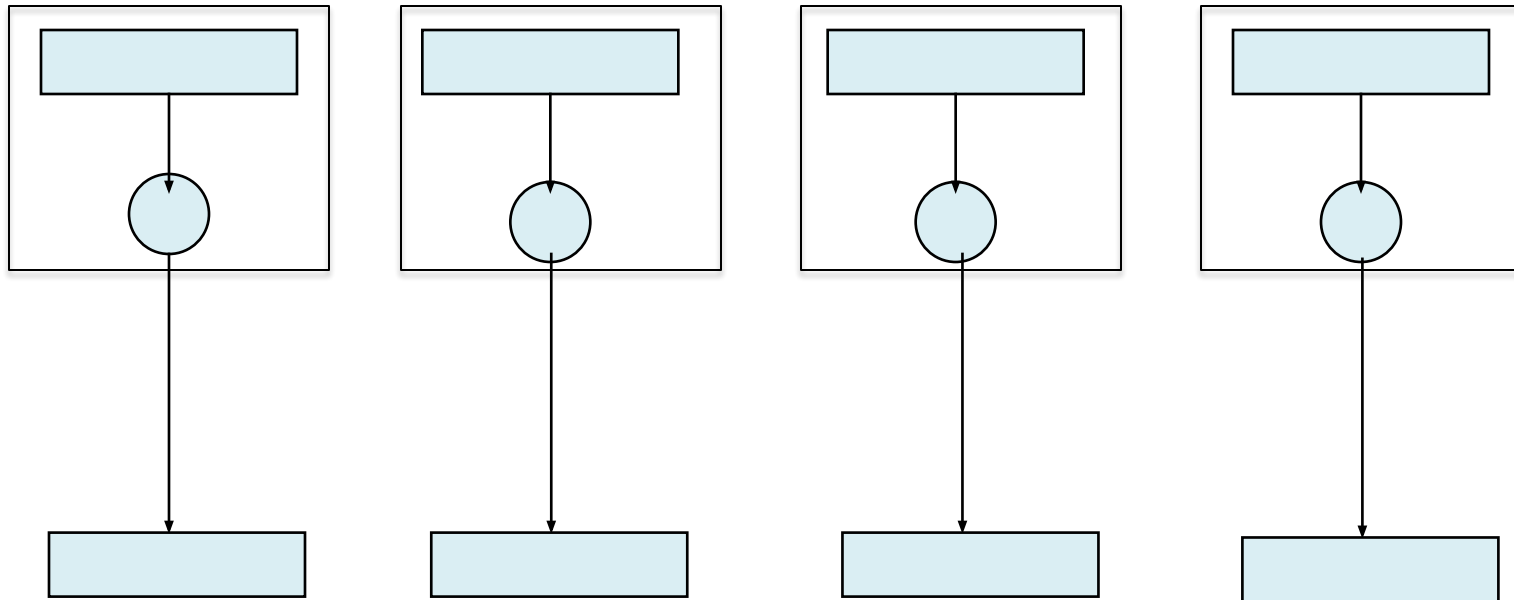
- Introduction
- I/O During Pre-Processing Stage
 - Choose the right data transfer protocol
 - Remove unnecessary data before processing
- **I/O During Processing Stage**
 - Understand your parallel file system
 - **Use parallel I/O**
- I/O During Post-Processing Stage
 - Move your data to secondary or tertiary storage media
- An Example of a System for Data Intensive Computing
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Typical Pattern: Parallel Programs Doing Sequential I/O

- All processes send data to master process, and then the process designated as master writes the collected data to the file
- This sequential nature of I/O can limit performance and scalability of many applications

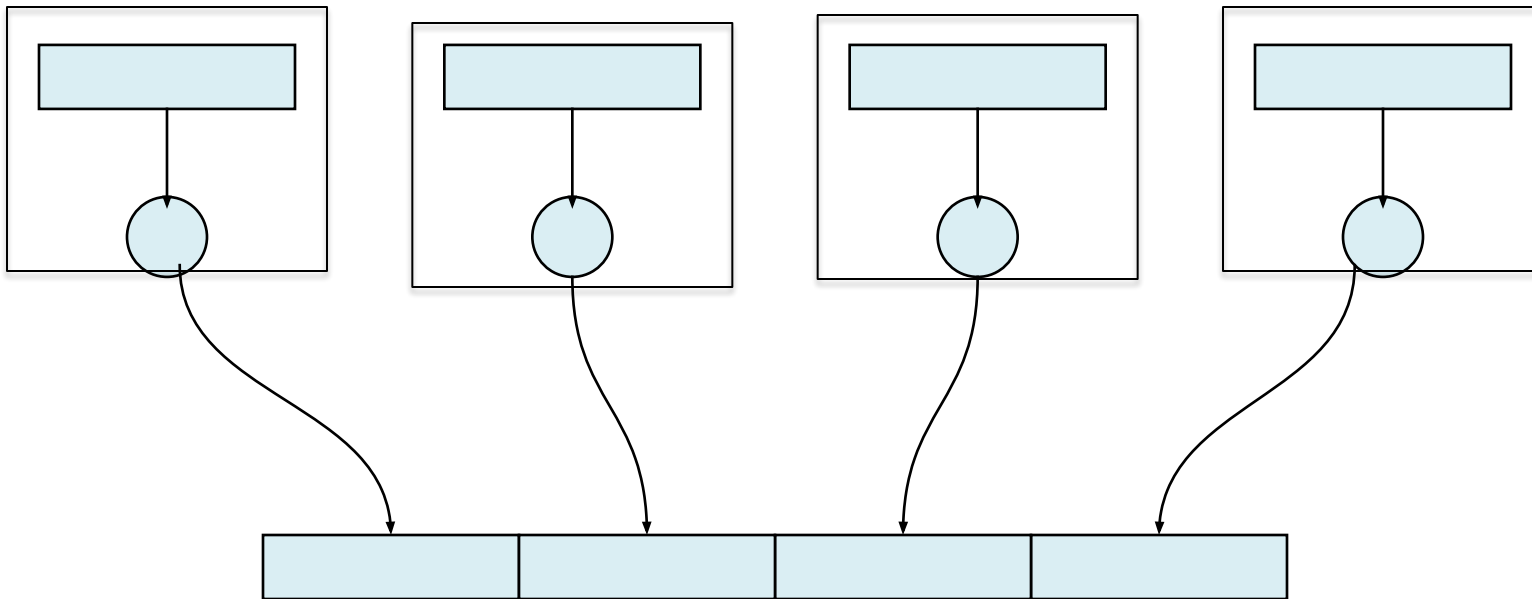


Another Pattern: Each Process Writing to a Separate File



Desired Pattern: Parallel Programs Doing Parallel I/O

- Multiple processes participating in reading data from or writing data to a common file in parallel
- This strategy improves performance and provides a single file for storage and transfer purposes

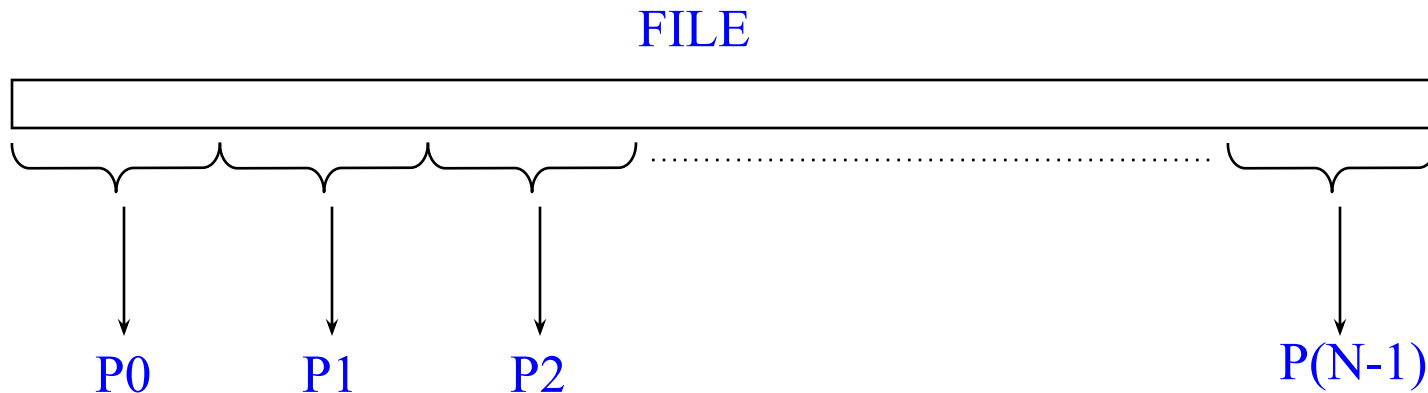


MPI for Parallel I/O

- A parallel I/O system for distributed memory architectures will need a mechanism to specify collective operations and specify noncontiguous data layout in memory and file
- Reading and writing in parallel is like receiving and sending messages
- Hence, an MPI-like machinery is a good setting for Parallel I/O (think MPI communicators and MPI datatypes)
- MPI-I/O featured in MPI-2 which was released in 1997, and it interoperates with the filesystem to enhance I/O performance for distributed-memory applications

Using MPI-I/O

- Given N number of processes, each process participates in reading or writing a portion of a common file
- There are three ways of positioning where the read or write takes place for each process:
 - Use individual file pointers (*e.g.*, `MPI_File_seek/MPI_File_read`)
 - Calculate byte offsets (*e.g.*, `MPI_File_read_at`)
 - Explicit offset operations perform data access at the file position given directly as an argument — no file pointer is used nor updated
 - Access a shared file pointer (*e.g.*, `MPI_File_seek_shared`, `MPI_File_read_shared`)



MPI-I/O API Opening and Closing a File

- Calls to the MPI functions for reading or writing must be preceded by a call to `MPI_File_open`
 - `int MPI_File_open(MPI_Comm comm, char *filename, int mode, MPI_Info info, MPI_File *fh)`
- The parameters below are used to indicate how the file is to be opened

<code>MPI_File_open mode</code>	Description
<code>MPI_MODE_RDONLY</code>	read only
<code>MPI_MODE_WRONLY</code>	write only
<code>MPI_MODE_RDWR</code>	read and write
<code>MPI_MODE_CREATE</code>	create file if it doesn't exist

- To combine multiple flags, use bitwise-or “|” in C, or addition “+” in Fortran
- Close the file using: `MPI_File_close (MPI_File fh)`

MPI-I/O API for Reading Files

After opening the file, read data from files by either using `MPI_File_seek` & `MPI_File_read` Or `MPI_File_read_at`

```
int MPI_File_seek( MPI_File fh, MPI_Offset offset,  
int whence )
```

```
int MPI_File_read(MPI_File fh, void *buf, int count,  
MPI_Datatype datatype, MPI_Status *status)
```

whence in `MPI_File_seek` updates the individual file pointer according to

`MPI_SEEK_SET`: the pointer is set to offset

`MPI_SEEK_CUR`: the pointer is set to the current pointer position plus offset

`MPI_SEEK_END`: the pointer is set to the end of file plus offset

```
int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void  
*buf, int count, MPI_Datatype datatype, MPI_Status *status)
```


Reading a File: readFile2.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
    int rank, size, bufsize, nints;
    MPI_File fh;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    bufsize = FILESIZE/size;
    nints = bufsize/sizeof(int);
    int buf[nints];
    MPI_File_open(MPI_COMM_WORLD,"dfile",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
    MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
    MPI_File_read(fh, buf, nints, MPI_INT, &status);
    printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
    MPI_File_close(&fh);
    MPI_Finalize();
    return 0;
}
```

Reading a File: readFile2.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
    int rank, size, bufsize, nints;
    MPI_File fh; ←----- Declaring a File Pointer
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    bufsize = FILESIZE/size; ←----- Calculating Buffer Size
    nints = bufsize/sizeof(int);
    int buf[nints]; ←----- Opening a File
    MPI_File_open(MPI_COMM_WORLD, "dfile", MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
    MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET); ←----- File seek &
    MPI_File_read(fh, buf, nints, MPI_INT, &status); ←----- Read
    printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
    MPI_File_close(&fh); ←----- Closing a File
    MPI_Finalize();
    return 0;
}
```

Reading a File: readFile1.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
    int rank, size, bufsize, nints;
    MPI_File fh;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    bufsize = FILESIZE/size;
    nints = bufsize/sizeof(int);
    int buf[nints];
    MPI_File_open(MPI_COMM_WORLD,"dfile",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
    MPI_File_read_at(fh, rank*bufsize, buf, nints, MPI_INT, &status);
    printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
    MPI_File_close(&fh);
    MPI_Finalize();
    return 0;
}
```

Combining file seek & read in
one step for thread safety in
MPI_File_read_at

MPI-I/O API for Writing Files

- While opening the file in the write mode, use the appropriate flag/s in `MPI_File_open`: `MPI_MODE_WRONLY` Or `MPI_MODE_RDWR` and if needed, `MPI_MODE_CREATE`
- For writing, use `MPI_File_set_view` and `MPI_File_write` or `MPI_File_write_at`

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,  
MPI_Datatype etype, MPI_Datatype filetype, char  
*datarep, MPI_Info info)
```

```
int MPI_File_write(MPI_File fh, void *buf, int count,  
MPI_Datatype datatype, MPI_Status *status)
```

```
int MPI_File_write_at(MPI_File fh, MPI_Offset offset,  
void *buf, int count, MPI_Datatype datatype,  
MPI_Status *status)
```

Writing a File: writeFile1.c (1)

```
1.  #include<stdio.h>
2.  #include "mpi.h"
3.  int main(int argc, char **argv){
4.      int i, rank, size, offset, nints, N=16 ;
5.      MPI_File fh;
6.      MPI_Status status;
7.      MPI_Init(&argc, &argv);
8.      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9.      MPI_Comm_size(MPI_COMM_WORLD, &size);
10.     int buf[N];
11.     for ( i=0;i<N;i++){
12.         buf[i] = i ;
13.     }
14.     ...
```

Writing a File: writeFile1.c (2)

```
15.  offset = rank * (N/size) * sizeof(int);

16.  MPI_File_open(MPI_COMM_WORLD, "datafile",
MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &fhw);

17.  printf("\nRank: %d, Offset: %d\n", rank, offset);

18.  MPI_File_write_at(fhw, offset, buf, (N/size),
MPI_INT, &status);

19.  MPI_File_close(&fhw);

20.  MPI_Finalize();
21.  return 0;
22. }
```

Compile & Run the Program on Compute Node

```
c401-204$ mpicc -o writeFile1 writeFile1.c
```

```
c401-204$ ibrun -np 4 ./writeFile1
```

TACC: Starting up job 1754636

TACC: Setting up parallel environment for MVAPICH2+mpispawn.

Rank: 0, Offset: 0

Rank: 1, Offset: 16

Rank: 3, Offset: 48

Rank: 2, Offset: 32

TACC: Shutdown complete. Exiting.

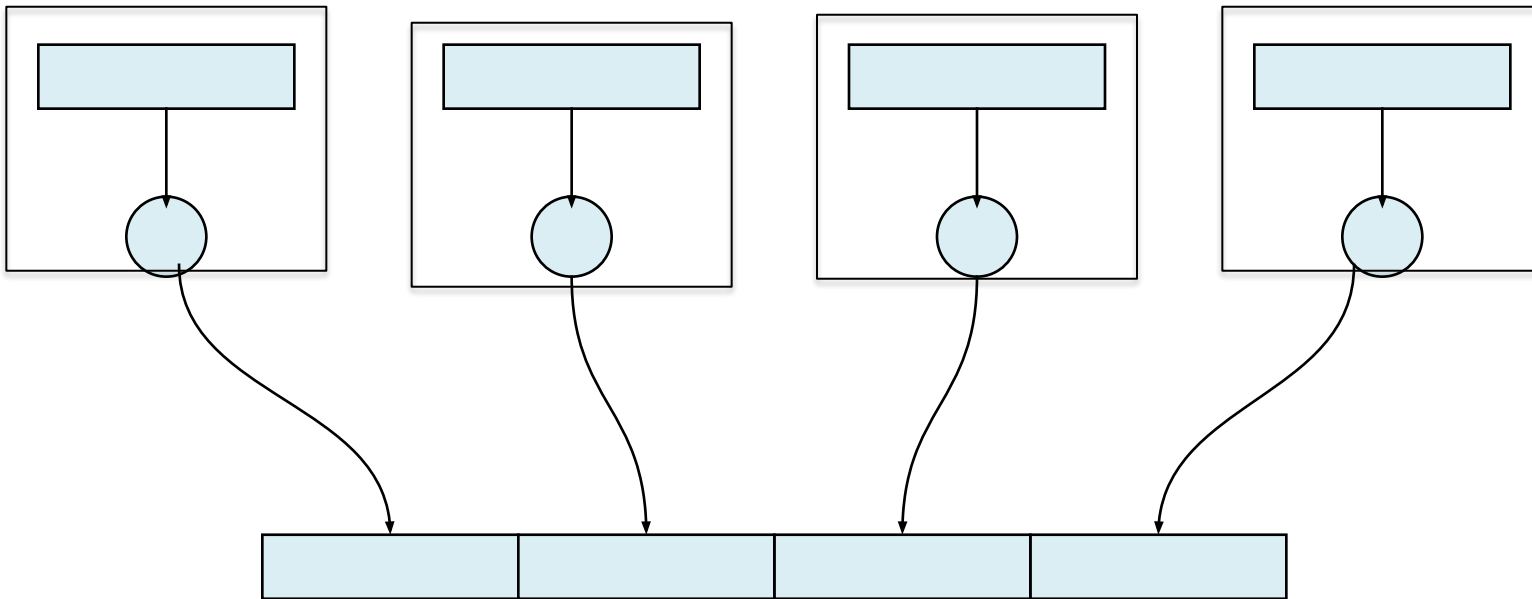
```
c401-204$ hexdump -v -e '7/4 "%10d "' -e '"\n"' datafile
```

```
 0    1    2    3    0    1    2
 3    0    1    2    3    0    1
 2    3
```

File Views for Writing to a Shared File (1)


When processes need to write to a shared file, assign regions of the file to separate processes using **MPI_File_set_view**


```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,  
MPI_Datatype etype, MPI_Datatype filetype, char  
*datarep, MPI_Info info)
```



File Views for Writing to a Shared File (2)

- File views are specified using a triplet - (*displacement*, *etype*, and *filetype*) – that is passed to `MPI_File_set_view`
displacement = number of bytes to skip from the start of the file
etype = unit of data access (can be any basic or derived datatype)
filetype = specifies which portion of the file is visible to the process

 `etype = MPI_INT`

 `filetype = 2xMPI_INT + gap of 4xMPI_INT`

 FILE

Source: https://www.chpc.utah.edu/images/news/sp2002_Martin07.jpg

- Data representation (datarep above) can be `native`, `internal`, or `external32`

Writing a File: writeFile2.c (1)

```
1. #include<stdio.h>
2. #include "mpi.h"
3. int main(int argc, char **argv){
4.     int i, rank, size, offset, nints, N=16;
5.     MPI_File fhw;
6.     MPI_Status status;
7.     MPI_Init(&argc, &argv);
8.     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9.     MPI_Comm_size(MPI_COMM_WORLD, &size);
10.    int buf[N];
11.    for ( i=0;i<N;i++){
12.        buf[i] = i ;
13.    }
14.    offset = rank*(N/size)*sizeof(int);
15.    ...
```

Writing a File: writeFile2.c (2)

```
16. MPI_File_open(MPI_COMM_WORLD, "datafile3",
MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL,
&fhw);
17.     printf("\nRank: %d, Offset: %d\n", rank,
offset);
18.     MPI_File_set_view(fhw, offset, MPI_INT,
MPI_INT, "native", MPI_INFO_NULL);
19.     MPI_File_write(fhw, buf, (N/size), MPI_INT,
&status);
20.     MPI_File_close(&fhw);
21.     MPI_Finalize();
22.     return 0;
23. }
```

Compile & Run the Program on Compute Node

```
c402-302$ mpicc -o writeFile2 writeFile2.c
```

```
c402-302$ ibrun -np 4 ./writeFile2
```

TACC: Starting up job 1755476

TACC: Setting up parallel environment for MVAPICH2+mpispawn.

Rank: 1, Offset: 16

Rank: 2, Offset: 32

Rank: 3, Offset: 48

Rank: 0, Offset: 0

TACC: Shutdown complete. Exiting.

```
c402-302$ hexdump -v -e '7/4 "%10d "' -e '"\n"' datafile3
```

```
0      1      2      3      0      1      2
```

```
3      0      1      2      3      0      1
```

```
2      3
```

Collective I/O (1)

- Collective I/O is a critical optimization strategy for reading from, and writing to, the parallel file system
- The collective read and write calls force all processes in the communicator to read/write data simultaneously and to wait for each other
- The MPI implementation optimizes the read/write request based on the combined requests of all processes and can merge the requests of different processes for efficiently servicing the requests
- This is particularly effective when the accesses of different processes are noncontiguous

Collective I/O (2)

- The collective functions for reading and writing are:
 - `MPI_File_read_all`
 - `MPI_File_write_all`
 - `MPI_File_read_at_all`
 - `MPI_File_write_at_all`
- Their signature is the same as for the non-collective versions

MPI-I/O Hints

- MPI-IO hints are extra information supplied to the MPI implementation through the following function calls for improving the I/O performance
 - `MPI_File_open`
 - `MPI_File_set_info`
 - `MPI_File_set_view`
- Hints are optional and implementation-dependent
 - you may specify hints but the implementation can ignore them
- `MPI_File_get_info` used to get list of hints, examples of Hints: **`striping_unit`**, **`striping_factor`**

Lustre – setting stripe count in MPI Code

- MPI may be built with Lustre support
 - MVAPICH2 & OpenMPI support Lustre
- Set stripe count in MPI code
 - Use MPI I/O hints to set Lustre stripe count, stripe size, and # of writers

Fortran:

```
call mpi_info_set(myinfo, "striping_factor", stripe_count, mpierr)
call mpi_info_set(myinfo, "striping_unit", stripe_size, mpierr)
call mpi_info_set(myinfo, "cb_nodes", num_writers, mpierr)
```

C:

```
mpi_info_set(myinfo, "striping_factor", stripe_count);
mpi_info_set(myinfo, "striping_unit", stripe_size);
mpi_info_set(myinfo, "cb_nodes", num_writers);
```

- Default:
 - # of writers = # Lustre stripes

HDF5 and Parallel HDF5

Hierarchical Data Format (HDF)

- HDF is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of numerical data
- It is portable across operating systems and architectures, and it supports flexible user-defined types
- HDF5 file structure includes two major types of objects:
 - **Datasets**, which are multidimensional arrays of a homogeneous type
 - **Groups**, which are container structures which can hold datasets and other groups
- Any HDF5 group or dataset may have an associated attribute list
 - An HDF5 **attribute** is a user-defined HDF5 structure that provides extra information about an HDF5 object

General Structure of HDF5 Code

Open HDF5

Open File

Open Group

Open Dataset

Write Dataset

Close Dataset

Close Group

Close File

Close HDF5

Code Samples: <https://www.hdfgroup.org/HDF5/examples>

HDF5 Code Example (1)

Source: https://www.hdfgroup.org/ftp/HDF5/current/src/unpacked/examples/h5_crtdat.c

```
1. //Example to create a dataset that is a 4 x 6 array
2. #include "hdf5.h"
3. #define FILE "dset.h5"
4. int main() {
5.     hid_t file_id, dataset_id, dataspace_id;
6.     //identifiers
7.     hsize_t dims[2];herr_t status;
8.     //Create a new file using default properties
9.     file_id = H5Fcreate(FILE, H5F_ACC_TRUNC,
10. H5P_DEFAULT, H5P_DEFAULT);
11.     //Create the data space for the dataset
12.     dims[0] = 4; dims[1] = 6;
13.     dataspace_id = H5Screate_simple(2, dims, NULL);
```

HDF5 Code Example (2)

Source: https://www.hdfgroup.org/ftp/HDF5/current/src/unpacked/examples/h5_crtdat.c

```
13.     //Create the dataset
14.     dataset_id = H5Dcreate2(file_id, "/dset",
    H5T_STD_I32BE, dataspace_id, H5P_DEFAULT, H5P_DEFAULT,
    H5P_DEFAULT);
15.     //End access to dataset & release resources it uses
16.     status = H5Dclose(dataset_id);
17.     //Terminate access to the data space
18.     status = H5Sclose(dataspace_id);
19.     //Close the file
20.     status = H5Fclose(file_id);
21. }
```

Compiling and Running the HDF5 Code

For Unix platforms, the following compile scripts are included with the binary distribution of the HDF5 software:

h5cc: compile script for HDF5 C programs.

h5fc: compile script for HDF5 F90 programs.

h5c++: compile script for HDF5 C++ programs.

Following are examples of compiling and running an application with the Unix compile scripts:

```
h5fc myprog.f90
```

```
./a.out
```

```
h5cc -o myprog myprog.c
```

```
./myprog
```

Dump of the Output File from the HDF5 Example Program

```
staff$ h5dump dset.h5
```

```
HDF5 "dset.h5" {  
  GROUP "/" {  
    DATASET "dset" {  
      DATATYPE  H5T_STD_I32BE  
      DATASPACE  SIMPLE { ( 4, 6 ) / ( 4, 6 ) }  
      DATA {  
        (0,0): 0, 0, 0, 0, 0, 0,  
        (1,0): 0, 0, 0, 0, 0, 0,  
        (2,0): 0, 0, 0, 0, 0, 0,  
        (3,0): 0, 0, 0, 0, 0, 0  
      }  
    }  
  }  
}
```

Highly Recommend to Explore Parallel HDF5 on your own 😊

<https://www.hdfgroup.org/HDF5/Tutor/parallel.html>

<https://www.hdfgroup.org/HDF5/PHDF5/>

Overview

- Introduction
- I/O During Pre-Processing Stage
 - Choose the right data transfer protocol
 - Remove unnecessary data before processing
- I/O During Processing Stage
 - Understand your parallel file system
 - Use parallel I/O
- **I/O During Post-Processing Stage**
 - Move your data to secondary or tertiary storage media
- An Example of a System for Data Intensive Computing
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Storage, Archival, and Information Visualization

- After your application has finished running, you might need to move the data involved to secondary or tertiary storage
- Depending upon the analysis that you might be doing, you could be generating multiple data products from your dataset
 - Preserve the raw data and the algorithms to generate the data products for data provenance purposes
 - Note: what might be noise for you could be useful data for someone else
 - As per your need (which should already be explained in the data management plan), you might want to retain different data products for different periods of time

Overview

- Introduction
- I/O During Pre-Processing Stage
 - Choose the right data transfer protocol
 - Remove unnecessary data before processing
- I/O During Processing Stage
 - Understand your parallel file system
 - Use parallel I/O
- I/O During Post-Processing Stage
 - Move your data to secondary or tertiary storage media
- **An Example of a System for Data Intensive Computing**
 - Interesting features: array of about 100,000 NAND flash dies - a 3-D RAID configuration to tolerate failures

Hadoop

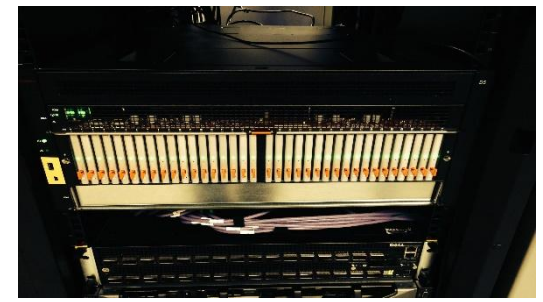
- In ~2011, we discovered an exciting new failure mode in large-scale systems
- We called this failure mode “Hadoop”
- Recipe for successful Hadoop failure: take a big system
 - A huge central filesystem
 - Optimized for large, sequential, access
 - With a highly tuned, low-level C interface
- And on that run software that:
 - Assumes a small, massively distributed filesystem
 - Optimized for very small files
 - With an untuned, well... Java
- Results: Deployed Rustler to keep such users off the supercomputers

Wrangler: An XSEDE Resource for Data Intensive Computing

Wrangler provides many different services to help researchers solve their data computing needs, and has

- Massive, replicated, secure high performance data storage (10PB each at Indiana and TACC)
- A large scale flash storage tier for analytics, with bandwidth of 1TB/s and 250M IOPS (6x faster than Stampede)
- Embedded processing of more than 3,000 processors cores for data analysis
- Flexible support for a wide range of data workflows, including those using Hadoop and databases
- Integration with Globus Online services for rapid and reliable data transfer and sharing
- A fully scalable design that can grow with the amount of users and as data applications grow.

DSSD Storage



- The flash storage provides the truly “innovative capability” of Wrangler
- Not SSD; a custom interface allows access to the NAND flash technology performance without the overhead of the traditional “disk” interface
- Opportunity to explore APIs that integrate natively with apps (*i.e.*, HDFS direct integration)
- Half a petabyte of usable space
- Nearly 100K NAND flash dies
- 960 Gen3 x4 PCI links to the storage system

A brief intro to Hadoop and Spark



JOSE NANDEZ
jnandez@sharcnet.ca

SHARCNET, UNIVERSITY OF WESTERN ONTARIO

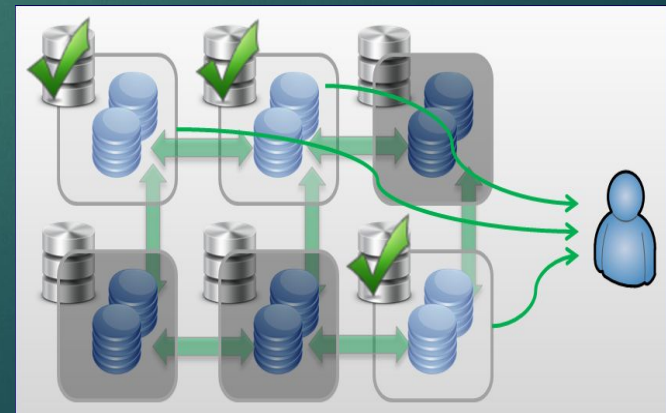
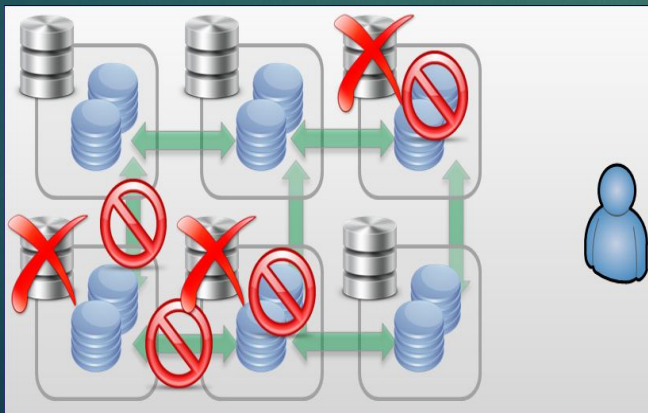
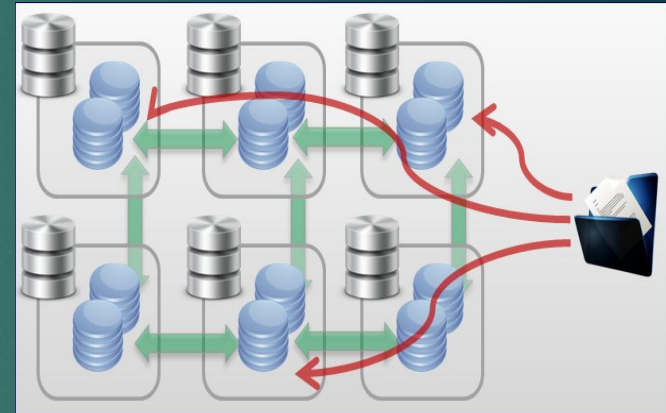
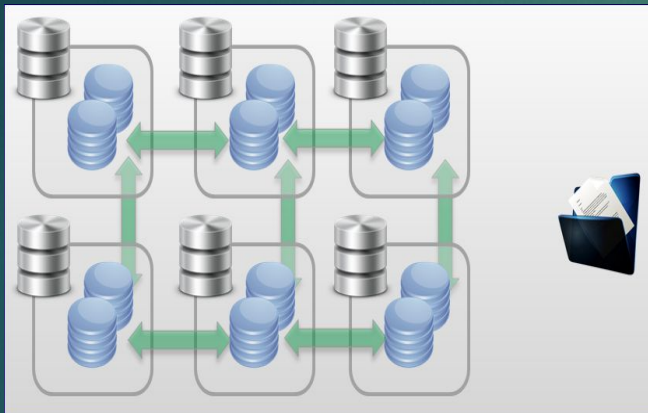
COMPUTE CANADA

INTERNATIONAL HPC SUMMER SCHOOL 2016

What is apache Hadoop?



- A framework (written in Java) that enables distributed processing of large datasets across cluster of computers.
- It scales up from single servers to thousands of machines
- It is fault tolerant (applies replication to the data)
- It follows the idea of moving compute to data NOT moving data to compute.
- Hadoop will have many commodity servers to process the data, and uses the MapReduce Model.

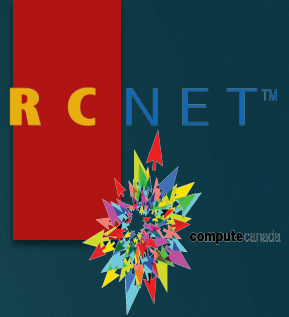


What is Apache Spark?



- Apache Spark or (just Spark, written in Scala Programming Language) is a fast and general engine for processing large-scale datasets
- Spark extends the MapReduce model, supporting interactive queries and stream processing
- Spark has the ability to run computations in memory or disk (MapReduce) depending on the complexity of the problem
- Spark is designed to work on iterative algorithms, interactive queries, and Streaming.
- It offers Python, Scala, Java, SQL and R APIs, as well a Machine Learning, and Graph libraries.
- It run on Hadoop, Amazon EC2, in a Cloud, Standalone.

Hadoop - python scripts for Word



Count

```
#!/usr/bin/python
#mapper function for hadoop
import re
import sys

def main():
    for line in sys.stdin:
        words = re.split('\W+',line.strip().
lower())
        for word in words:
            print '%s\t%s' % (word,1)

if __name__ == "__main__":
    main()
```

```
#!/usr/bin/python
#reducer function for Hadoop
import sys

def main():
    word_count = {}
    for line in sys.stdin:
        line = line.strip()
        try:
            word,count = line.split('\t',1)
        except:
            continue
        try:
            count = int(count)
        except ValueError:
            continue
        try:
            word_count[word] += count
        except:
            word_count[word] = count
    for word in word_count.keys():
        print '%s\t%s' % (word,
word_count[word])

if __name__ == "__main__":
    main()
```

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -file .
/mapper.py -mapper ./mapper.py -file ./reducer.py -reducer ./reducer.py -input
/user/hduser/gutenberg/* -output /user/hduser/gutenberg_hadoop_output
```

Spark-python script for WordCount



```
#!/usr/local/spark/bin/spark-submit  
from pyspark import SparkContext  
import re
```

```
def main():
```

```
    sc = SparkContext(appName="WordCount")
```

```
    data = sc.textFile("hdfs://localhost:54310/user/hduser/gutenberg") #load the data
```

```
    mapper = data.flatMap(lambda x: re.split('\W+',x.strip().lower())) #split the words
```

```
    key_value = mapper.map(lambda x: (x,1)) #create key-value pair (similar than
```

```
mapper.py
```

```
    reducer = key_value.reduceByKey( lambda x,y: x+y) #sum occurrences of each word
```

```
#similar than reducer.py
```

```
    reducer.saveAsTextFile("hdfs://localhost:
```

```
54310/user/hduser/gutenberg_spark_output") #writes to disk the output
```

```
if __name__ == "__main__":
```

```
    main()
```

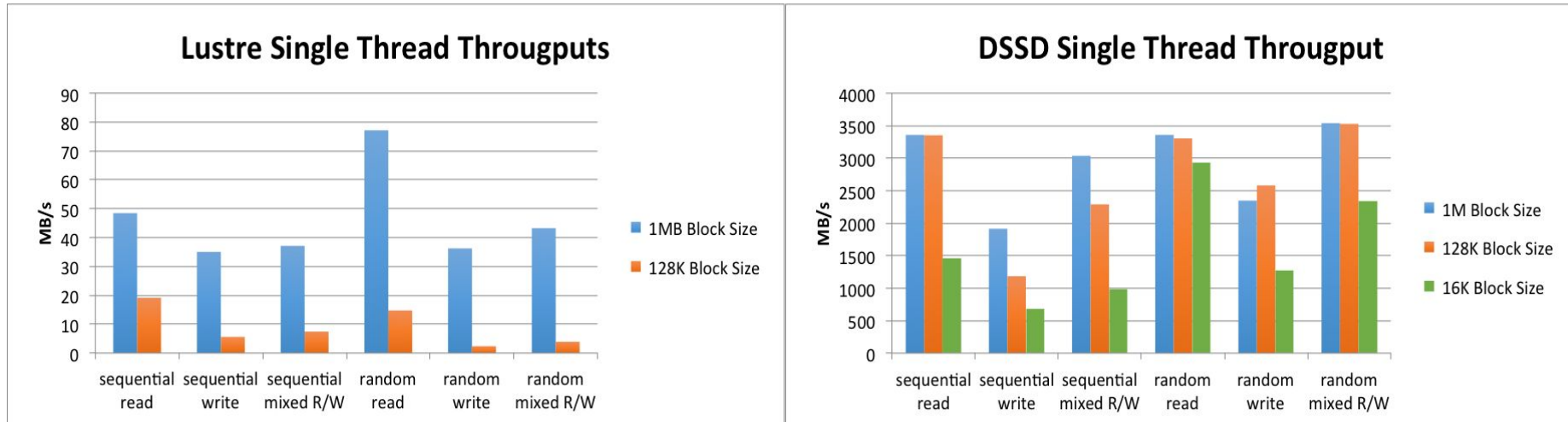
```
$ spark-submit
```

```
spark_word_count.py
```

References

- <https://dennyglee.com/2012/01/31/moving-data-to-compute-or-compute-to-data-that-is-the-big-data-question/>
- <http://spark.apache.org/>
- <http://hadoop.apache.org/>
- <https://www.gutenberg.org/>

Where DSSD Really Shines



- Single thread IO for different block sizes
 - Flash is faster than single spinning disk (no surprise)
 - DSSD sustains most throughput for small block sizes and for sequential and random I/O patterns



Wrangler in the TACC Ecosystem

- TACC is traditionally a provider of HPC, Visualization, and storage systems and we still are
- But new communities provide kinds of data-intensive problems our HPC systems just aren't built for
 - Run Hadoop on your favorite supercomputer to see what we need
 - Or do a bunch of random access to a bunch of really small files
- Wrangler is not to replace our supercomputer, visualization, or cloud offerings; it supplements this environment.

References

[.org/HDF5/Tutor/introductory.html](#)

:
[cs.tennessee.edu/computing-resources/file-systems/io-lustre-fundamentals](#)

[com/TACC/t3pio](#)

o Parallel I/O:
[cf.ornl.gov/wp-content/uploads/2011/10/Fall_IO.pdf](#)

o Parallel I/O and MPI-IO by Rajeev Thakur

State-of-the-Art Parallel File Systems for Linux
[linuxclustersinstitute.org/conferences/archive/2004/PDF/20-](#)

Thanks!

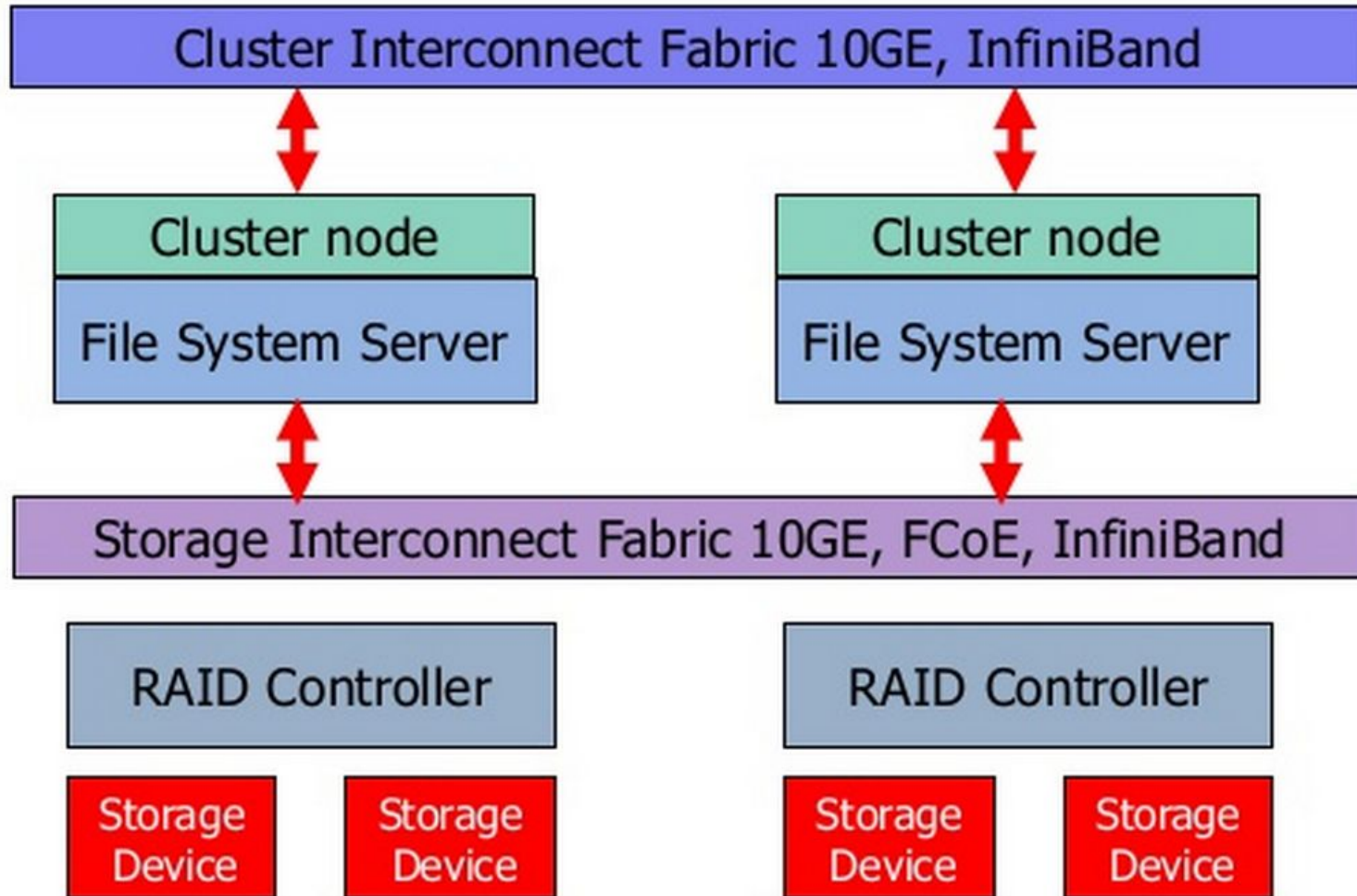
Questions or Comments?

Compute-Intensive Applications Versus Data Intensive Applications

- **Compute-Intensive Applications** devote most of their execution time to computational requirements
- **Data-Intensive Applications** devote most of their execution time to I/O and manipulation of data, and read/write large volumes of data, for example:
 - Running simulations for studying climate change over last 10 years
 - eDiscovery

GPFS Topology 1

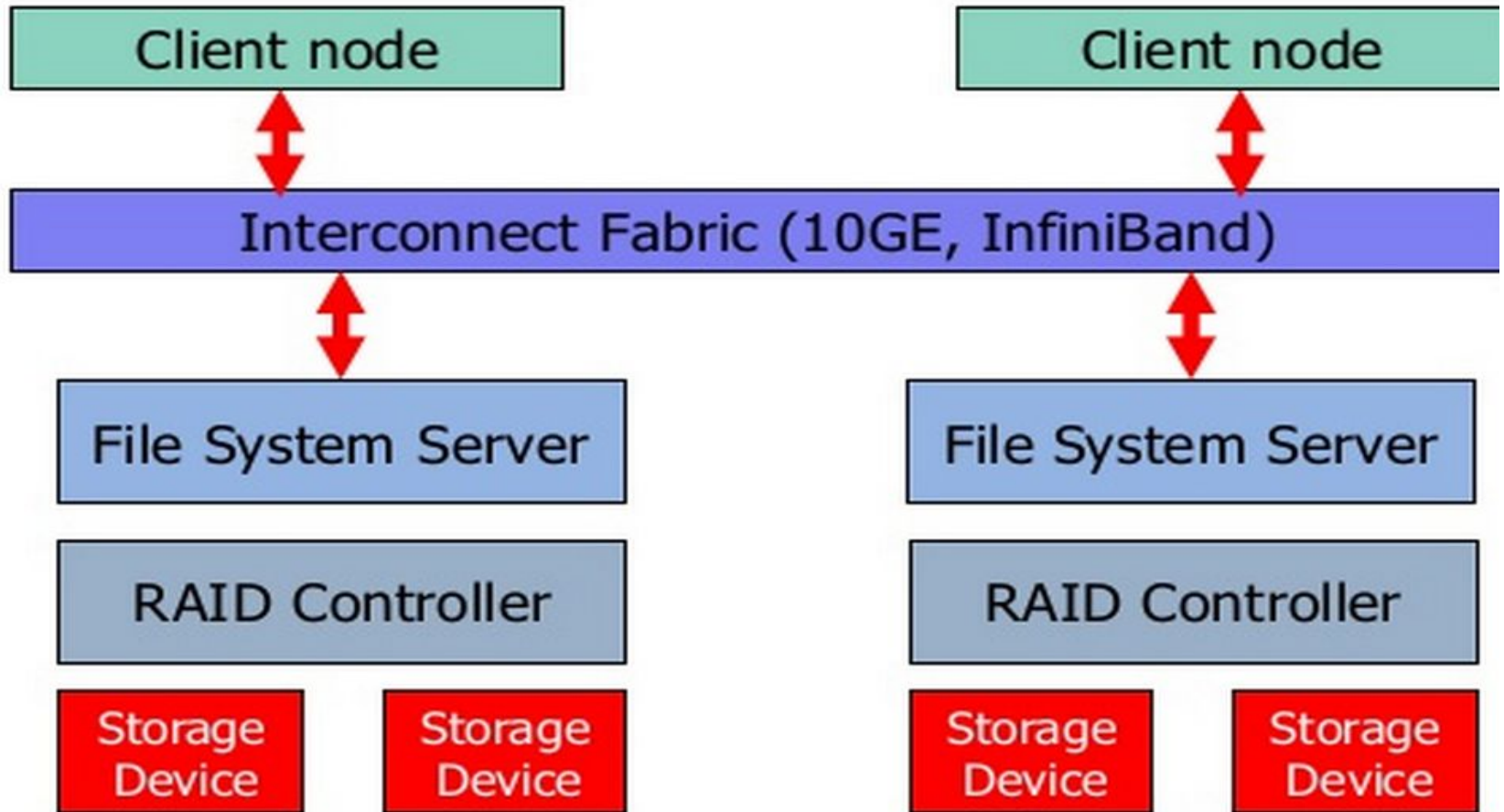
Direct Attached Storage



Source: <http://www.slideshare.net/GabrielMateescu/sonas-44390281>

GPFS Topology 2

Network Attached Storage



Source: <http://www.slideshare.net/GabrielMateescu/sonas-44390281>

GPFS versus Lustre

	GPFS	Lustre
MDS	In direct-attached storage topology, all nodes acts like MDS, whereas in network-attached topology, some nodes (server nodes) act like MDS	Often 1 primary + 1 failover; since version 2.4, supported for clustered MDS is available
Storage Type	RAID, SAN, ...	RAID, SAN, ...
User Control for Tuning	None; optimized by administrators at the time of installation	User can change some parameters like stripe size and stripe count
Daemon Communication	TCP/IP	Portal
License	Proprietary (IBM product)	Open-Source

Source: Reference 6

Note about atomicity Read/Write

```
int MPI_File_set_atomicity ( MPI_File mpi_fh, int flag );
```

- Use this API to set the atomicity mode – 1 for true and 0 for false – so that only one process can access the file at a time
- When atomic mode is enabled, MPI-IO will guarantee sequential consistency and this can result in significant performance drop
- This is a collective function

This example creates a parallel HDF5 file (1)

```
1.  #include "hdf5.h"
2.  #define H5FILE_NAME "SDS_row.h5"
3.  int main (int argc, char **argv) {
4.      // HDF5 APIs definitions
5.      hid_t file_id;
6.      //file and dataset identifiers
7.      hid_t plist_id; //property list identifier
8.      herr_t status;
9.      //MPI variables
10.     int mpi_size, mpi_rank;
11.     MPI_Comm comm = MPI_COMM_WORLD;
12.     MPI_Info info = MPI_INFO_NULL;
13.     //Initialize MPI
14.     MPI_Init(&argc, &argv);
```

This example creates a parallel HDF5 file (2)

```
15.     MPI_Comm_size(comm, &mpi_size);
16.     MPI_Comm_rank(comm, &mpi_rank);
17.     //Setup file access property list with parallel I/O access
18.     plist_id = H5Pcreate(H5P_FILE_ACCESS);
19.     H5Pset_fapl_mpio(plist_id, comm, info);
20.     //Create a new file collectively.
21.     file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC,
H5P_DEFAULT, plist_id);
22.     //Close property list.
23.     H5Pclose(plist_id);
24.     // Close the file.
25.     H5Fclose(file_id);
26.     MPI_Finalize();
27.     return 0;
28. }
```