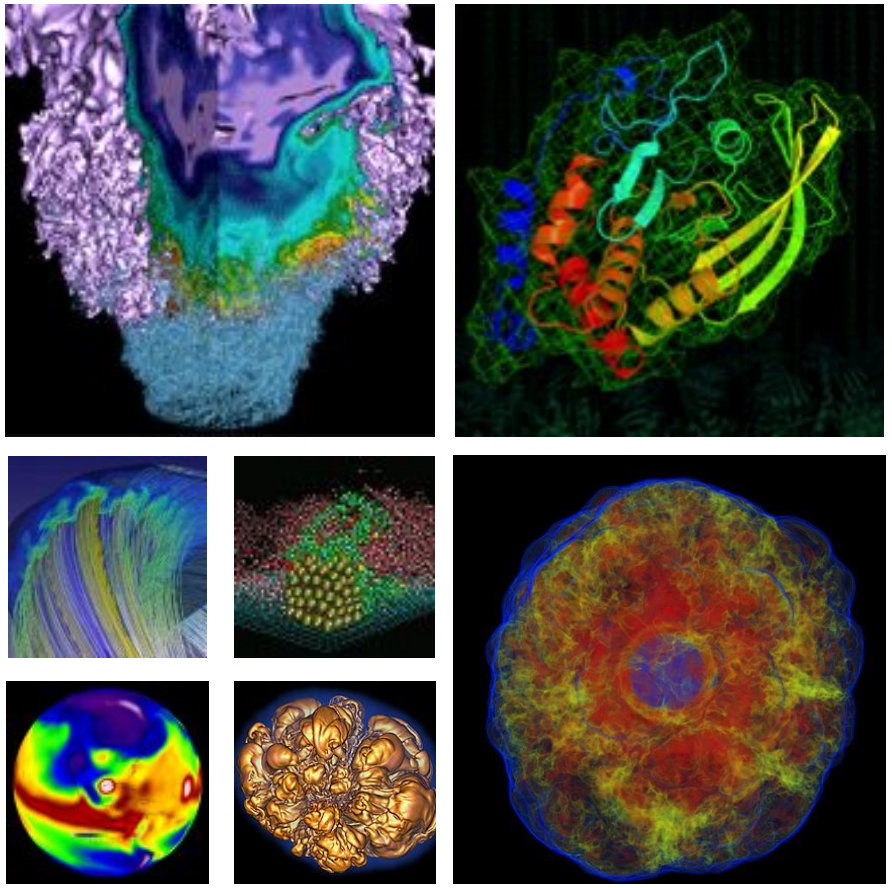
Accelerate your IO with the Burst Buffer



Debbie Bard
Data and Analytics Services
NERSC

SC17 Tutorial

U.S. DEPARTMENT OF ENERGY | Office of Science
BERKELEY LAB

Get started!

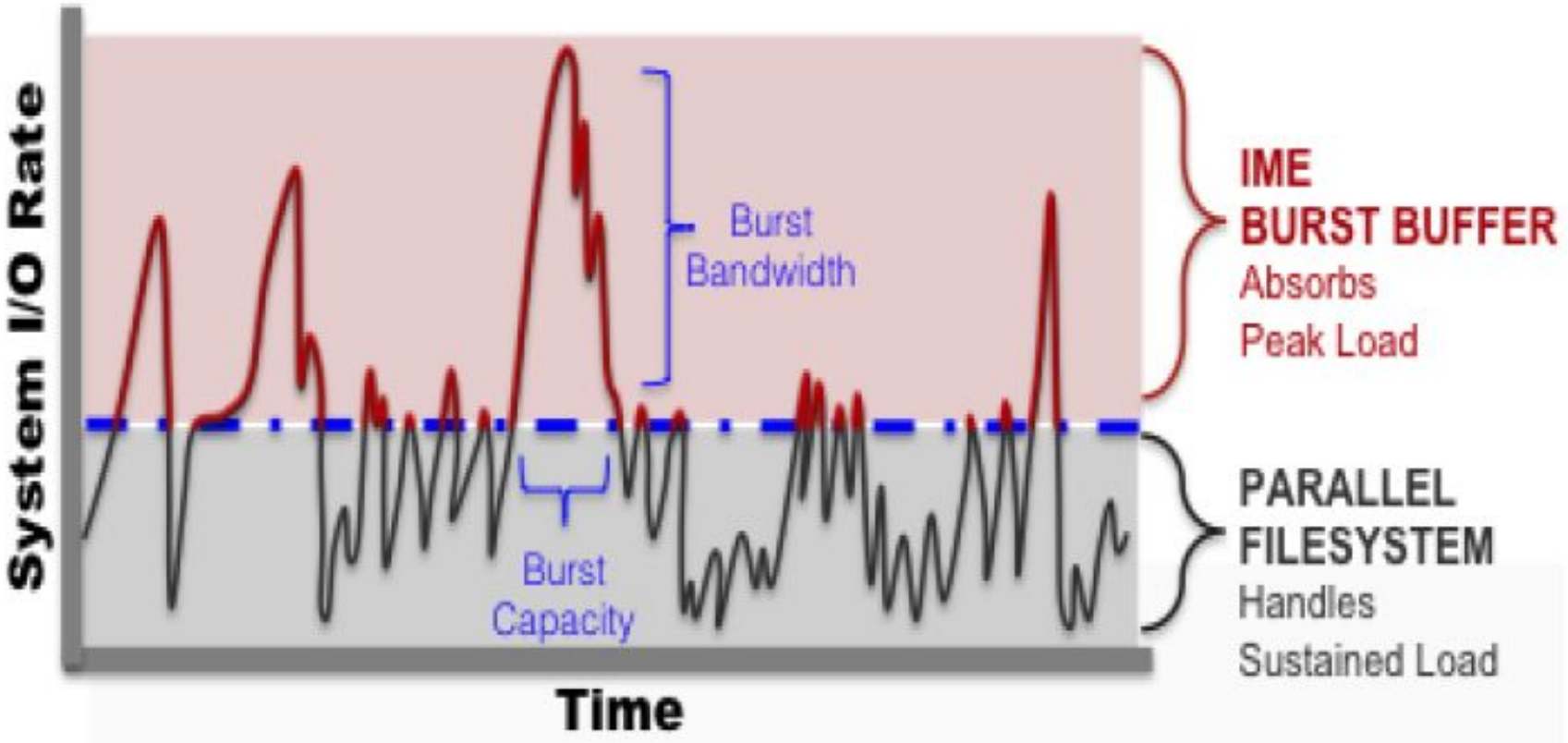
- **Get a NERSC training account!**
  - Valid for today only
  - Fill out the “appropriate use” form!
- **Log on; go to your scratch space**
  - ssh -Y trainXXX@cori.nersc.gov
  - cd $SCRATCH
- **Pull down the training repo:**
  - git clone https://github.com/KAUST-KSL/SC17_BB_Tutorial.git
- **Copy over some data (this will take a little while)**
  - cd SC17_BB_Tutorial/intro
  - bash copy.sh
- **Browse the simple example scripts...**

SSD vs HDD

- **Spinning disk has mechanical limitation in how fast data can be read from the disk**
  - SSDs do not have the physical drive components so will always read faster
  - Problem exacerbated for small/random reads
  - But for large files striped over many disks e.g. via Lustre, HDD still performs well.
- **But SSDs are *expensive*!**
- **SSDs have limited RWs – the memory cells will wear out over time**
  - This is a real concern for a data-intensive computing center like NERSC.

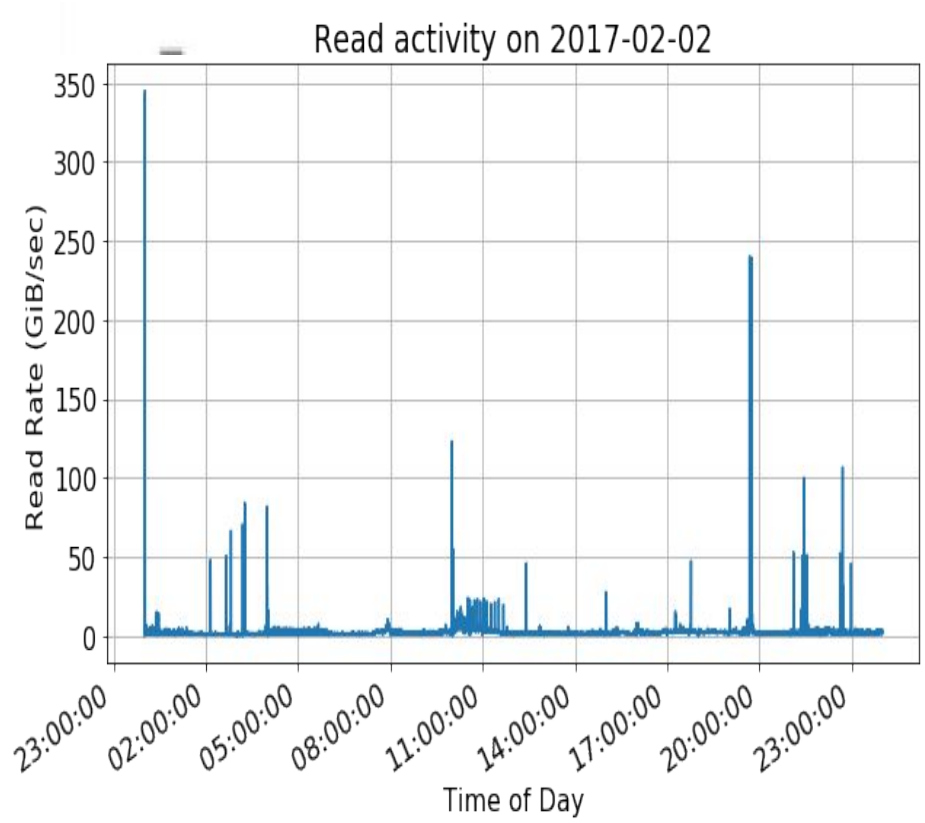
Why an SSD Burst Buffer?

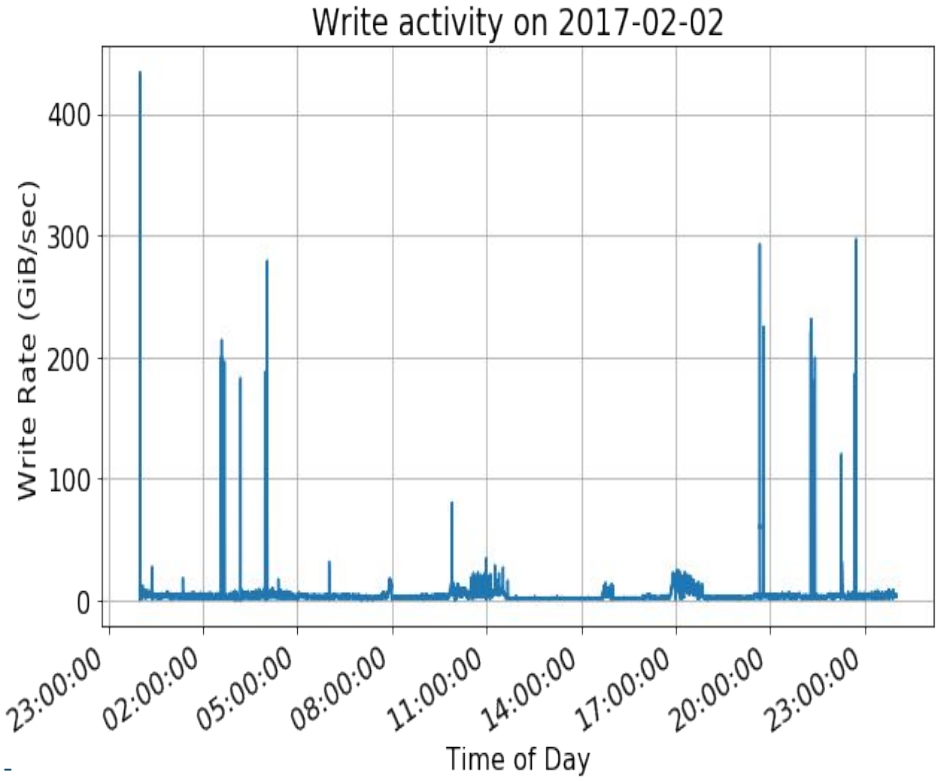
- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts



Why an SSD Burst Buffer?

- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts





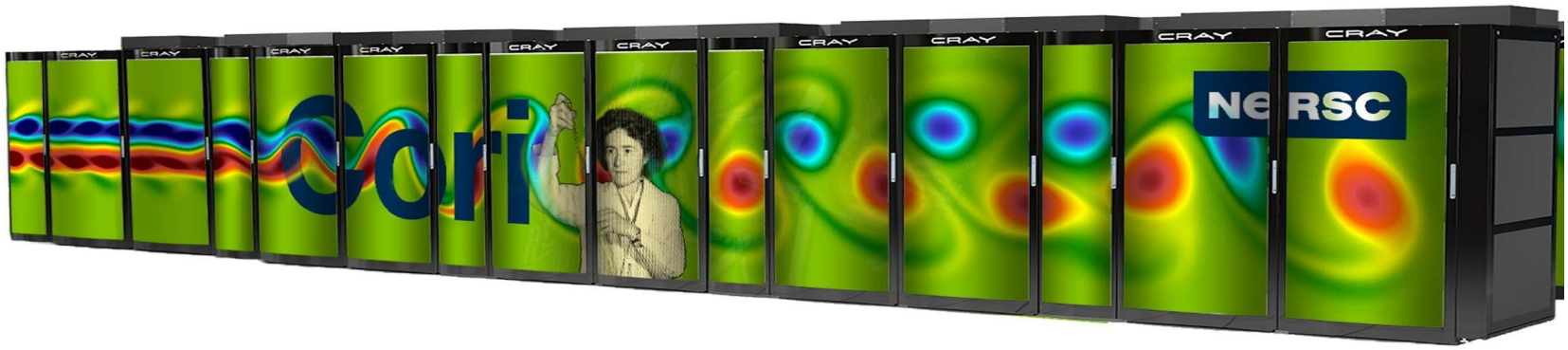
Why an SSD Burst Buffer?

- **Motivation: Handle spikes in I/O bandwidth requirements**
  - Reduce overall application run time
  - Compute resources are idle during I/O bursts
- **Some user applications have challenging I/O patterns**
  - High IOPs, random reads, different concurrency… perfect for SSDs
- **Cost rationale: Disk-based PFS bandwidth is expensive**
  - Disk capacity is relatively cheap
  - SSD *bandwidth* is relatively cheap

    =>Separate bandwidth and spinning disk
    - Provide high BW without wasting PFS capacity
    - Leverage Cray Aries network speed

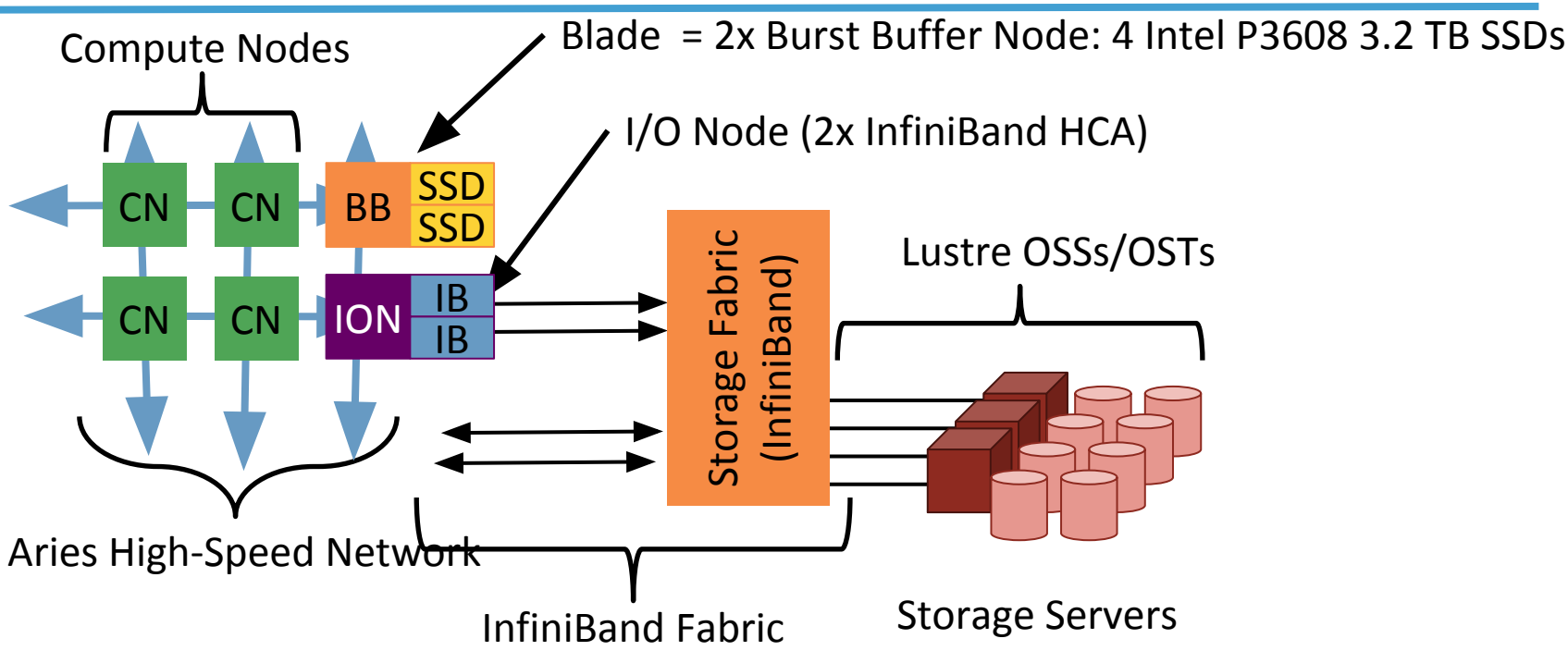
Cori @ NERSC

- **NERSC at LBL, production HPC center for DoE**
  - >6000 diverse users across all DoE science domains
- **Cori – NERSCs Newest Supercomputer – Cray XC40**
  - 2,388 Intel Haswell dual 16-core nodes
  - 9,688 Intel Knights Landing Xeon Phi nodes, 68 cores
- **Cray Aries high-speed "dragonfly" topology interconnect**
- **Lustre Filesystem: 27 PB ; 248 OSTs; 700 GB/s peak performance**
- **1.8PB of Burst Buffer**



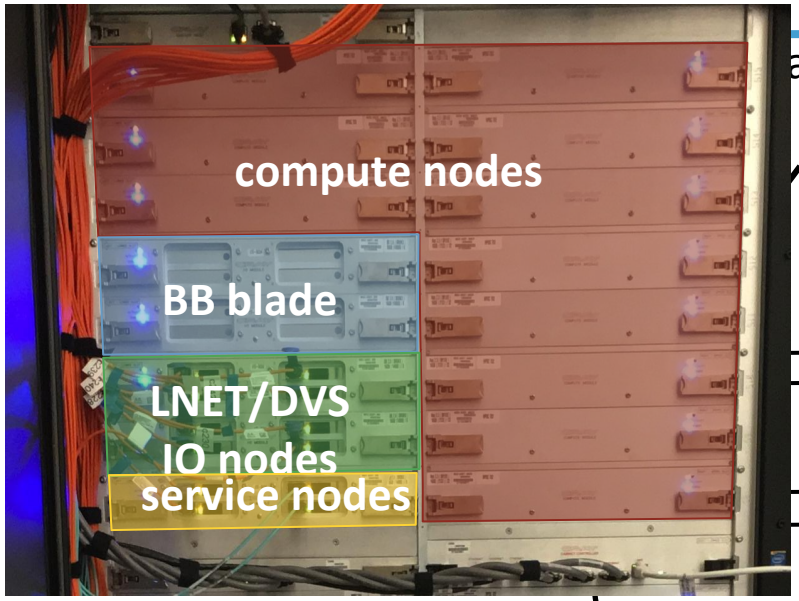
BERKELEY LAB
Lawrence Berkeley National Laboratory

Burst Buffer Architecture



- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or 'persistent').
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on reservation size requested)

Burst Buffer Architecture



ade = 2x Burst Buffer Node: 4 Intel P3608 3.2 TB SSDs

I/O Node (2x InfiniBand HCA)

Storage Fabric (InfiniBand)

Lustre OSSs/OSTs

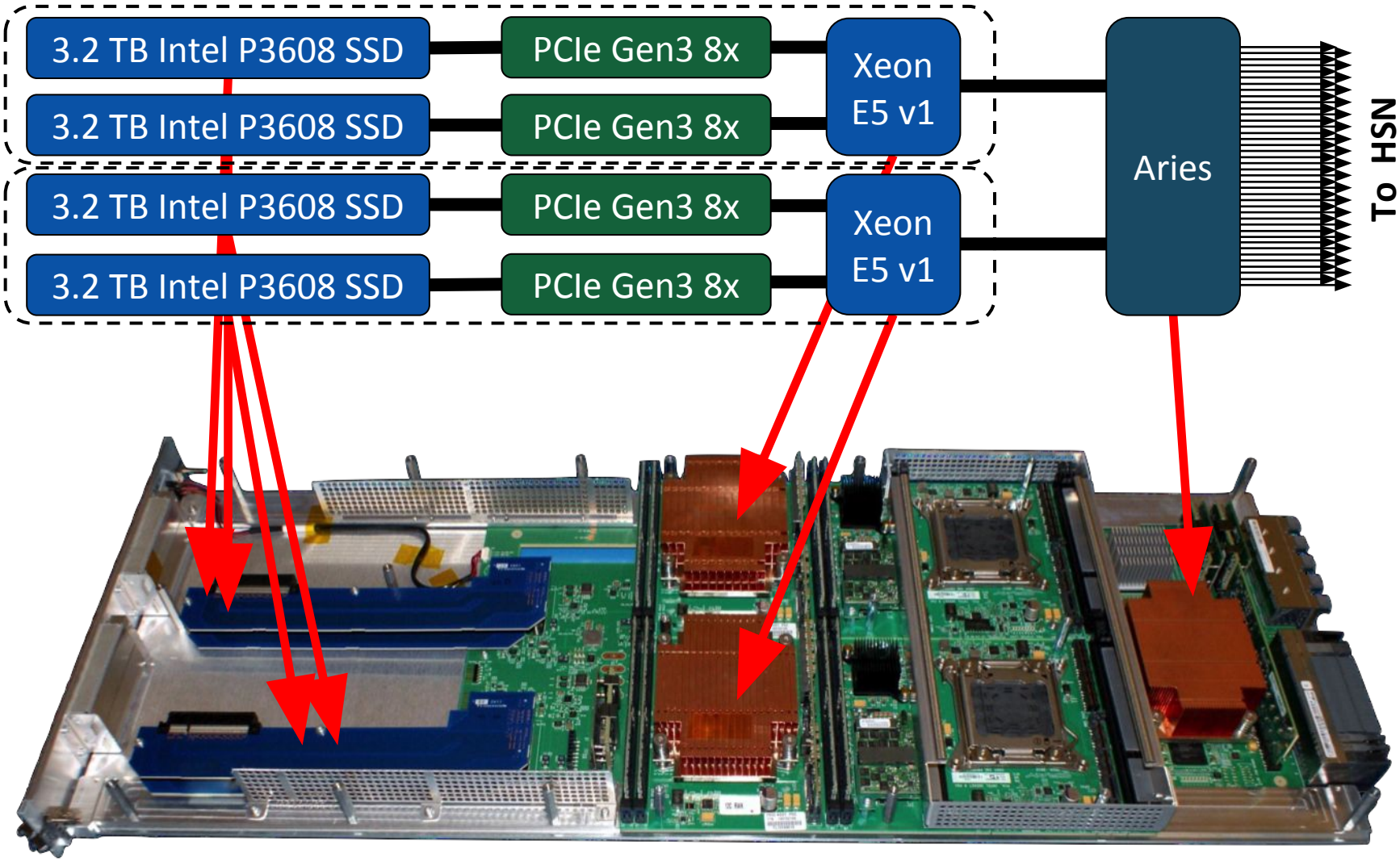
Aries High-Speed Network

InfiniBand Fabric

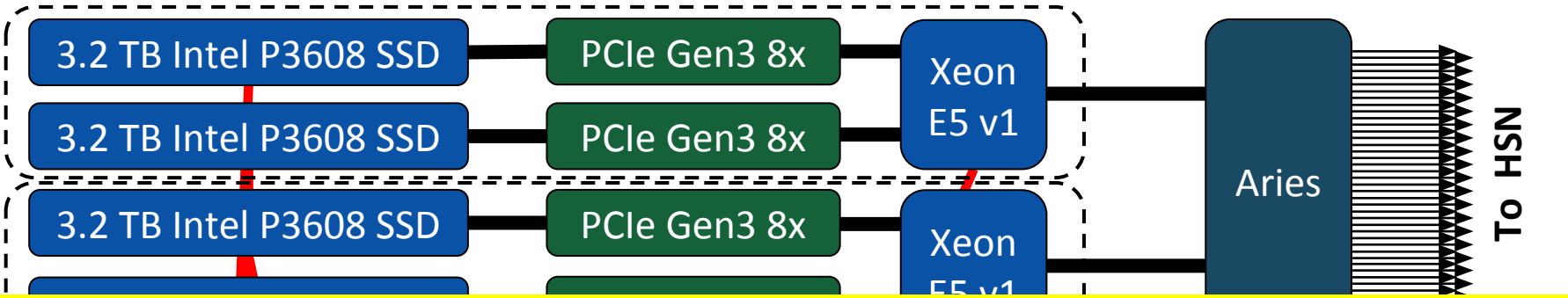
Storage Servers

- DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or 'persistent').
- Users see a POSIX filesystem
- Filesystem can be striped across multiple BB nodes (depending on reservation size requested)

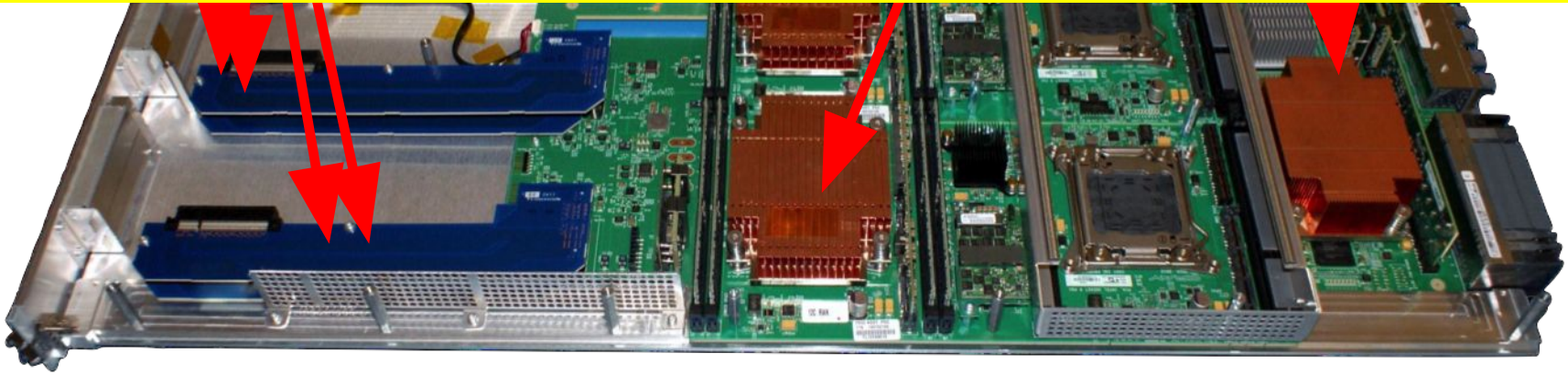
Burst Buffer Blade = 2xNodes



Burst Buffer Blade = 2xNodes

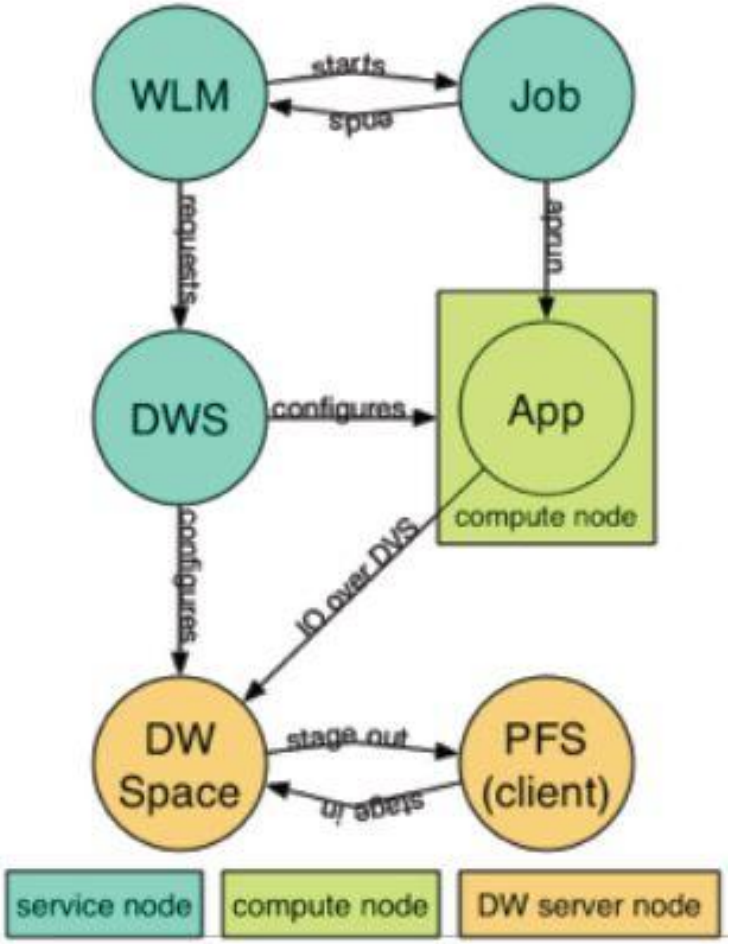


- ~1.8PiB of SSDs over 288 nodes
- Accessible from both HSW and KNL nodes
- NOT node-local SSDs!



DataWarp: Under the hood

- Workload Manager (Slurm) schedules job in the queue on Cori
- DataWarp Service (DWS) configures DW space and compute node access to DW
- DataWarp Filesystem handles stage interactions with PFS (Parallel File System, i.e. scratch)
- Compute nodes access DW via a mount point

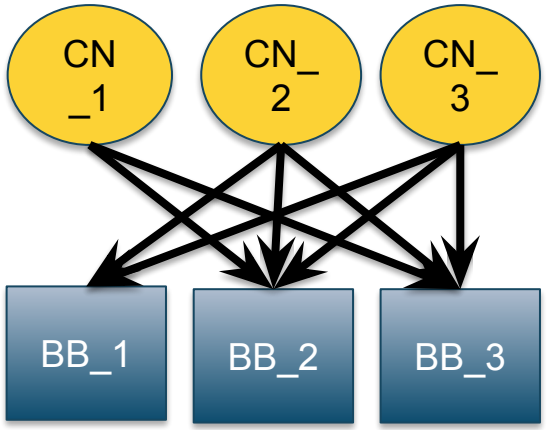


Two kinds of DataWarp Instances

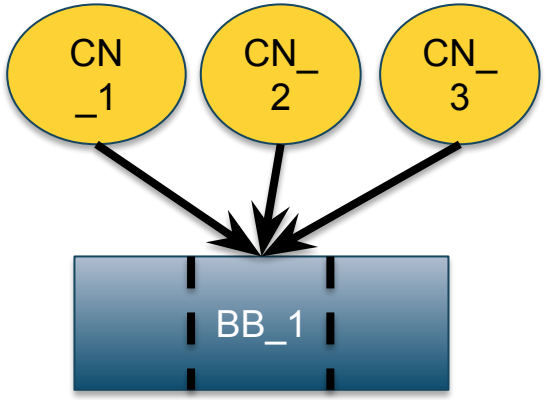
- **"Instance": a reservation on the BB**
- **Can it be shared? What is its lifetime?**
  - **Per-Job Instance**
    - Can only be used by compute job that creates it
    - Lifetime is the same as the creating job
    - Use cases: PFS staging, application scratch, checkpoints
  - **Persistent Instance**
    - Can be used by any job (subject to UNIX file permissions)
    - Lifetime is controlled by creator
    - Use cases: Shared data, Coupled job workflow
    - ***NOT for long-term storage of data!***

Two DataWarp Access Modes

- **Striped ("Shared")**
  - Files are striped across all DataWarp nodes
  - Files are visible to all compute nodes
    Aggregates both capacity and BW per file
  - One DataWarp node elected as the metadata server (MDS)



- **Private**
  - Files are assigned to one or more DataWarp node (can chose to stripe)
  - File are visible to *only the compute node that created them*
  - Each DataWarp node is an MDS for one or more compute nodes



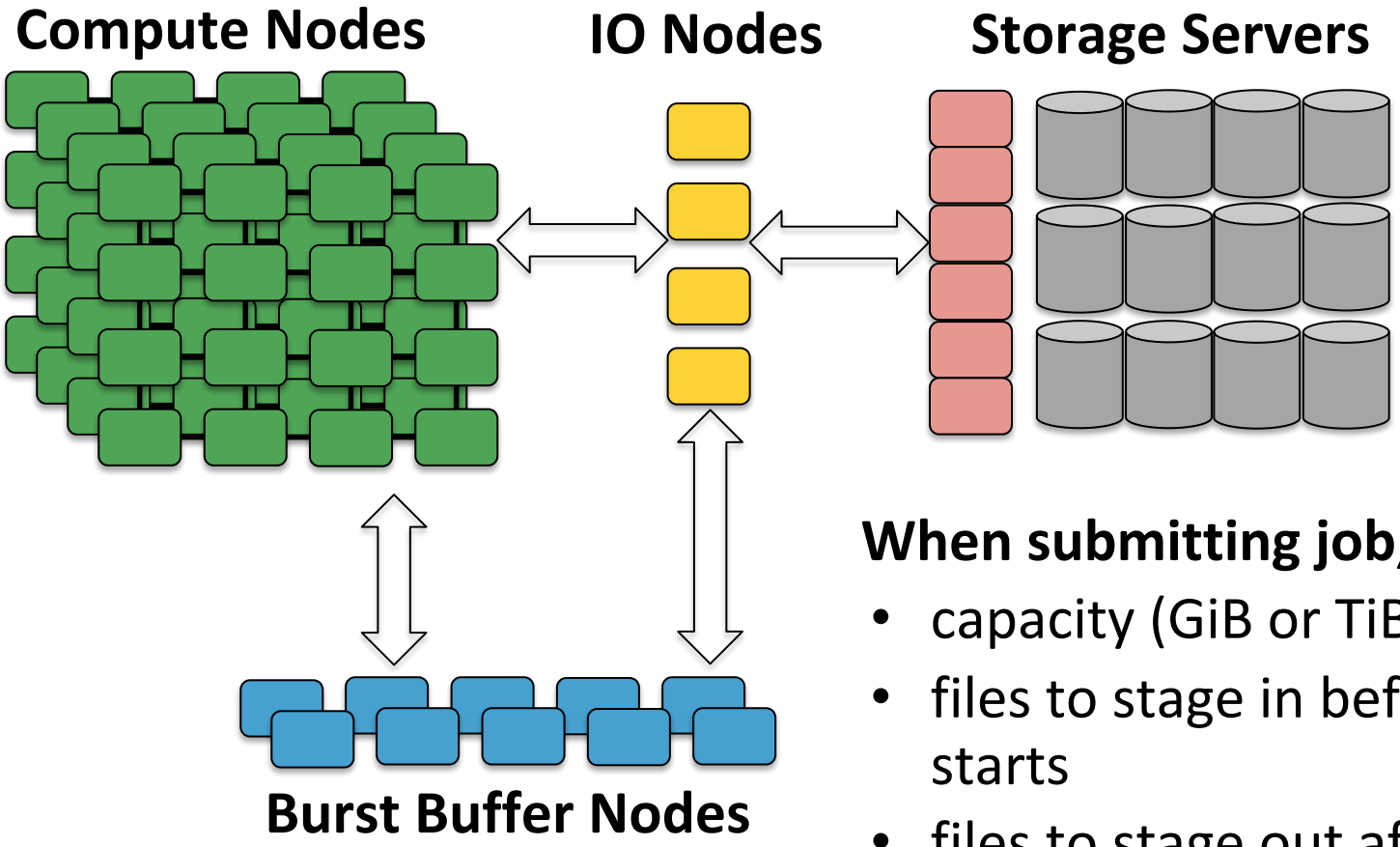
Striping, granularity and pools

- **DataWarp nodes are configured to have "granularity"**
  - Minimum amount of data that will land on one node
- **Two "pools" of DataWarp nodes on Cori right now, with different granularity**
  - wlm_pool (default): 82GiB
    - `#DW jobdw capacity=1000GB access_mode=striped type=scratch pool=wlm_pool`
  - sm_pool: 20.14 GiB
    - `#DW jobdw capacity=1000GB access_mode=striped type=scratch pool=sm_pool`
- **For example, 1.2TiB will be striped over 15 BB nodes in wlm_pool, but over 60 BB nodes in sm_pool**
  - No guarantee that allocation will be spread evenly over SSDs - may see >1 "grain" on a single node

I/O PFS ↔ BB

- **Each DataWarp node separately manages all PFS I/O for the files or stripes it contains**
  - ***Striped***: each DW node has a stripe of a file, multiple PFS clients per file
  - ***Private***: if not "private, striped", each DW node has an entire file, one PFS client per node
- **So I/O to PFS from DW is automatically done in parallel**
  - Note that at present, can only access PFS (i.e. $CSCRATCH) from BB
- ***Compute nodes are not involved with this PFS I/O***

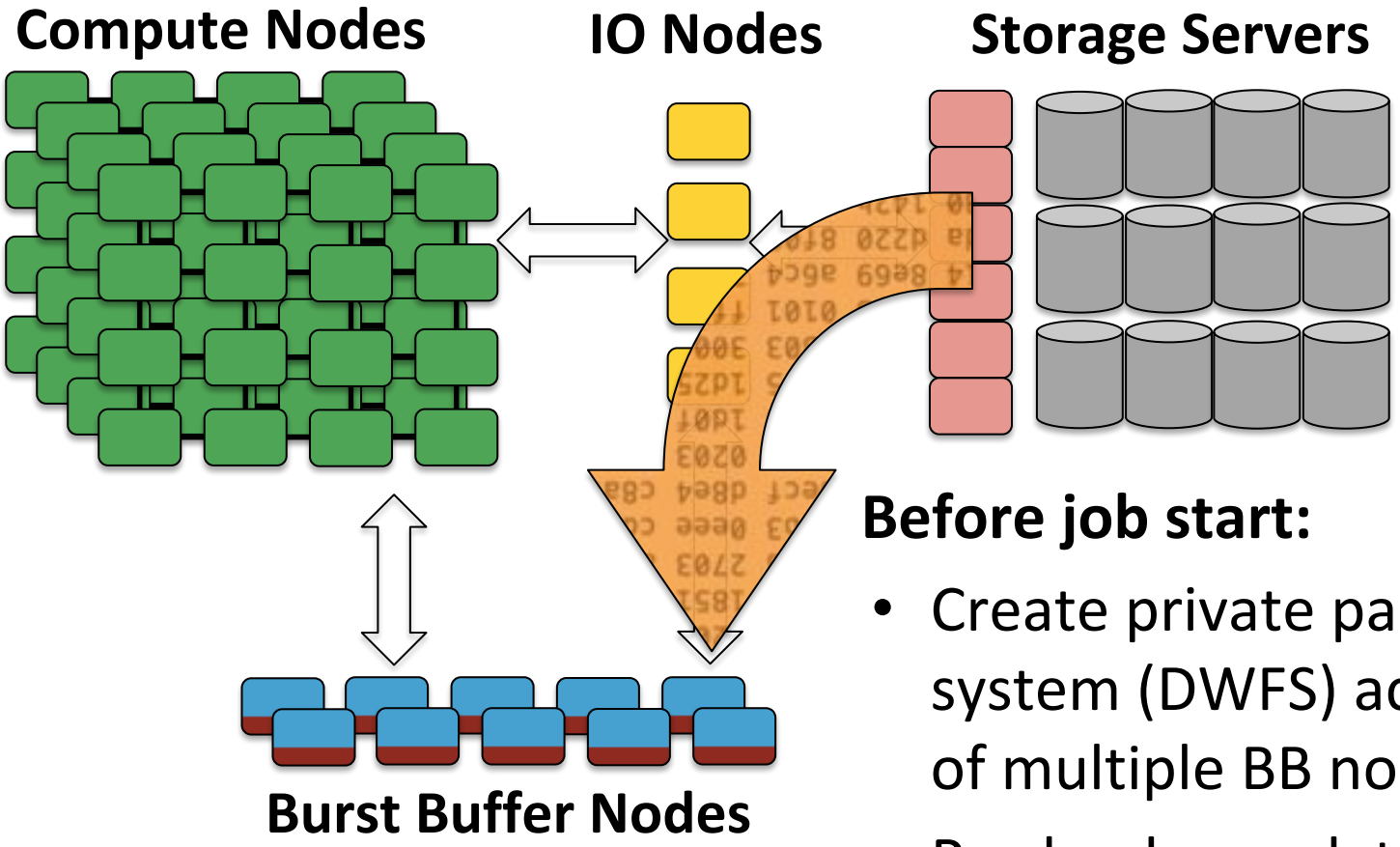
Cori's Data Paths



When submitting job, request:

- capacity (GiB or TiB)
- files to stage in before job starts
- files to stage out after job finishes

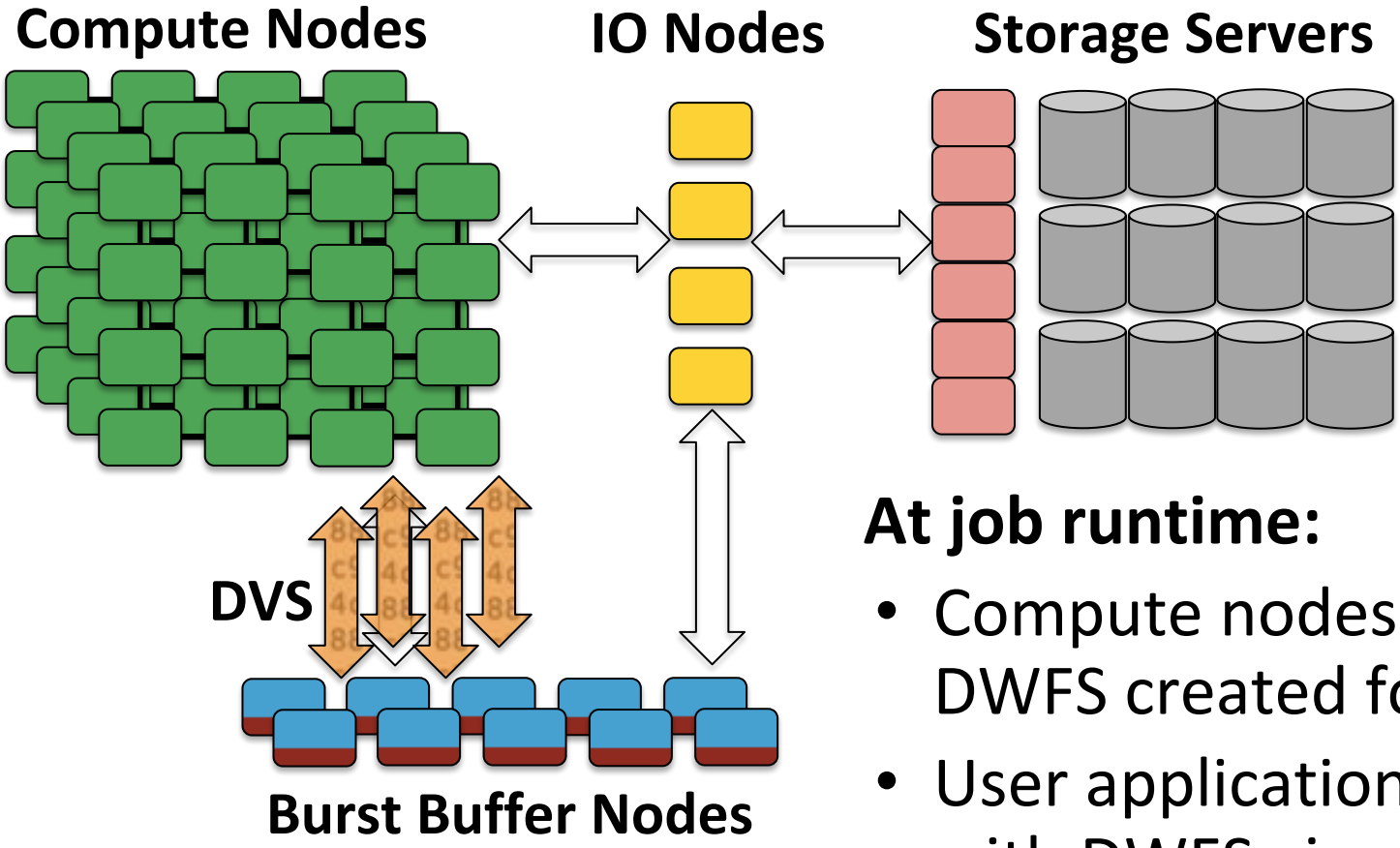
Cori's Data Paths



Before job start:

- Create private parallel file system (DWFS) across parts of multiple BB nodes
- Pre-load user data into this DWFS

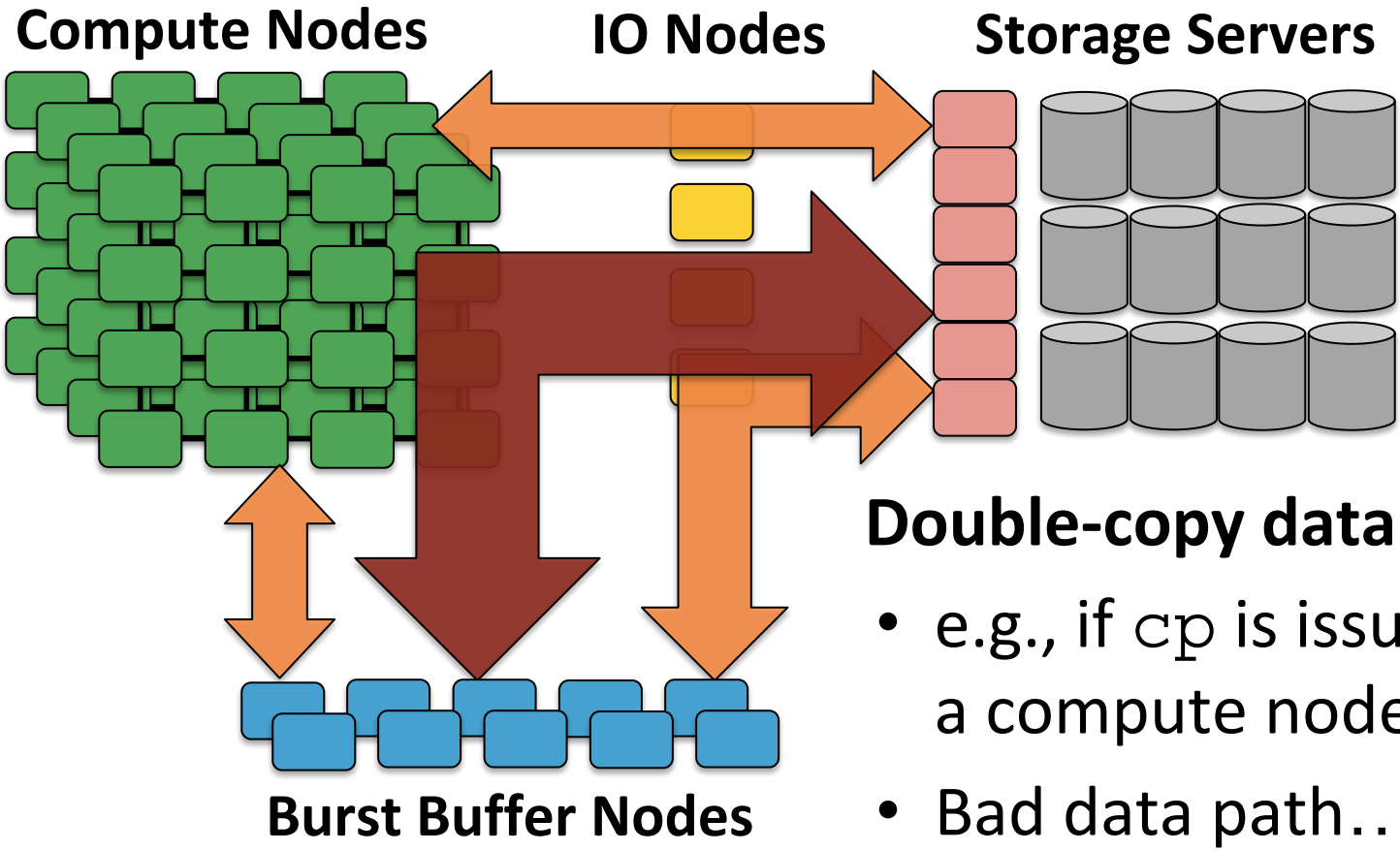
Cori's Data Paths



At job runtime:

- Compute nodes mount DWFS created for job
- User application interacts with DWFS via standard POSIX I/O

Cori's Data Paths



Double-copy data path

- e.g., if `cp` is issued from a compute node
- Bad data path…except when #CN >> #BBNs

How to use DataWarp

- **Principal user access: SLURM Job script directives: `#DW`**
  - Allocate job or persistent DataWarp space
  - Stage files or directories in from PFS to DW; out DW to PFS
  - Access BB mount point via `$DW_JOB_STRIPED`, `$DW_JOB_PRIVATE`, `$DW_PERSISTENT_STRIPED_name`
- **We'll go through this in more detail later….**
- **User library API – `libdatawarp`**
  - Allows direct control of staging files asynchronously
  - C library interface
  - https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-8
  - https://github.com/NERSC/BB-unit-tests/tree/master/datawarpAPI

Benchmark Performance on Cori

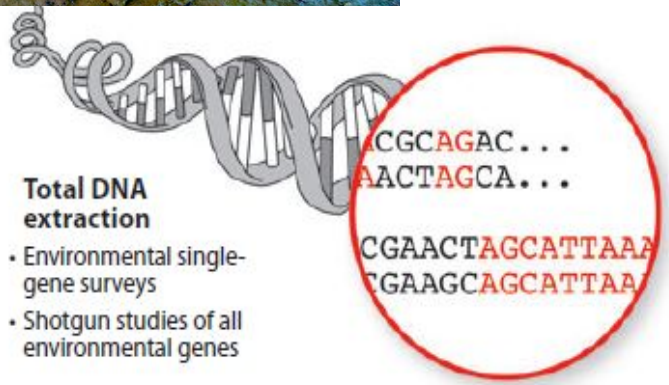
- **Burst Buffer is now doing very well against benchmark performance targets**
  - Out-performs Lustre significantly
  - One of the fastest IO systems in the world!

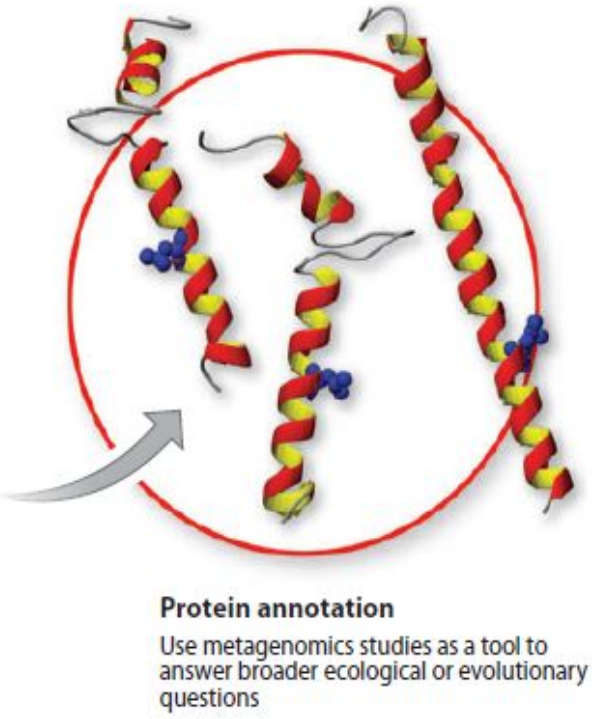
	IOR Posix FPP		IOR MPIO Shared File		IOPS	
	Read	**Write**	**Read**	**Write**	**Read**	**Write**
Best Measured (287 Burst Buffer Nodes : 11120 Compute Nodes; 4 ranks/node)*	1.7 TB/s	1.6 TB/s	1.3 TB/s	1.4 TB/s	28M	13M

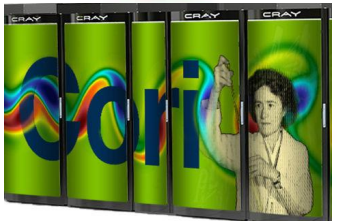
**Bandwidth tests: 8 GB block-size 1MB transfers IOPS tests: 1M blocks 4k transfer*

Success story: JGI

- **Metagenome assembly**



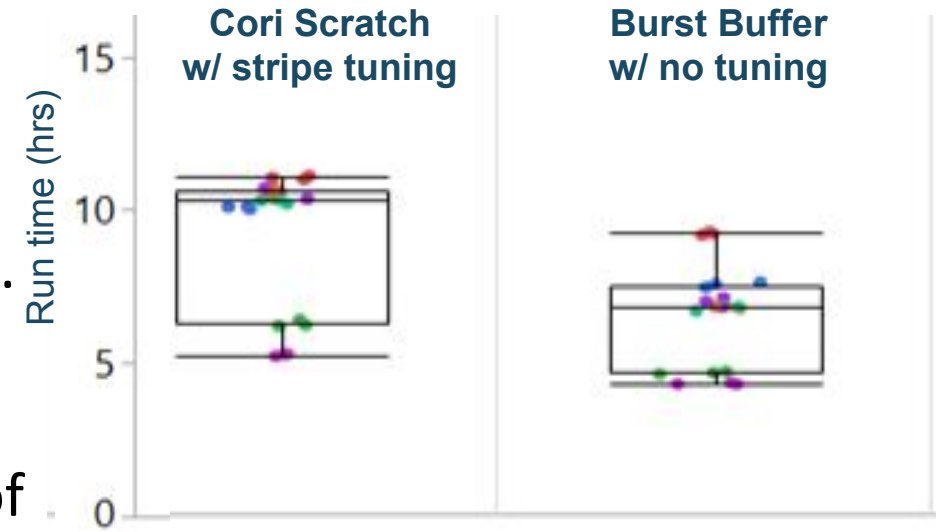




U.S. DEPARTMENT OF ENERGY | Office of Science

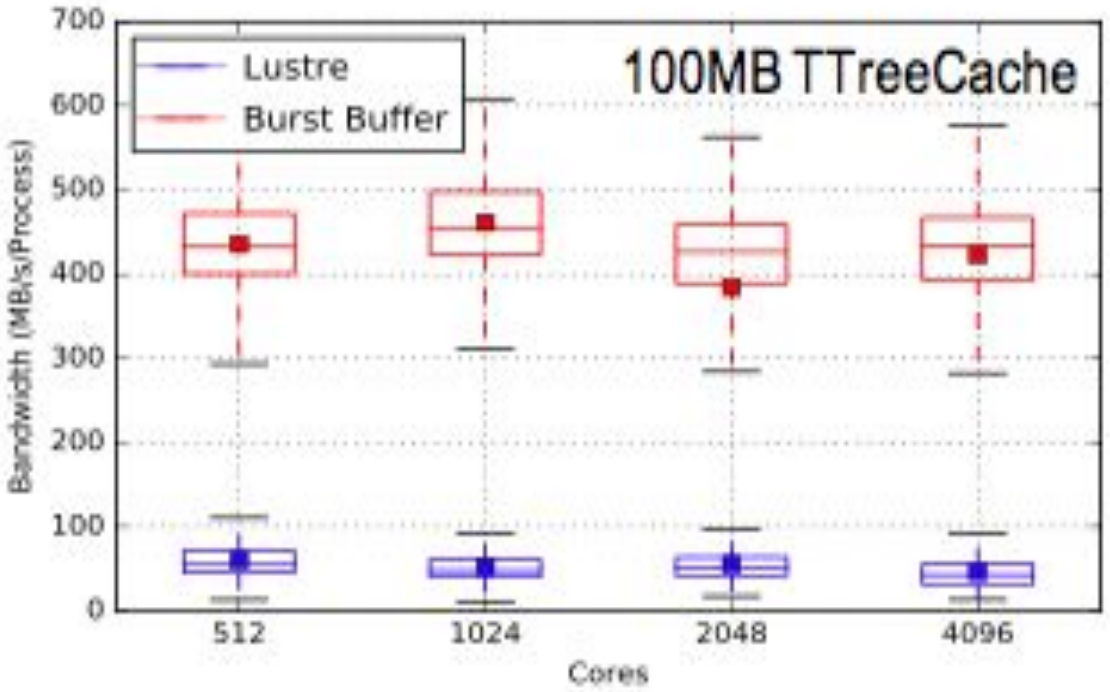
Success story: JGI

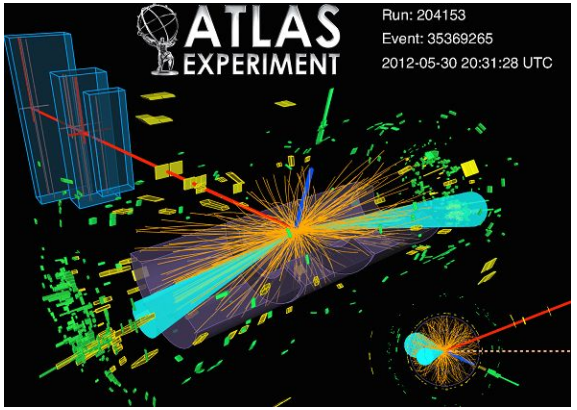
- **Metagenome assembly algorithm metaSPAdes**
  - dataset of DNA fragments
  - Lots of small, random reads.
  - I/O is a significant bottleneck!
  - Limits the size/complexity of microbial population that can be analysed.



- **Using the Burst Buffer gains factor of 2 in I/O performance out of the box, compared to heavily tuned Lustre.**

Success story: ATLAS

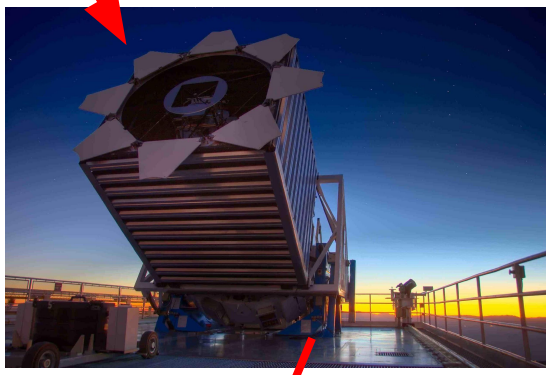


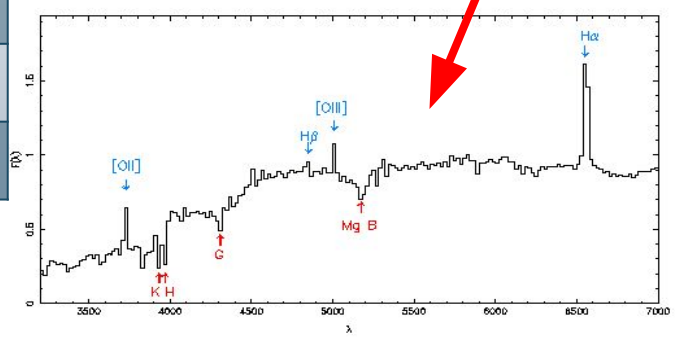


- **IOPS-heavy Data analysis**
  - Random reads from large numbers of data files
  - Used 50TB of BB space
  - ~9x faster I/O compared to Scratch.

Challenging IO use case: Astronomy data

- **Selecting subsets of galaxy spectra from a large dataset**
  - –Small, random memory accesses
  - –Typical web query for SDSS dataset
  - –Example code in the github repo!





Time taken to extract 1000 random spectra	From one hdf5 file	From individual fits files
From Lustre	44.1s	270s
From BB	1.3s	69s
Speedup:	**33x**	**4x**

Detailed look at high IOPs: SQLite

- A library which implements an SQL database engine
- No separate server process like there is in other database engines, e.g. MySQL, PostgreSQL, Oracle
- Database is stored in a single cross-platform file
- Installed on many supercomputers
- *"SQLite does not compete with client/server databases. SQLite competes with fopen()"* (https://sqlite.org/whentouse.html)

SQLite benchmark

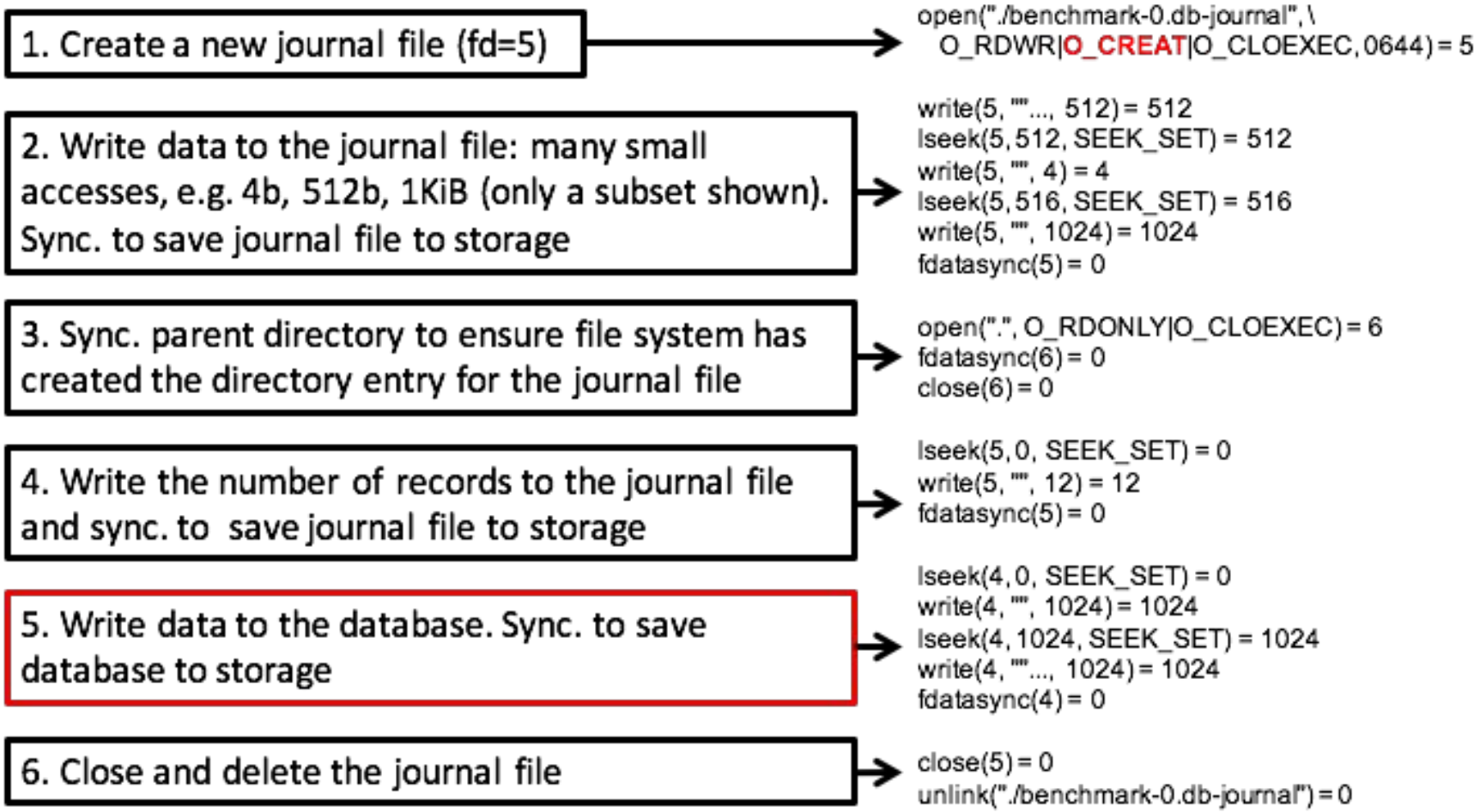
- **Inserts 2500 records into an SQLite database**
- **Written in C and optionally parallelized with MPI**
  - In parallel runs each MPI rank writes 2500 records to its own uniquely named database file

```
INSERT INTO 'pts1' ('I', 'DT', 'F1', 'F2')
VALUES ('1', CURRENT_TIMESTAMP, '6758',
'9844343722998287');
```

- **Anatomy of insert transaction:**
  - Dozens of I/O system calls are required for each SQLite transaction

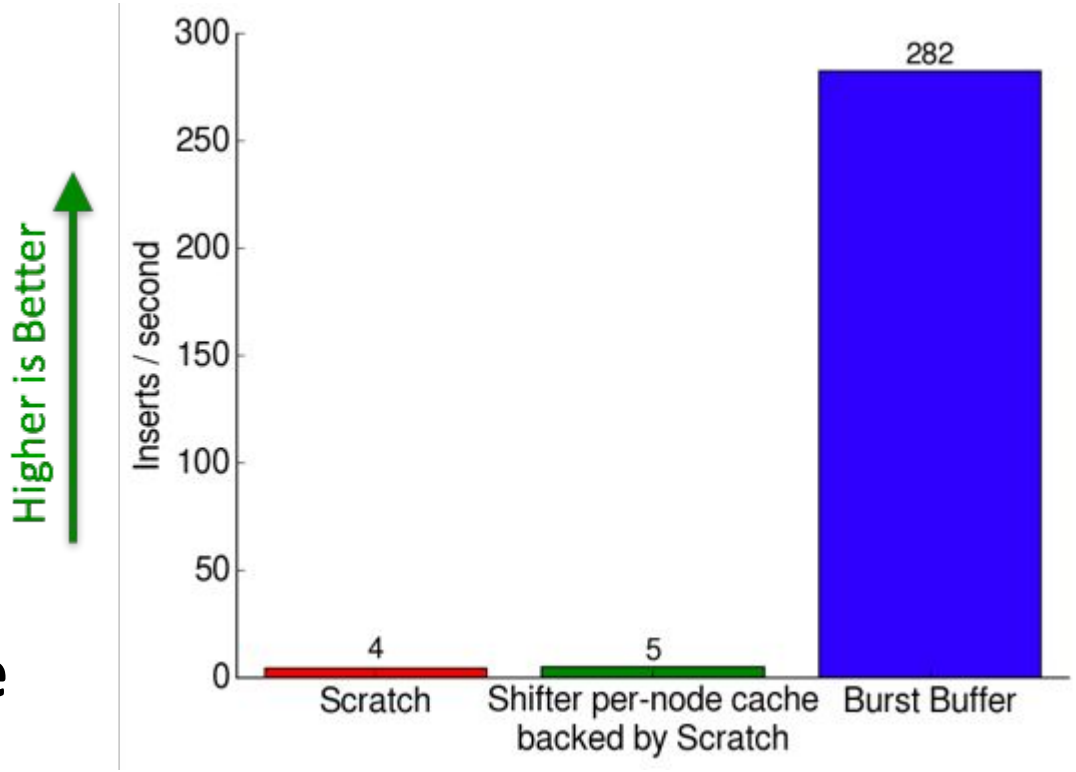
System call	Count
fdatasync	4
read	2
write	10
lseek	12
fcntl	9
open	2
close	2
unlink	1
fstat	5
stat	2
access	2

Many I/O ops for 1 DB insert!



~50x faster on the BB!

- Benchmark run with 1 MPI rank
- Scratch configuration uses 1 OST
- Burst Buffer configuration uses 1 granule of storage



Frequent synchs perform badly on Lustre

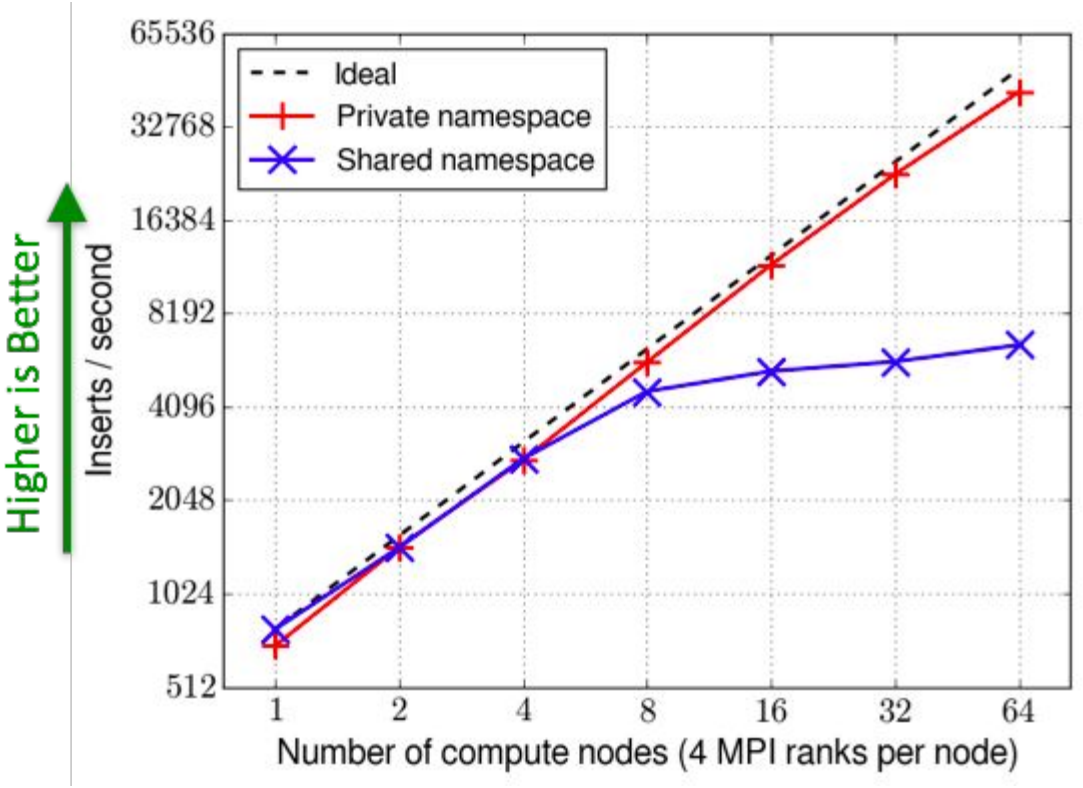
- 4 synchs for each of 2500 inserts: 98% of wall time!
- 1 synchronization every 2.5 writes gives no opportunity for the kernel to buffer the writes

#		[time]	[count]	<%wall>
#	**fdatasync**	**577.59**	**10004**	**97.83**
#	unlink	2.39	2511	0.41
#	**write**	**2.09**	**25036**	**0.35**
#	__fxstat64	1.33	10016	0.22
#	open64	1.32	5004	0.22
#	close	1.29	5004	0.22
#	__xstat64	0.93	10008	0.16
#	read	0.06	5003	0.01
#	lseek64	0.02	30038	0.00
#	__lxstat64	0.00	1	0.00
#	fflush	0.00	1	0.00

- ***The data transfer is limited by the latency of spinning disk***

MD performance scales well in private mode

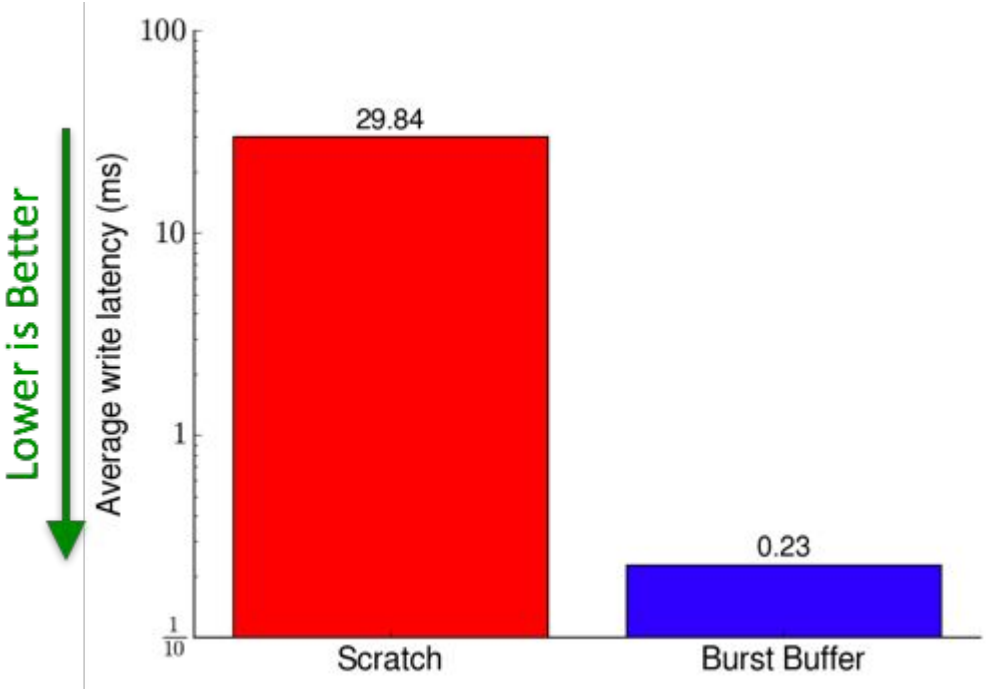
- **Private mode enables scalable metadata performance as we add compute nodes**
  - 1 metadata server per compute node



(All runs use 64 BB granules)

MD in IOR benchmark

- **Single-stream IOR with a data synchronization after every POSIX write (-Y flag)**
- **Average write latency < 1 millisecond on BB**
  - two orders of magnitude faster than disk!



Summary

- **NERSC has the first Burst Buffer for open science in the USA**
- **Users are able to take advantage of SSD performance**
  - Some tuning may be required to maximise performance
- **Many bugs now worked through**
  - But care is needed when using this new technology!
- **User experience today is generally good**
- **Performance for metadata-intensive operations is particularly excellent**

Resources

- **NERSC Burst Buffer Web Pages**

http://www.nersc.gov/users/computational-systems/cori/burst-buffer/

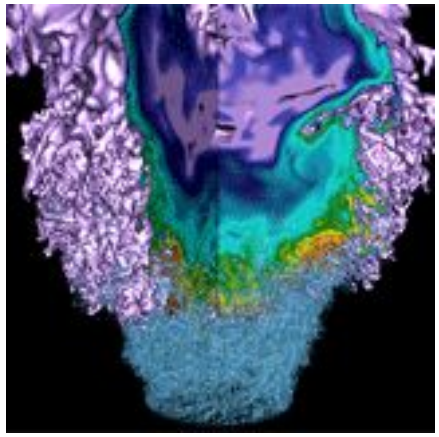
- **Example batch scripts**

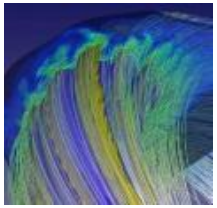
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/

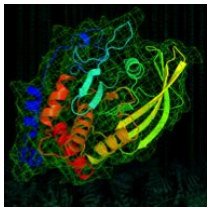
- **FAQ**

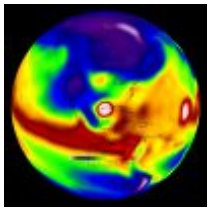
https://www.nersc.gov/users/computational-systems/cori/burst-buffer/burst-buffer-faq/

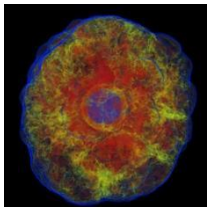
Extra slides

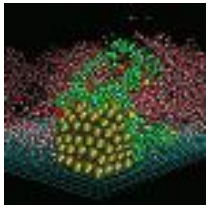






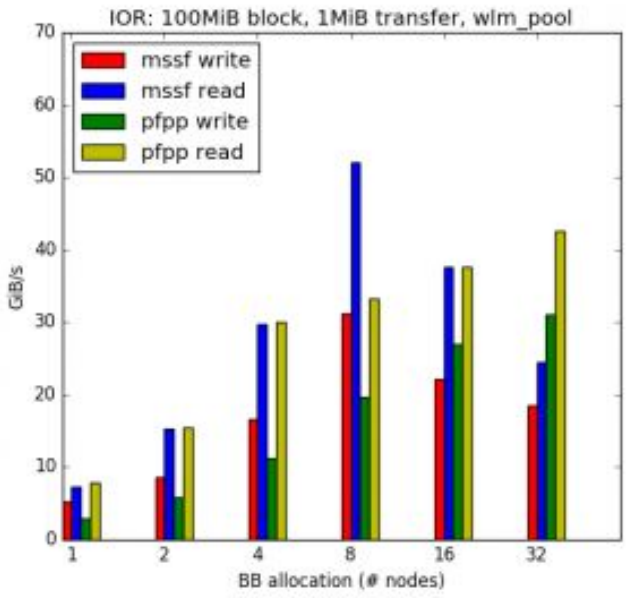


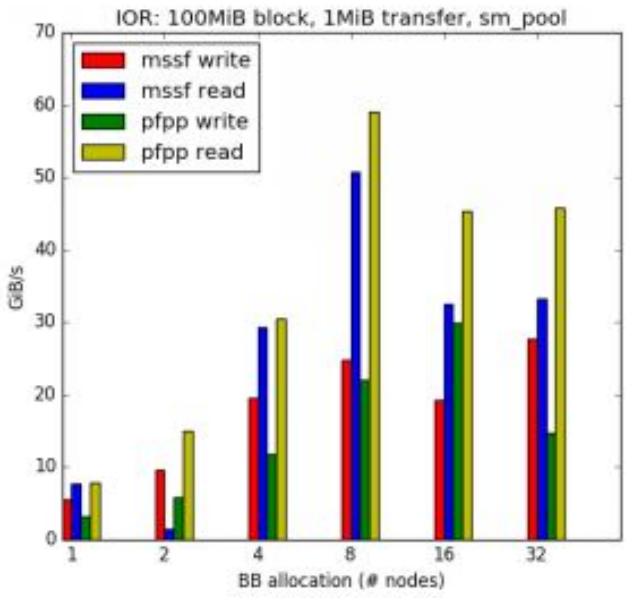




Performance tips

- **Stripe your files across multiple BB servers**
  - To obtain good scaling, need to drive IO with sufficient compute - scale up # BB nodes with # compute nodes



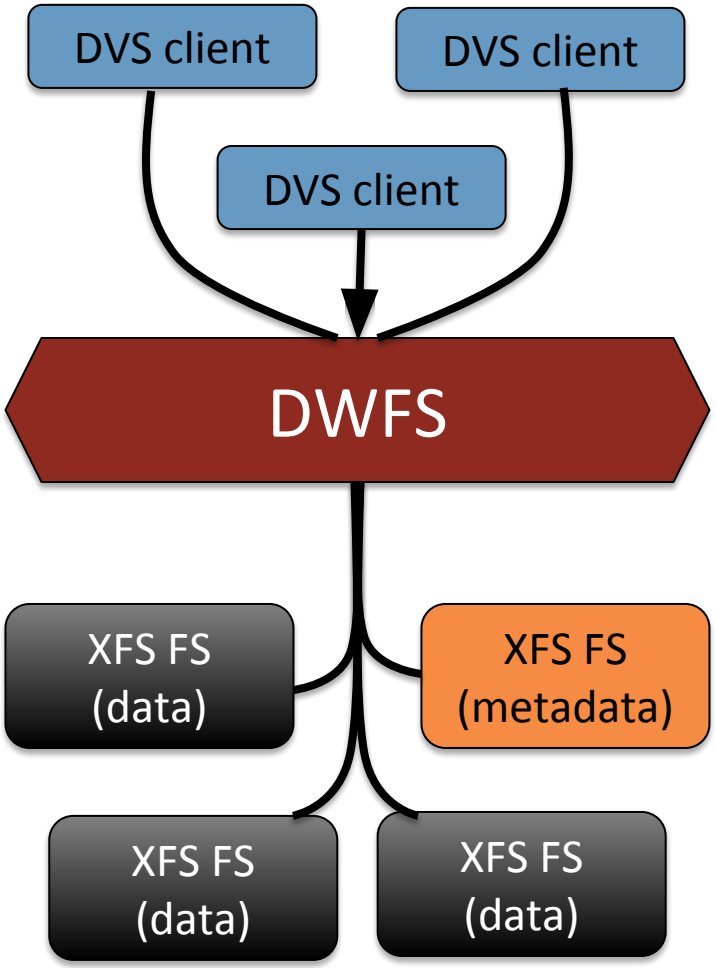


SSD write protection

- **SSDs support a set amount of write activity before they wear out**
- **Runaway application processes may write an excessive amount of data, and therefore, "destroy" the SSDs**
- **Three write protection policies**
  - –Maximum number of bytes written in a period of time
  - –Maximum size of a file in a namespace
  - –Maximum number of files allowed to be created in a namespace
- **Log, error, log and error**
  - –`EROFS` (write window exceeded)
  - –`EMFILE` (maximum files created exceeded)
  - –`EFBIG` (maximum file size exceeded)

DataWarp File System (DWFS)

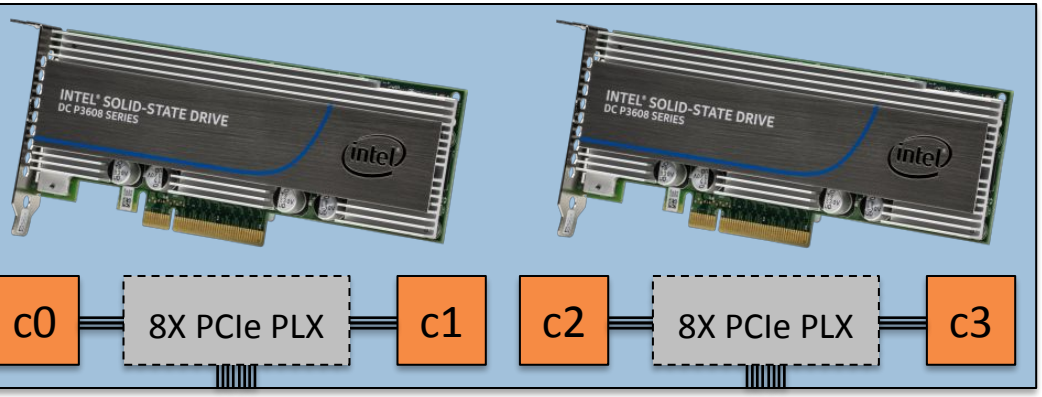
- **File system built on Wrapfs that glues together**
  - Cray DVS for client-server RPCs
  - many XFS file systems for data (called "fragments")
  - one XFS file system for metadata
- **Conceptually very simple**
  - No DLM
    - rely on server-side VFS file locking
    - no client-side page cache (yet)
  - Data placement determined by deterministic hash of inode, offset
  - Stubbed XFS file system encodes most file metadata



DWFS Storage Substrate

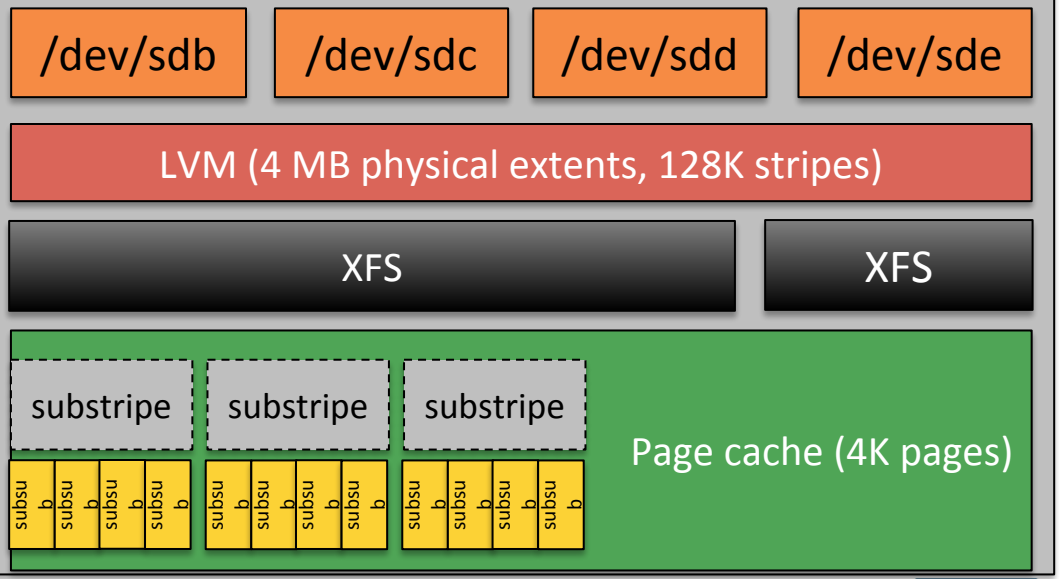
Physical Node

- 1x Sandy Bridge E5, 8-core
- 64 GB DDR3
- 2x Intel P3608 (3.2 TB ea.)
- 4x Intel P3600 controllers

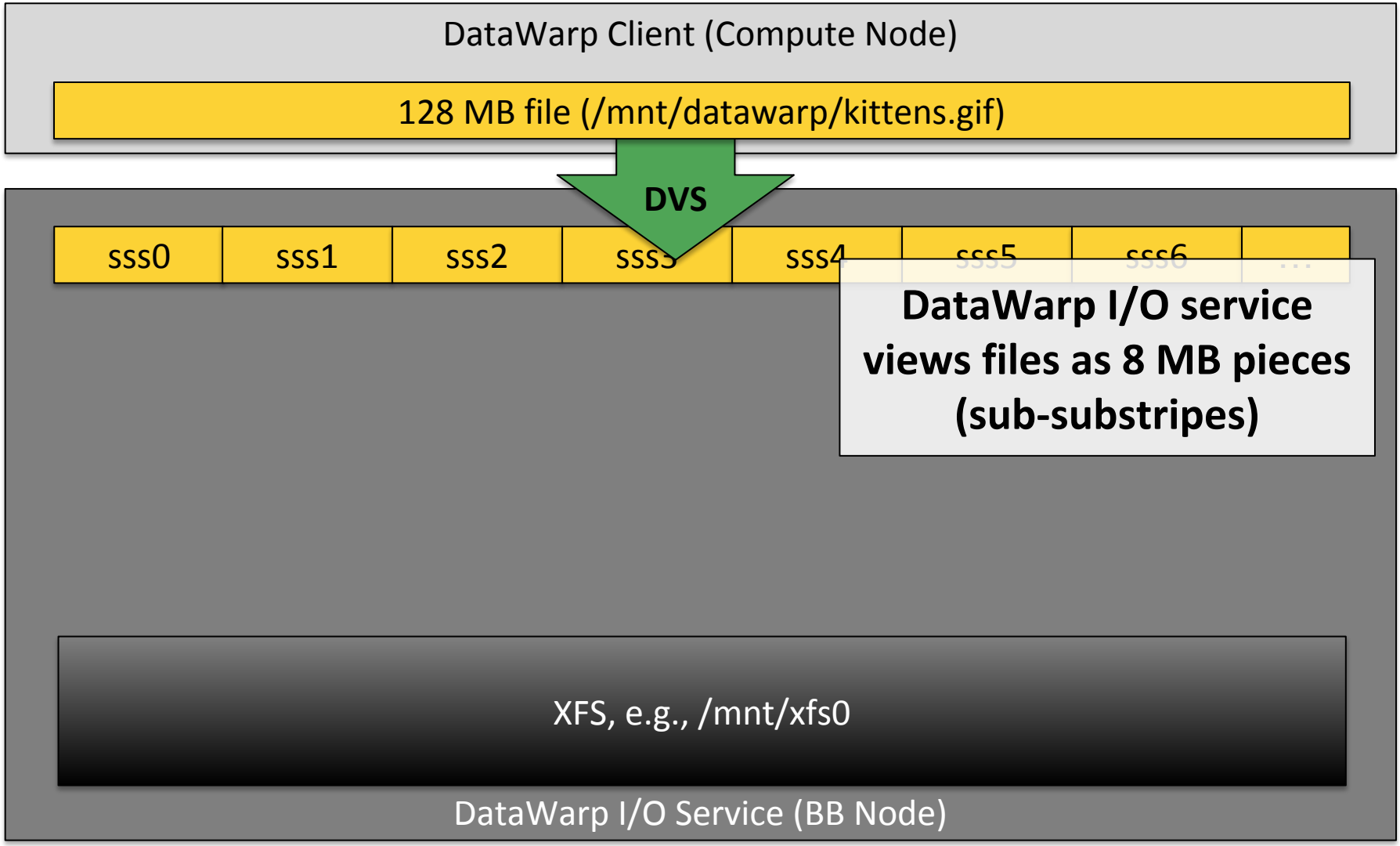


Linux OS

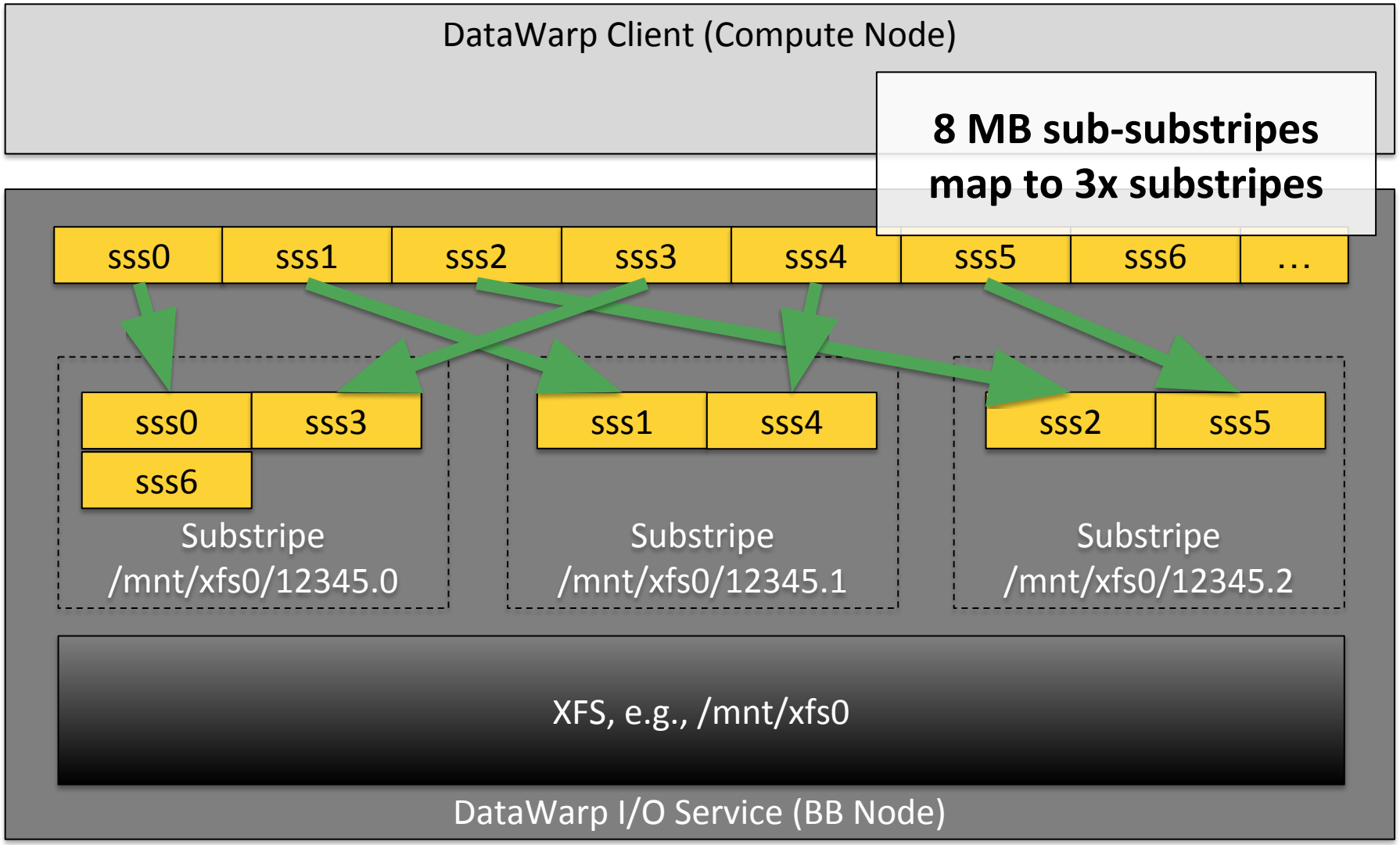
- Logically four block devices
- LVM aggregates block devices
- Linux vol group and XFS fs
- 3 substripes per file per BB node
- 8 MB sub-substripes in substripe



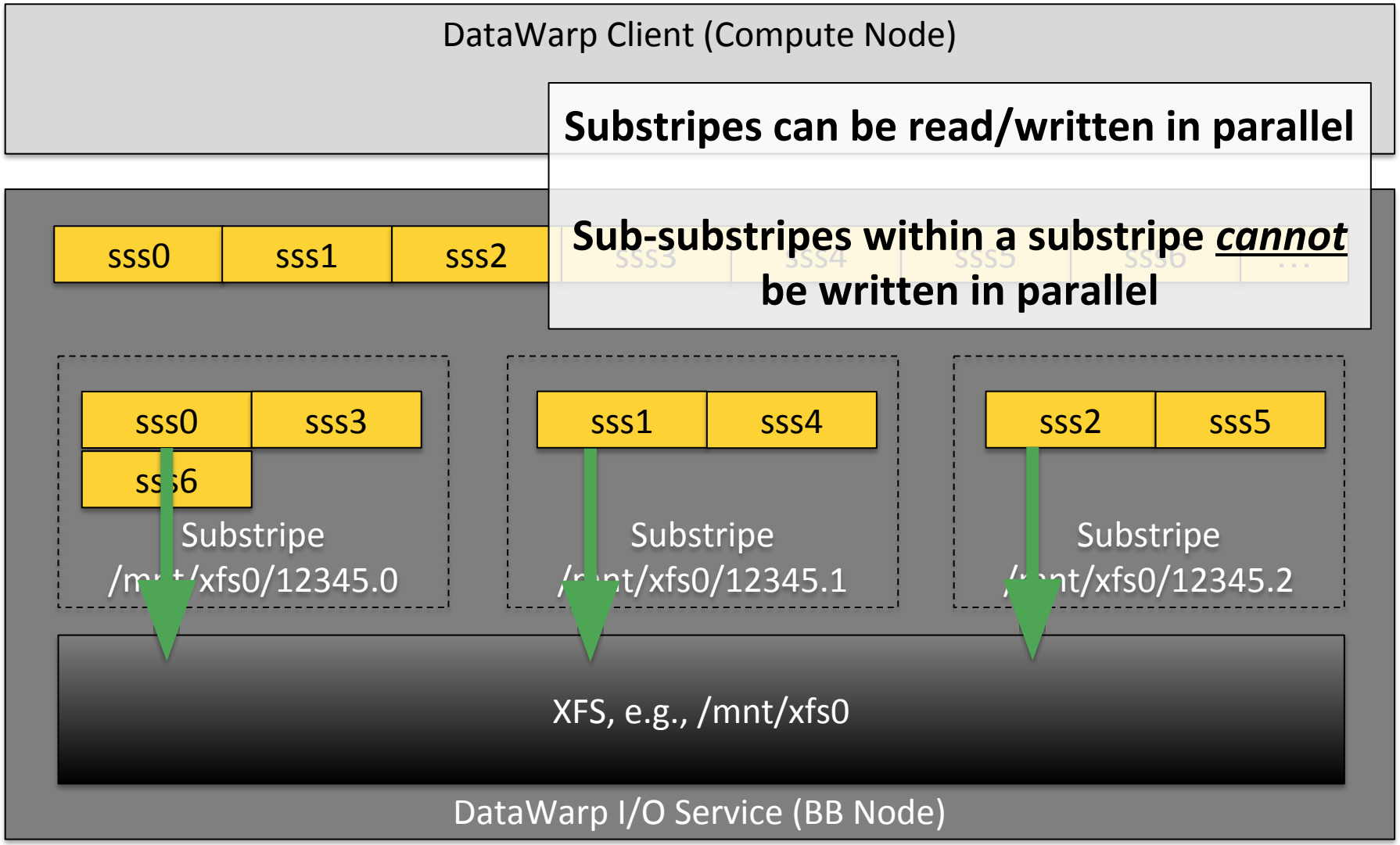
Data Layout: Simple Case (1 BB node)



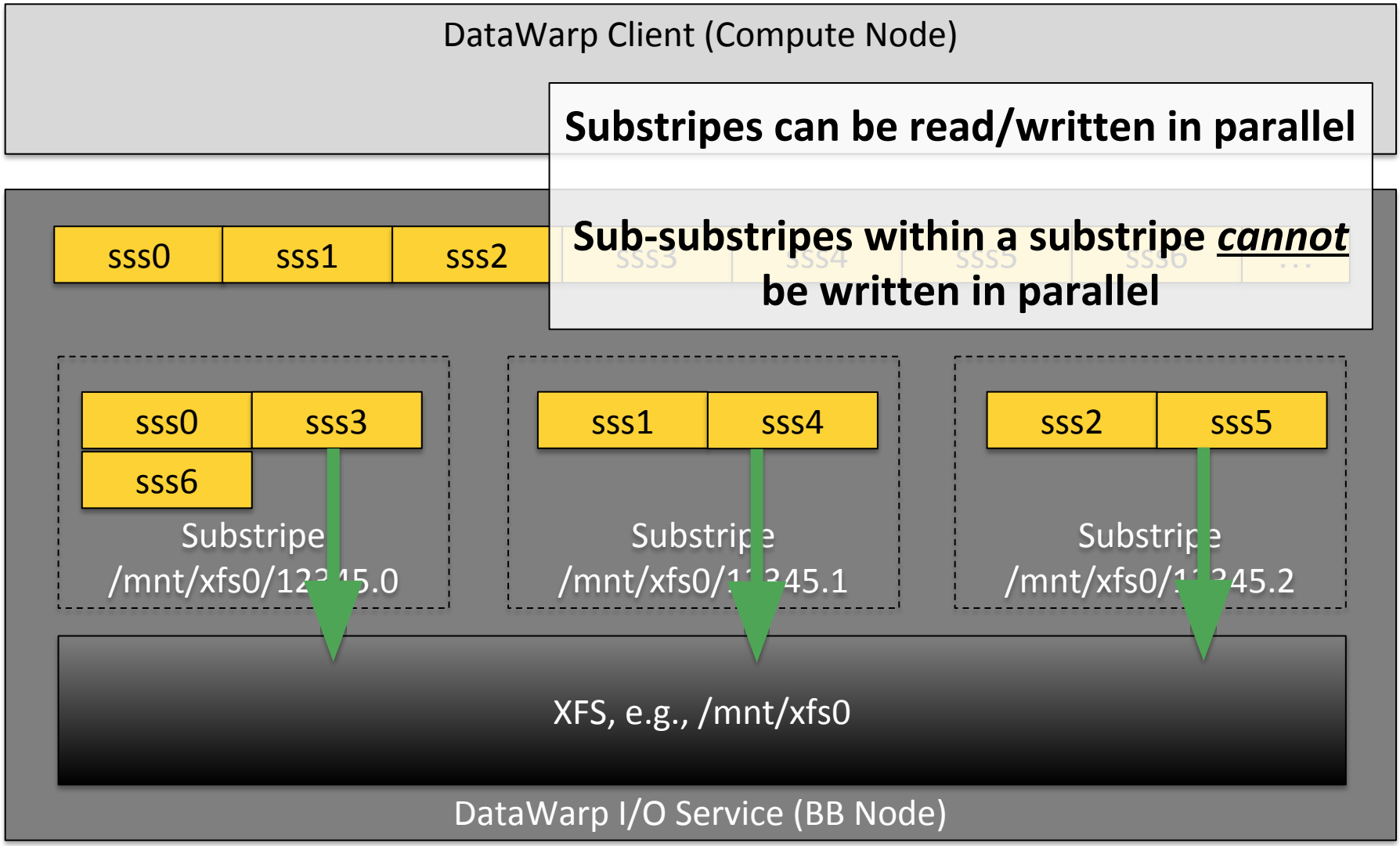
Data Layout: Simple Case (1 BB node)



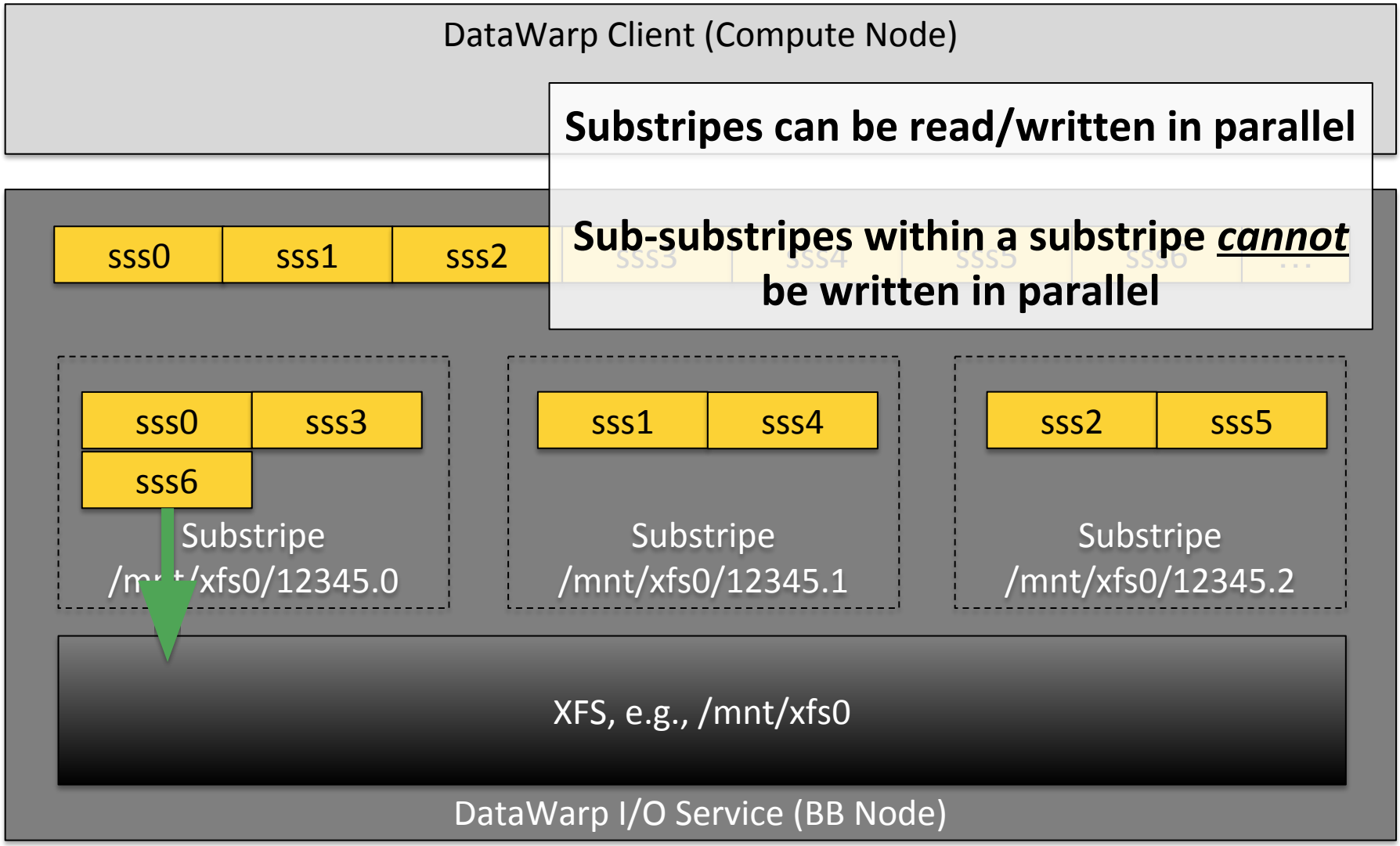
Data Layout: Simple Case (1 BB node)



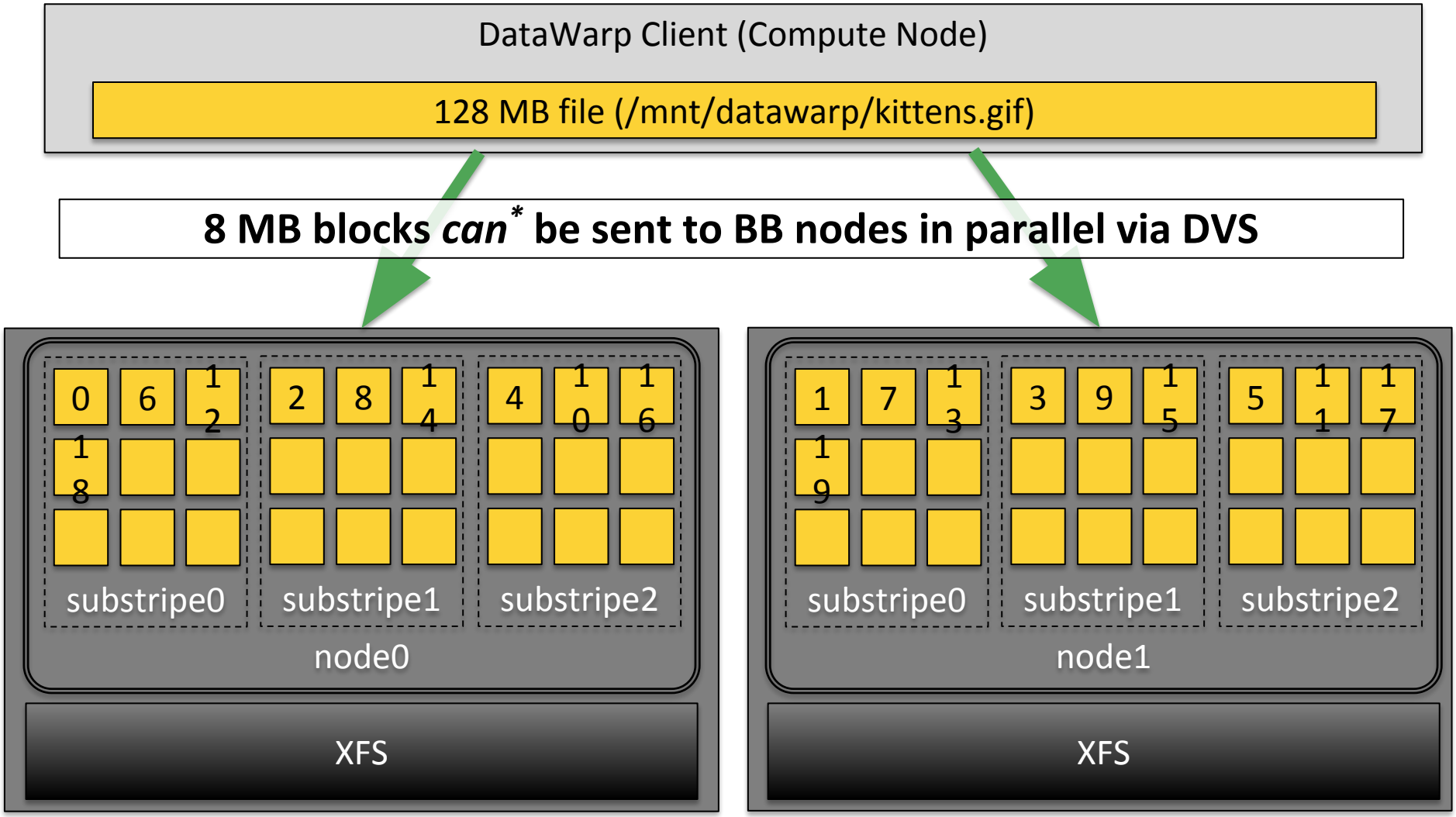
Data Layout: Simple Case (1 BB node)



Data Layout: Simple Case (1 BB node)



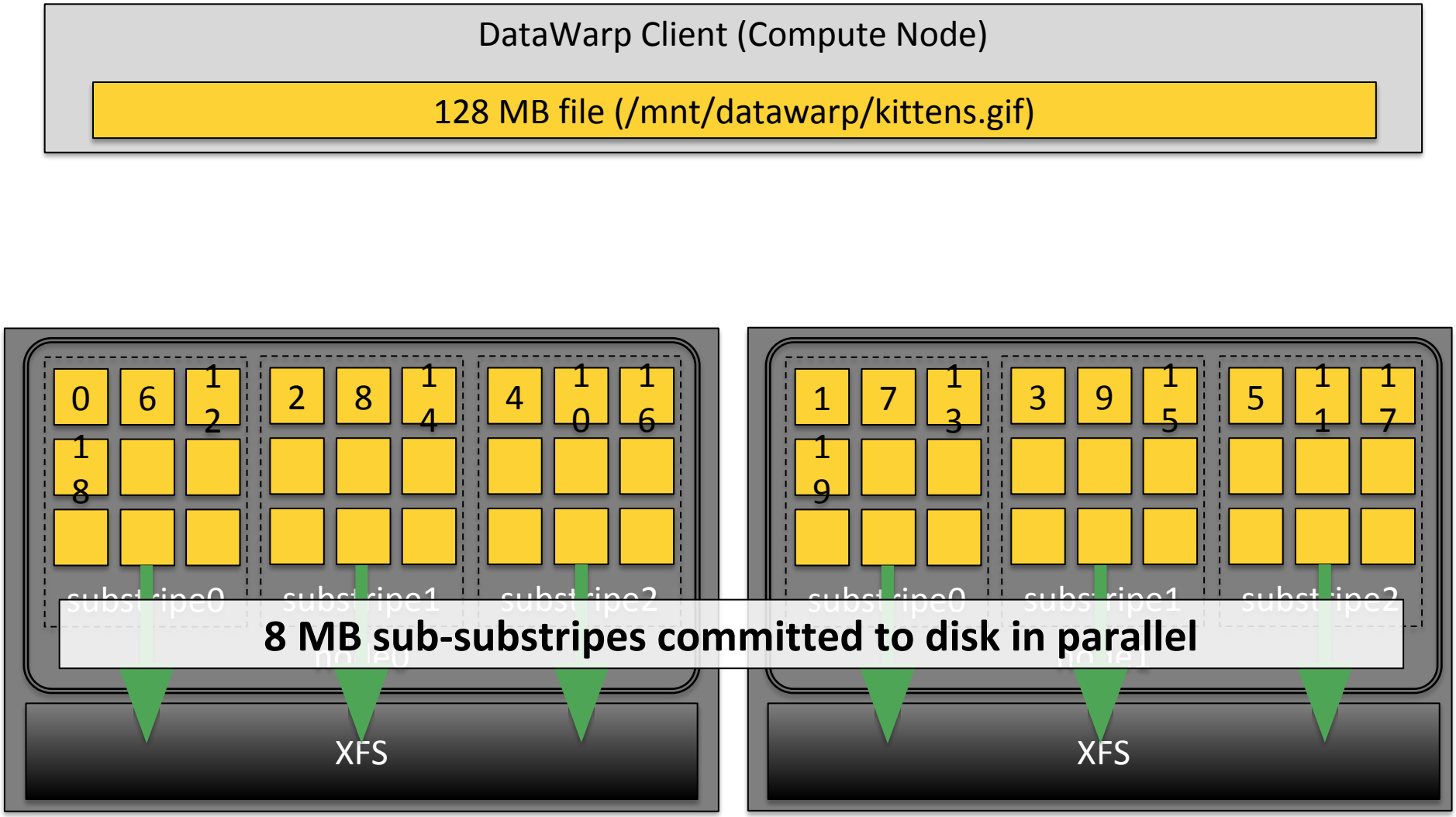
Data Layout: 2 BB nodes



* under certain conditions

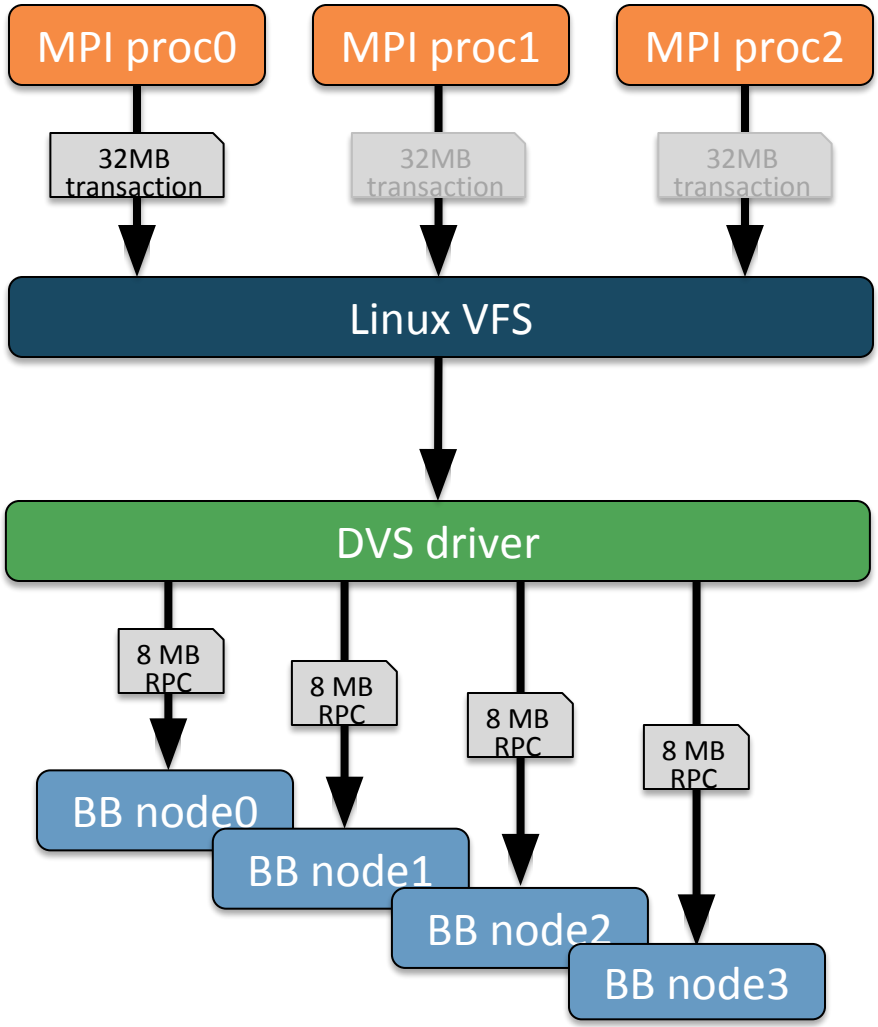
Slides from Glenn Lockwood, NERSC

Data Layout: 2 BB nodes

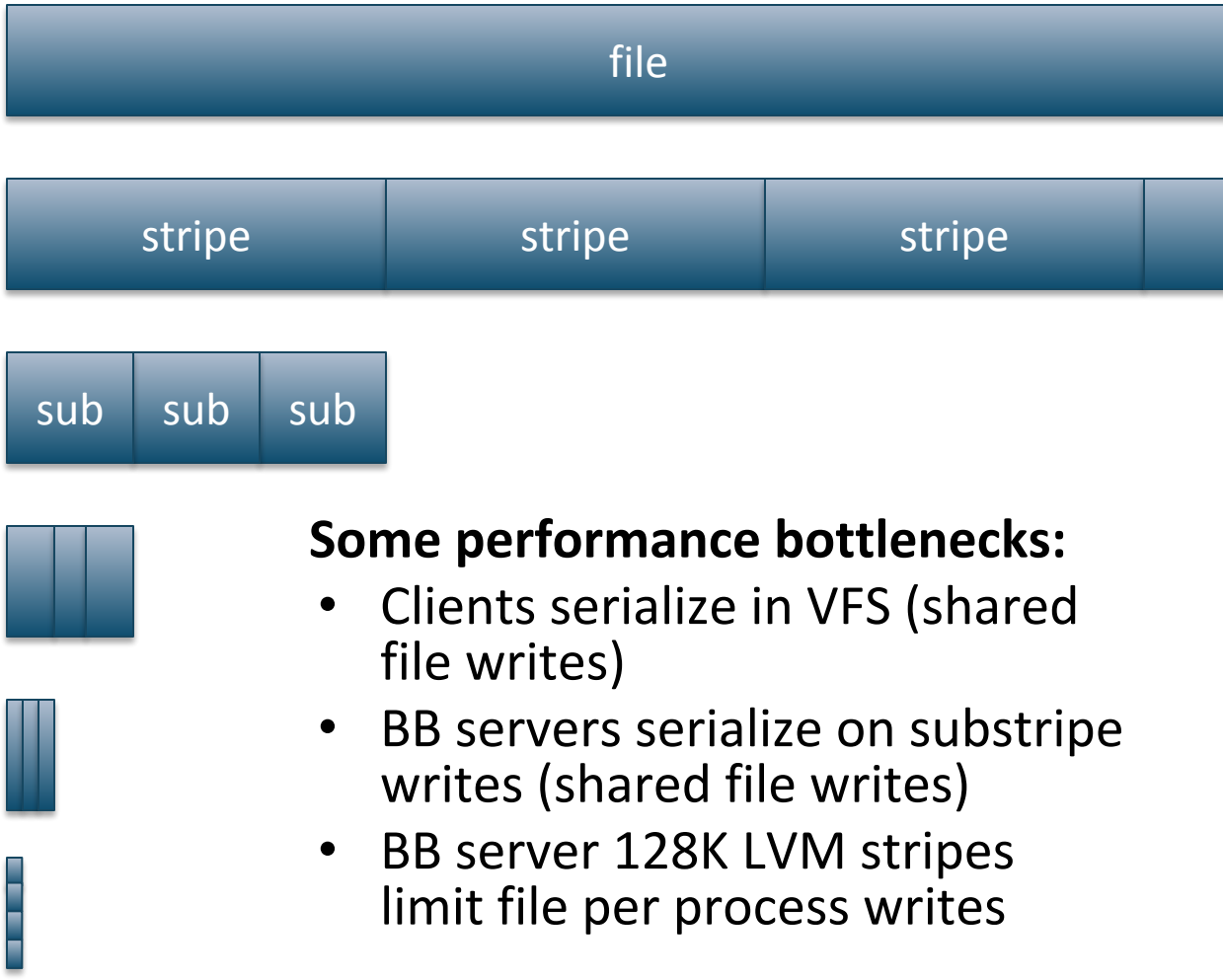


DWFS Data Path - Client

- No page cache for write-back
- Shared-file writes are serialized by VFS
- DVS can parallelize very large transactions



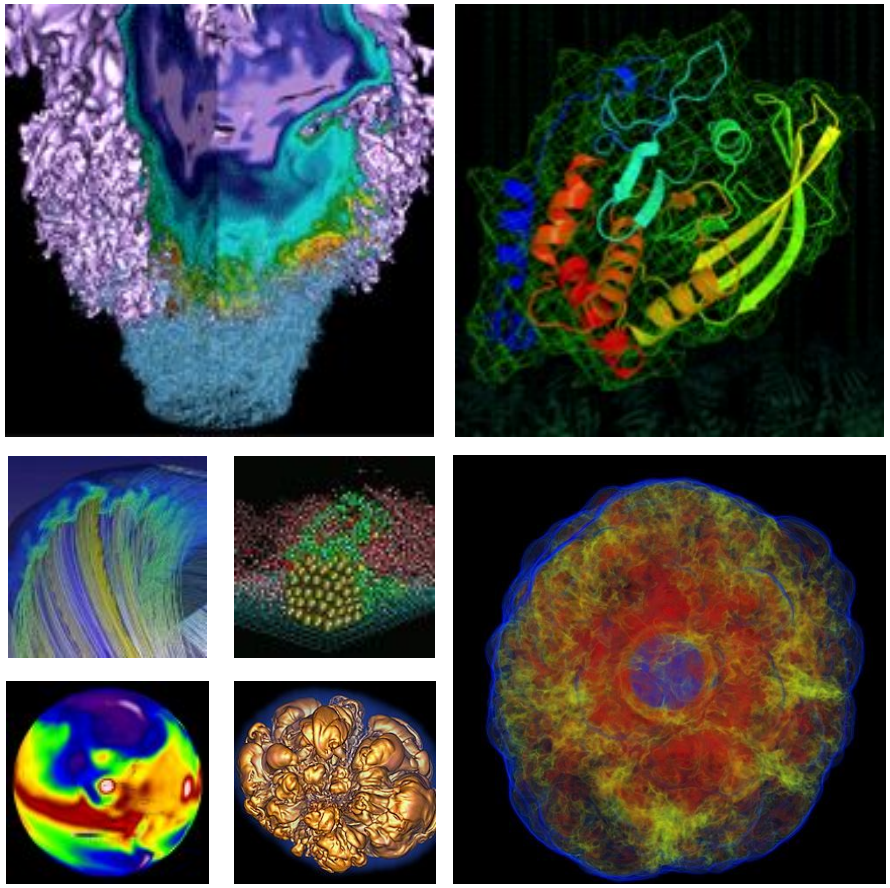
Hierarchical Parallelism



Some performance bottlenecks:

- Clients serialize in VFS (shared file writes)
- BB servers serialize on substripe writes (shared file writes)
- BB server 128K LVM stripes limit file per process writes

Getting your hands on Cori's Burst Buffer



Debbie Bard

Data and Analytics Services
NERSC

SC17 Tutorial

Scratch allocation

```
#!/bin/bash
#SBATCH -p regular -N 10 -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - Actual distribution across BB Nodes is in units of (configurable) granularity (currently 80 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 13 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Scratch allocation

```
#!/bin/bash
#SBATCH -p regular -N 10 -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - Actual distribution across BB Nodes is in units of (configurable) granularity (currently 80 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 13 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Scratch allocation

```
#!/bin/bash
#SBATCH -p regular -N 10 -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - –Actual distribution across BB Nodes is in units of (configurable) granularity (currently 80 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 13 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Scratch allocation

```
#!/bin/bash
#SBATCH -p regular -N 10 -t 00:10:00
#DW jobdw capacity=1000GB access_mode=striped type=scratch
#DW stage_in source=/lustre/inputs destination=$DW_JOB_STRIPED/inputs \
type=directory
#DW stage_in source=/lustre/file.dat destination=$DW_JOB_STRIPED/ type=file
#DW stage_out source=$DW_JOB_STRIPED/outputs destination=/lustre/outputs \
type=directory
srun my.x --indir=$DW_JOB_STRIPED/inputs --infile=$DW_JOB_STRIPED/file.dat \
--outdir=$DW_JOB_STRIPED/outputs
```

- **'type=scratch' – duration just for compute job (i.e. not 'persistent')**
- **'access_mode=striped' – visible to all compute nodes (i.e. not 'private') and striped across multiple BB nodes**
  - Actual distribution across BB Nodes is in units of (configurable) granularity (currently 80 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 13 BB nodes)
- **Data 'stage_in' before job start and 'stage_out' after**

Step 1: log onto Cori

- **We have temporary user accounts for NERSC (that will expire at the end of the day) and a reservation of KNL nodes on Cori.**
- **Using your training account (or your own NERSC account): ssh username@cori.nersc.gov**
  - Note that if you use your own NERSC account you won't be part of our compute reservation but you can still submit jobs that run on the Burst Buffer.

```
                    complete.

Cori:           05/16/17 17:00-05/17/17 17:00 PDT, Dedicated runs.
                 Cori HPL Dedicated run. Logins and job submission available.
                 Jobs will remained queued until the dedicated run is complete.

---------------------------------- Past Outages ----------------------------------

Cori:           05/04/17 8:00-16:30 PDT, Scheduled maintenance.
                 Cori will be undergoing planned maintenance during this time
                 period.

Cori:           05/04/17 17:22-22:15 PDT, System in degraded mode.
                 Cori is experiencing a possible hardware issue that is under
                 investigation. Logins and job submission are available but new
                 jobs will not start

For past outages, see: http://my.nersc.gov/outagelog-cs.php

--------------------------------------------------------------------------------

train61@cori07:~>
```

Copy over the example scripts and test data

- **Pull down the example scripts and the test data file onto your scratch space**
  - We're using scratch (i.e. Lustre) because it's currently the only filesystem the burst buffer can access on Cori.

```
cd $SCRATCH
git clone
https://github.com/KAUST-KSL/SC17_BB_Tutorial.git
cd SC17_BB_Tutorial/intro/
bash copy.sh
```

Run the first simple script!

- **Look at the example script "scratch.sh":**

```
#!/bin/bash

#### Which partition? Use "regular" for the reservation
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:1:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_scratch"

#### Set the output file name
#SBATCH -o "job_scratch.log"

#### Request a 200GB scratch allocation, striped over BB nodes, in the default pool
(which has 82GiB granularity, so this gives you grains on 3 BB nodes)
#DW jobdw capacity=200GB access_mode=striped type=scratch pool=wlm_pool


#### print the mount point of your BB allocation on the compute nodes
echo "*** DW path is:"
echo $DW_JOB_STRIPED

#### Write a file on the BB
echo "*** saying hello...."
echo "Hello! I'm on the Burst Buffer!" > $DW_JOB_STRIPED/hello.txt

echo "*** ls -larth $DW_JOB_STRIPED/"
ls -larth $DW_JOB_STRIPED/

echo "*** cat $DW_JOB_STRIPED/hello.txt"
cat $DW_JOB_STRIPED/hello.txt
```

Run the first simple script!

- **Submit the job using “sbatch scratch.sh”**
- **User “sqs” and “squeue” to view the status of your job**

“sqs”: information about your job status

“squeue”: info about all jobs. “-l” option gives any BB error messages.

```
train61@cori07:~> cd  $SCRATCH/CUG2017/examples
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> ls
IOR                  destroy_persistent.sh  scratch.sh   stage_out.sh
create_persistent.sh  run_IOR.sh            stage_in.sh  use_persistent.sh
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples>
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples>
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> sbatch  scratch.sh
Submitted batch job 4883133
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> sqs
JOBID             ST   REASON       USER         NAME         NODES        USED         REQ
UESTED    SUBMIT                PARTITION    RANK_P       RANK_BF
4883133           PD   BurstBuffe   train61      scratch      1            0:00         5:0
0         2017-05-08T10:19:56   debug        11811        N/A
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> sqs
JOBID             ST   REASON       USER         NAME         NODES        USED         REQUESTED    SUBMIT               PARTITION    RANK_P       RANK_BF
4883133           PD   BurstBuffe   train61      scratch      1            0:00         5:00         2017-05-08T10:19:56  debug        11811        N/A
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> squeue -u train61
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           4883133     debug  scratch  train61 PD       0:00      1 (BurstBufferResources)
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> squeue -u train61 -l
Mon May  8 10:20:53 2017
             JOBID PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES NODELIST(REASON)
           4883133     debug  scratch  train61  PENDING       0:00      5:00      1 (BurstBufferResources)
```

View Burst Buffer status

- **User "scontrol show burst" to look at what the Burst Buffer is doing right now**

```
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples> scontrol show burst
Name=cray DefaultPool=wlm_pool Granularity=82496M TotalSpace=1192325G FreeSpace=893277G UsedSpace=272814272M
  AltPoolName[0]=dev_pool Granularity=20624M TotalSpace=47693G FreeSpace=47693G UsedSpace=0
  AltPoolName[1]=sm_pool Granularity=20624M TotalSpace=476930G FreeSpace=476930G UsedSpace=0
  Flags=EnablePersistent,TeardownFailure
  StageInTimeout=86400 StageOutTimeout=86400 ValidateTimeout=5 OtherTimeout=300
  GetSysState=/opt/cray/dw_wlm/default/bin/dw_wlm_cli
  Allocated Buffers:
    JobID=4829054_2(4829056) CreateTime=2017-05-04T17:01:02 Pool=wlm_pool Size=82496M State=staged-in UserID=aclum(16603)
    JobID=4883133 CreateTime=2017-05-08T10:21:51 Pool=wlm_pool Size=247488M State=staged-in UserID=train61(47535)
    JobID=4831969 CreateTime=2017-05-05T09:37:25 Pool=wlm_pool Size=8414592M State=staged-in UserID=jyc(57038)
    JobID=4831963 CreateTime=2017-05-05T09:37:25 Pool=wlm_pool Size=3299840M State=staged-in UserID=jyc(57038)
    Name=octotiger CreateTime=2017-03-29T01:18:15 Pool=(null) Size=10312G State=allocated UserID=sithhell(61642)
    JobID=4829283 CreateTime=2017-05-07T19:21:56 Pool=wlm_pool Size=21036480M State=staged-in UserID=bharti(70083)
    Name=PDC_BB CreateTime=2017-04-24T17:14:51 Pool=(null) Size=164992M State=allocated UserID=bharti(70083)
    Name=my_persistent_reservation CreateTime=2017-05-07T12:18:33 Pool=wlm_pool Size=329984M State=allocated UserID=djbard(61692)
    Name=celestebb CreateTime=2017-04-09T11:25:05 Pool=(null) Size=114339456M State=allocated UserID=kpamnany(69499)
    Name=celestebb2 CreateTime=2017-03-24T20:14:11 Pool=(null) Size=114339456M State=allocated UserID=kpamnany(69499)
  Per User Buffer Use:
    UserID=aclum(16603) Used=82496M
    UserID=train61(47535) Used=247488M
    UserID=jyc(57038) Used=11714432M
    UserID=sithhell(61642) Used=10312G
    UserID=bharti(70083) Used=21201472M
    UserID=djbard(61692) Used=329984M
    UserID=kpamnany(69499) Used=228678912M
train61@cori07:/global/cscratch1/sd/train61/CUG2017/examples>
```

Stage in data to a scratch allocation

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:1:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_stage_in"

#### Set the output file name
#SBATCH -o "job_stage_in.log"

#### Request a 200GB scratch allocation, striped over BB nodes
#DW jobdw capacity=200GB access_mode=striped type=scratch pool=wlm_pool

#### Stage in a directory. Remember to change this directory to your training
# account scratch directory! You can also stage_in a file by specifying "type=
file".
#DW stage_in source=/global/cscratch1/sd/djbard/SC17_BB_Tutorial/intro/data/
# destination=$DW_JOB_STRIPED/data type=directory



#### print the mount point of your BB allocation on the compute nodes
echo "*** The path to my BB allocation is:"
echo $DW_JOB_STRIPED

#### Check what's been staged in to your allocation
echo "*** what's on my scratch allocation?"

echo "ls -lrtha  $DW_JOB_STRIPED/ "
ls -lrtha  $DW_JOB_STRIPED/

echo "ls -lrtha  $DW_JOB_STRIPED/data/  "
ls -lrtha  $DW_JOB_STRIPED/data/
```

- Look at “stage_in.sh”.
- Edit it to point to your OWN file/directory that you want to stage_in
- Submit the job
- Check the output log file - did the directory stage in as expected?

Stage out data from a scratch allocation

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:01:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_stage_out"

#### Set the output file name
#SBATCH -o "job_stage_out.log"

#### Request a 200GB scratch allocation, striped over BB nodes
#DW jobdw capacity=200GB access_mode=striped type=scratch pool=wlm_pool

#### Stage a file out of the BB onto scratch.
#DW stage_out destination=/global/cscratch1/sd/djbard/SC17_BB_Tutorial/intro/data/hello.txt source=$DW_JOB_STRIPED/hello.txt type=file



#### print the mount point of your BB allocation on the compute nodes
echo "*** DW path is:"
echo $DW_JOB_STRIPED

#### Write a file on the BB
echo "*** saying hello...."
echo "Hello! I'm on the Burst Buffer!" > $DW_JOB_STRIPED/hello.txt
echo "*** ls -larth $DW_JOB_STRIPED/"
ls -larth $DW_JOB_STRIPED/
echo "*** cat $DW_JOB_STRIPED/hello.txt"
cat $DW_JOB_STRIPED/hello.txt
```

- Look at “stage_out.sh”.
- Edit it to point to your OWN scratch directory
- Submit the job
- Check your scratch directory - did “hello.txt” stage out as expected?

Persistent reservations

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job (posix permissions permitting)
  - `name=xyz`: Name of persistent instance to use

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB destroy_persistent name=myBBname
```

Use in another job

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#DW persistentdw name=myBBname
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Persistent reservations

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job (posix permissions permitting)
  - `name=xyz`: Name of persistent instance to use

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB destroy_persistent name=myBBname
```

Use in another job

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#DW persistentdw name=myBBname
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Persistent reservations

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job (posix permissions permitting)
  - `name=xyz`: Name of persistent instance to use

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB destroy_persistent name=myBBname
```

Use in another job

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#DW persistentdw name=myBBname
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Persistent reservations

- **Using a *persistent* DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job (posix permissions permitting)
  - `name=xyz`: Name of persistent instance to use

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB create_persistent name=myBBname capacity=10GB access=striped type=scratch
```

Delete

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#BB destroy_persistent name=myBBname
```

Use in another job

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -t 00:05:00
#DW persistentdw name=myBBname
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```

Create a persistent reservation

- Look at “create_persistent .sh”
- Edit it to rename the persistent reservation!
- Submit it, then use “scontrol show burst” to check it’s been created.

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job? This doesn't need to be long as we're not actually executing anything on the compute node.
#SBATCH -t 00:1:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_create_persistent"

#### Set the output file name
#SBATCH -o "job_create_persistent.log"


#### Create a persistent reservation (PR). Choose what size and name you want to give it. Note that you MUST change this name! Every PR name needs to be unique - if there already exists a PR of this name the job will fail.
#BB create_persistent name=my_persistent_reservation capacity=300GB access_mode=striped type=scratch
```

Use a persistent reservation

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:5:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_use_persistent"

#### Set the output file name
#SBATCH -o "job_use_persistent.log"


#### Specify which persistent reservation you will access. Remember to use
eservation name!
#DW persistentdw name=my_persistent_reservation


#### Stage in some data to the persistent reservation. Remember to change this to point
to your data!

#### Stage in a directory. Remember to change this directory to your training account sc
ratch directory! You can also stage_in a file by specifying "type=file".
#DW stage_in source=/global/cscratch1/sd/djbard/SC17_BB_Tutorial/intro/data  destination
=$DW_JPERSISTENT_STRIPED_my_persistent_reservation/data  type=directory




#### print the mount point of your BB allocation on the compute nodes
echo "*** The path to my BB allocation is:"
echo $DW_PERSISTENT_STRIPED_my_persistent_reservation

#### Check what's been staged in to your allocation
echo "*** What's on my persistent reservation?:"

echo "*** ls -lrt  $DW_PERSISTENT_STRIPED_my_persistent_reservation "
ls -lrt  $DW_PERSISTENT_STRIPED_my_persistent_reservation


echo "*** ls -lrt  $DW_PERSISTENT_STRIPED_my_persistent_reservation "
ls -lrt  $DW_PERSISTENT_STRIPED_my_persistent_reservation/data
```

- **Look at “use_persistent.sh”**
- **Change:**
  - The PR name,
  - The stage_in path
  - The variable $DW_PERSISTENT_STRIPED_name

Destroy a persistent reservation

- **Look at "destroy_persistent.sh"**
- **Change the name to your own PR name.**
- **Submit it, then use "scontrol show burst" to check it's been destroyed.**
  - If you get the name wrong, you'll get NO error messages.

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
#SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:1:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_destroy_persistent"

#### Set the output file name
#SBATCH -o "job_destroy_persistent.log"


#### Destroy the persistent reservation. All data on the reservation will be lost.
# Remember to use the correct reservation name!
#BB destroy_persistent name=my_persistent_reservation
```

Using the Burst Buffer interactively

- **Look at “interactive_scratch.conf”:**

```
#DW jobdw capacity=100GB access_mode=striped type=scratch
```

- **Request an interactive session using “salloc”:**

```
salloc -N 1 -t 5 -C haswell --reservation="sc17_bb"
--bbf=”interactive_scratch.conf”
```

```
salloc -N 1 -t 5 -C knl --qos=interactive
--bbf=”interactive_scratch.conf”
```

- **Play around with the BB interactively!**
  - Try running IOR using the executable in your examples directory:

```
srun -n 8 ./IOR -a MPIIO -g -t 10MiB -b 100MiB -o
$DW_JOB_STRIPED/IOR_file
```

- **Also “interactive_persistent.conf” for a PR.**

Using the Burst Buffer interactively

```
djbard@nid00578:~/BB/CUG17> ./IOR -a MPIIO -g -t 20MiB -b 100MiB -o $DW_JOB_STRIPED/IOR_file
IOR-3.0.1: MPI Coordinated Test of Parallel I/O

Began: Mon May  8 10:03:07 2017
Command line used: ./IOR -a MPIIO -g -t 20MiB -b 100MiB -o /var/opt/cray/dws/mounts/batch/4882533_striped_scratch//IOR_file
Machine: Linux nid00578

Test 0 started: Mon May  8 10:03:07 2017
Summary:
        api                = MPIIO (version=3, subversion=1)
        test filename      = /var/opt/cray/dws/mounts/batch/4882533_striped_scratch//IOR_file
        access             = single-shared-file
        ordering in a file = sequential offsets
        ordering inter file= no tasks offsets
        clients            = 1 (1 per node)
        repetitions        = 1
        xfersize           = 20 MiB
        blocksize          = 100 MiB
        aggregate filesize = 100 MiB

access    bw(MiB/s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)   total(s)   iter
------    ---------  ---------- ---------  --------   --------   --------   --------   ----
MPIIO WARNING: DVS stripe width of 32 was requested but DVS set it to 2
See MPICH_MPIIO_DVS_MAXNODES in the intro_mpi man page.
MPIIO WARNING: DVS stripe width of 32 was requested but DVS set it to 2
See MPICH_MPIIO_DVS_MAXNODES in the intro_mpi man page.
write     1385.56    102400     20480      0.002353   0.069377   0.000442   0.072173   0
MPIIO WARNING: DVS stripe width of 32 was requested but DVS set it to 2
See MPICH_MPIIO_DVS_MAXNODES in the intro_mpi man page.
MPIIO WARNING: DVS stripe width of 32 was requested but DVS set it to 2
See MPICH_MPIIO_DVS_MAXNODES in the intro_mpi man page.
read      2458.82    102400     20480      0.000481   0.039974   0.000215   0.040670   0
remove    -          -          -          -          -          -          0.000232   0

Max Write: 1385.56 MiB/sec (1452.86 MB/sec)
Max Read:  2458.82 MiB/sec (2578.26 MB/sec)

Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev    Mean(s) Test# #Tasks tPN reps fPP reord reordoff reordrand seed segcnt blksiz xsize aggsize API
write        1385.56    1385.56    1385.56       0.00    0.07217 0 1 1 1 0 0 1 0 0 1 104857600 20971520 104857600 MPIIO 0
read         2458.82    2458.82    2458.82       0.00    0.04067 0 1 1 1 0 0 1 0 0 1 104857600 20971520 104857600 MPIIO 0

Finished: Mon May  8 10:03:07 2017
```

Using dwstat

```
#!/bin/bash

#### Which partition?
#SBATCH -p regular

#### name of the training reservation
###SBATCH --reservation="sc17_bb"

#### How many nodes?
#SBATCH -N 1

#### How long to run the job?
#SBATCH -t 00:5:00

#### Our reservation is for KNL nodes
#SBATCH -C knl

#### Name the job
#SBATCH -J "job_dwstat"

#### Set the output file name
#SBATCH -o "job_dwstat.log"


#### Request a 200GB scratch allocation, striped over BB nodes
#DW jobdw capacity=200GB access_mode=striped type=scratch pool=wlm_pool



#### print the mount point of your BB allocation on the compute nodes
echo "*** DW path is:"
echo $DW_JOB_STRIPED


#### find out which DW nodes your allocation is striped over
echo "*** what nodes are my allocation striped over?"
module load dws

sessID=$(dwstat sessions | grep $SLURM_JOBID | awk '{print $1}')
echo "session ID is: "${sessID}
instID=$(dwstat instances | grep $sessID | awk '{print $1}')
echo "instance ID is: "${instID}
echo "fragments list:"
echo "frag state instID capacity node"
dwstat fragments | grep ${instID}
```

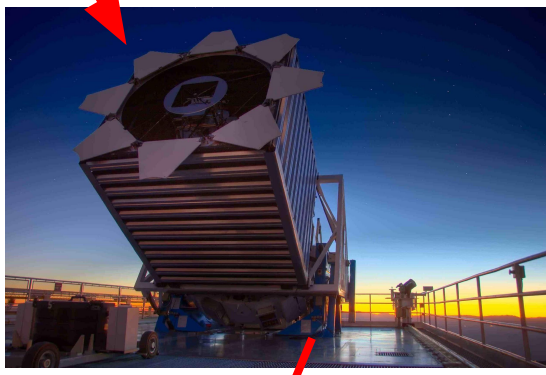
- **Dwstat is a useful tool to learn exactly what's going on with the BB allocations.**
  - Only accessible from compute nodes.
- **Look at "dwstat.sh" for how to find what DW nodes *your* BB allocation is striped over.**

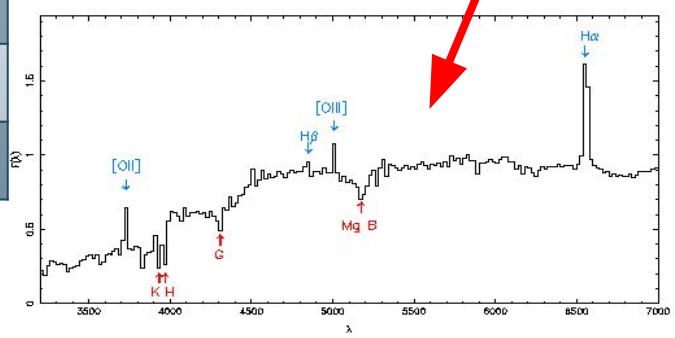
Running IOR on the BB

- **Take a look at "run_IOR.sh"**
  - This script sets all the options to run IOR. You can edit it to run on a scratch BB allocation, or on scratch, or in your home directory.
  - Try some different timings!
  - See if you can get close to max possible bandwidth (~6.5GB/s per BB node)
    - You'll need to drive the IO bandwidth with plenty of threads...

Challenging IO use case: Astronomy data

- **Selecting subsets of galaxy spectra from a large dataset**
  - –Small, random memory accesses
  - –Typical web query for SDSS dataset
  - –Example code in the github repo!





Time taken to extract 1000 random spectra	From one hdf5 file	From individual fits files
From Lustre	44.1s	270s
From BB	1.3s	69s
Speedup:	**33x**	**4x**

Reading fits files from scratch/BB

- **Read 1000 lines from ~300 fits files (typical astronomy workload)**
- **Readfits.conf sets up an interactive BB reservation**
  - Edit this to change paths to point to your own account!

```
#DW jobdw capacity=200GB access_mode=striped type=scratch pool=dev_pool
#DW stage_in source=/global/cscratch1/sd/djbard/SC17_BB_Tutorial_data/spPlates/ destination=$DW_JOB_STRIPED/spPlates/ type=directory
#DW stage_in source=/global/cscratch1/sd/djbard/SC17_BB_Tutorial/intro/readfits.py destination=$DW_JOB_STRIPED/ type=file
#DW stage_in source=/global/cscratch1/sd/djbard/SC17_BB_Tutorial/intro/pmf1k destination=$DW_JOB_STRIPED/ type=file
```

- **Request an interactive session:**
  - `salloc -N 1 -C knl -t 10 --qos=interactive --bbf=readfits.conf`
- **Run the python code on scratch (i.e. in place) and on the BB (i.e.** `cd $DW_JOB_STRIPED` **and run from there)**
- **Compare timing!**

SC17 Tutorial: Getting Started with the Burst Buffer: Using DataWarp Technology

George Markomanolis
Computational Scientist
KAUST Supercomputing Laboratory
georgios.markomanolis@kaust.edu.sa
12 November 2017
Denver, Colorado, USA

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Evaluation

http://bit.ly/sc17-eval

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Outline

- Introduction to Parallel I/O
- Some possible issues with Burst Buffer
- Important MPI environment variables
- Accelerating the performance

Disclaimer: Shaheen DataWarp operates under CLE 5.2

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Material

- Execute the following command to download the corresponding material

  git clone https://github.com/KAUST-KSL/SC17_BB_Tutorial.git

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

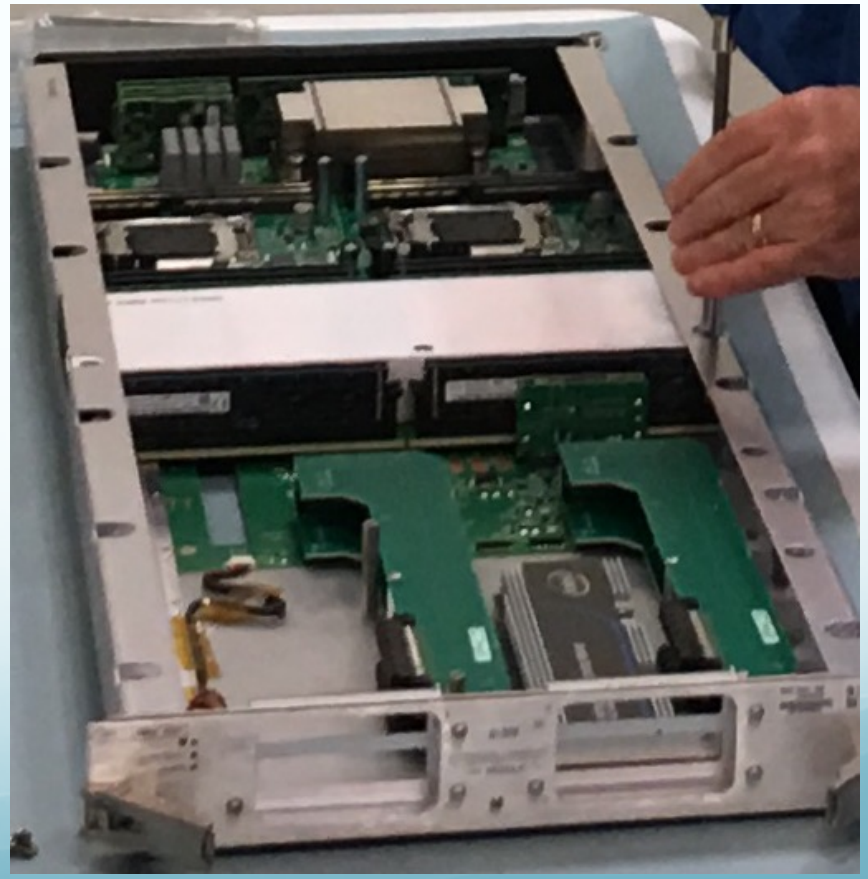
Shaheen II Supercomputer

Compute	Node	Processor type: Intel Haswell	2 CPU sockets per node @2.3GHz 16 processor cores per CPU
		6174 nodes	197,568 cores
		128 GB of memory per node	Over 790 TB total memory
	Power	Up to 3.5MW	Water cooled
	Weight/Size	More than 100 metrics tons	36 XC40 Compute cabinets, disk, blowers, management nodes
	Speed	7.2 Peta FLOPS peak performance	5.53 Peta FLOPS sustained LINPACK and ranked 15th in the latest Top500 list
	Network	Cray Aries interconnect with Dragonfly topology	57% of the maximum global bandwidth between the 18 groups of two cabinets
Storage	Storage	Sonexion 2000 Lustre appliance	17.6 Peta Bytes of usable storage Over 500 GB/s bandwidth
	Burst Buffer	DataWarp	Intel Solid Sate Devices (SDD) fast data cache Over 1.5 TB/s bandwidth
	Archive	Tiered Adaptive Storage (TAS)	Hierarchical storage with 200 TB disk cache and 20 PB of tape storage, using a spectra logic tape library (Upgradable to 100 PB)

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Burst Buffer

- Shaheen II: 268 Burst Buffer (BB) nodes, 536 SSDs, totally 1.52 PB, each node has 2 SSDs, CLE 5.2
- BB adds a layer between the compute nodes and the parallel file system
- Cray DataWarp (DW) I/O is the technology and Burst Buffer is the implementation



جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Introduction to parallel I/O

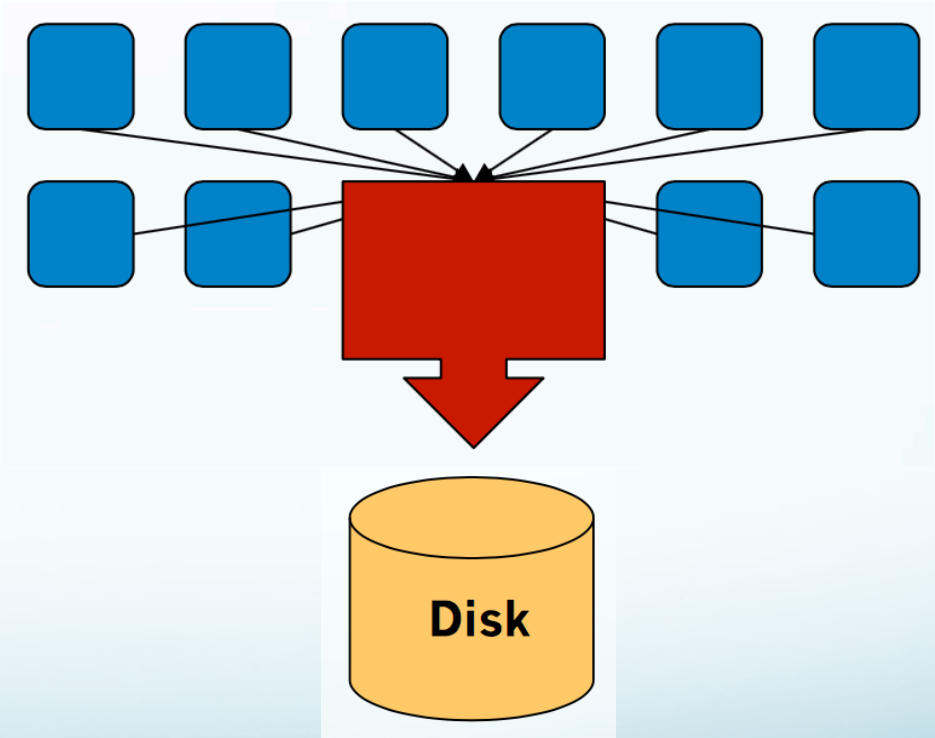
- I/O can create bottlenecks
  - I/O components are much slower than the compute parts of a supercomputer
  - If the bandwidth is saturated, larger scale of execution can not improve the I/O performance
- Parallel I/O is needed to
  - Do more science than waiting files to be read/written
  - No waste of resources
  - Not stressing the file system, thus affecting other users

I/O Performance

- There is no magic solution
- I/O performance depends on the pattern
- Of course a bottleneck can occur from any part of an application
- Increasing computation and decreasing I/O is a good solution but not always possible

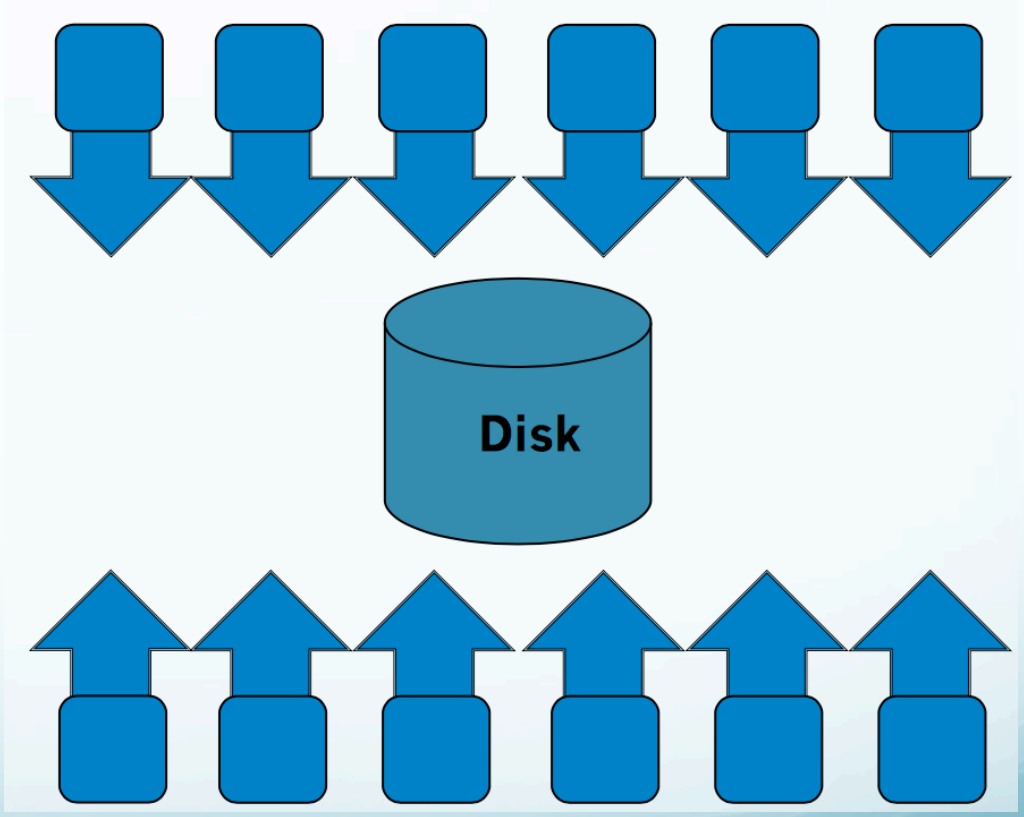
Serial I/O

- Only one process performs I/O (default option for WRF)
  - Data Aggregation or Duplication
  - Limited by single I/O process
- Simple solution but does not scale
- Time increases with amount of data and also with number of processes



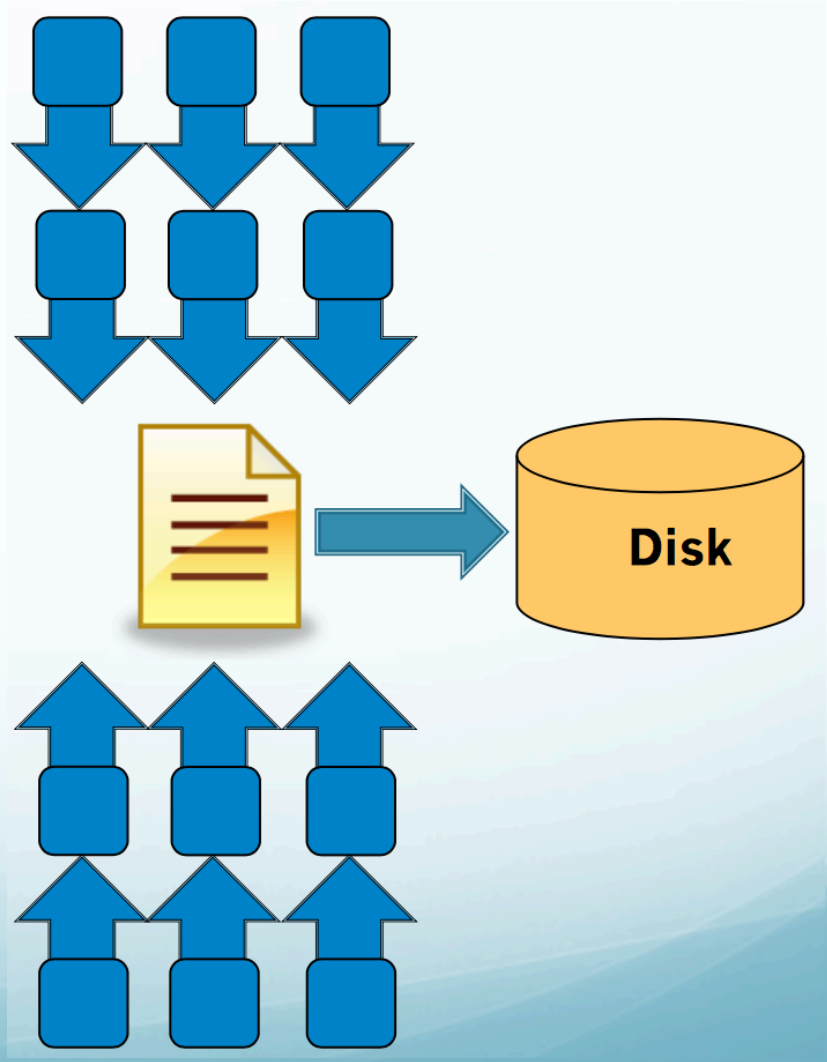
Parallel I/O: File-per-Process

- All processes read/write their own separate file
  - The number of the files can be limited by file system
  - Significant contention can be observed



Parallel I/O: Shared File

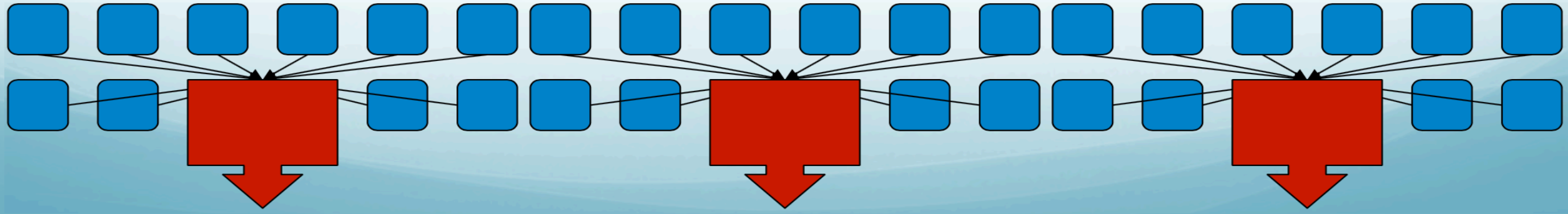
- Shared File
  - One file is accessed from all the processes
  - The performance depends on the data layout
  - Large number of processes can cause contention



جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Pattern Combinations

- Subset of processes perform I/O
  - **Aggregation** of a group of processes data
  - I/O process may access independent files
  - Group of processes perform parallel I/O to a shared file



جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Burst Buffer – Use cases

- Periodic burst
- Transfer to PFS between bursts
- I/O improvements
- Accessed via POSIX I/O requests
- Stage-in/stage-out
- Shared BB allocation for multiple jobs
- Coupling applications

Burst Buffer – Status (KAUST)

- **268** DataWarp (DW) nodes, total 1.52PB with granularity 397.44GB

> ***dwstat most***

```
            pool units   quantity    free      gran
wlm_pool bytes  1.52PiB 1.52PiB 397.44GiB
```

did not find any of [sessions, instances, configurations, registrations, activations]

> **dwstat nodes**

```
    node     pool online drain  gran capacity insts activs
nid00002 wlm_pool   true false 16MiB  5.82TiB     0      0
...
nid07618 wlm_pool   true false 16MiB  5.82TiB     0      0
```

Burst Buffer Nodes Allocation

- How many DW instances per node?

DW_instances_per_node = 5.82*1024/397.44 = **14.995**

A DW node can accommodate up to 14*397.44/1024 = 5.43 TB

- A user requests 60TB of DW nodes, how many DW nodes is he going to reserve (for striped mode explained later)?

On Shaheen II, we have 268 DW nodes, each nodes provides initially one DW instance and when all of them are used, then it starts from the first DW node again. The allocation occurs under round - robin basis

Requested_DW_nodes = **60***1024/397.44 = **154.58**, so we will reserve **155** DW nodes.

Important: If you reserve more than 268 * 397.44/1024 = **104TB**, then some DW nodes will be used twice and this can cause I/O performance issues

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Compute the required DW space

- It is already mentioned that we need to have enough space for our experiments
- If the experiments are about DW scalability and the number of the MPI/OpenMP processes remain stable, then you could modify the MPI I/O aggregators and the number of DW nodes.
- If you need for example 64 DW nodes, then you should calculate the requested space as follows:
  - Multiply with the DW granularity:
    - 64*397.44=25436.16
  - Round down and request this value in the submission script, for example:
    - 25436GiB

DW persistent space I

- Do not submit a second job on the persistent space which stage-in same files:
  - **squeue -u markomg**

    JOBID USER ACCOUNT NAME ST REASON START_TIME TIME TIME_LEFT NODES
    2729358 markomg k01 test PD **burst_buf** N/A 0:00 3:00 40

  - **scontrol show job 2729358**

    …
    JobState=PENDING
    Reason=burst_buffer/cray:_dws_data_in:_**Error_creating_staging_object**_for_file_(/scratch/markomg/burst_buffer_early_access/wrfchem/wrfchem-3.7.1_burst/test/em_real/forburst)_-2_Staging_failures_reported_
    Dependency=(null)
    …
  - **scancel 2729358**

DW persistent space II

- Submit another jobs by either stage-in different files, or without stage-in
- In the case that you want to connect interactively on the compute node to have access to BB and check the files, follow the instructions:
  - Create a file, called it for example persistent.conf with the following:
    - #DW persistentdw name=**george_test**
    - Execute:

      salloc -N 1 -t 00:10:00 --bbf=”persistent.conf”
      srun –N 1 bash –i
      cd $DW_PERSISTENT_STRIPED_george_test
  - markomg@nid00024:/var/opt/cray/dws/mounts/batch/george_test/ss

Use DW persistent space III

- Three jobs were executed on persistent DW space and created a job folder with the job id as their name:
  nid00024:/var/opt/cray/dws/mounts/batch/george_test/ss/ ls -l 2729*

2729356:

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 18:49 wrfout_d01_2007-04-03_00_00_00

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 18:49 wrfout_d01_2007-04-03_01_00_00

2729361:

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 18:57 wrfout_d01_2007-04-03_00_00_00

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 18:57 wrfout_d01_2007-04-03_01_00_00

2729362:

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 19:09 wrfout_d01_2007-04-03_00_00_00

-rw-r--r-- 1 markomg g-markomg 3053617664 Jan 20 19:09 wrfout_d01_2007-04-03_01_00_00

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Use DataWarp for Metadata intensive jobs

- Real case, a user was hurting the metadata server with just one compute node, reading/writing into the same file more than 140 million times.
- Login nodes almost could not be used, lagging for seconds. Users were reporting slow IO.
- Moving the user to DataWarp, we were able to have many parallel executions of the same job without influencing login nodes or other jobs.

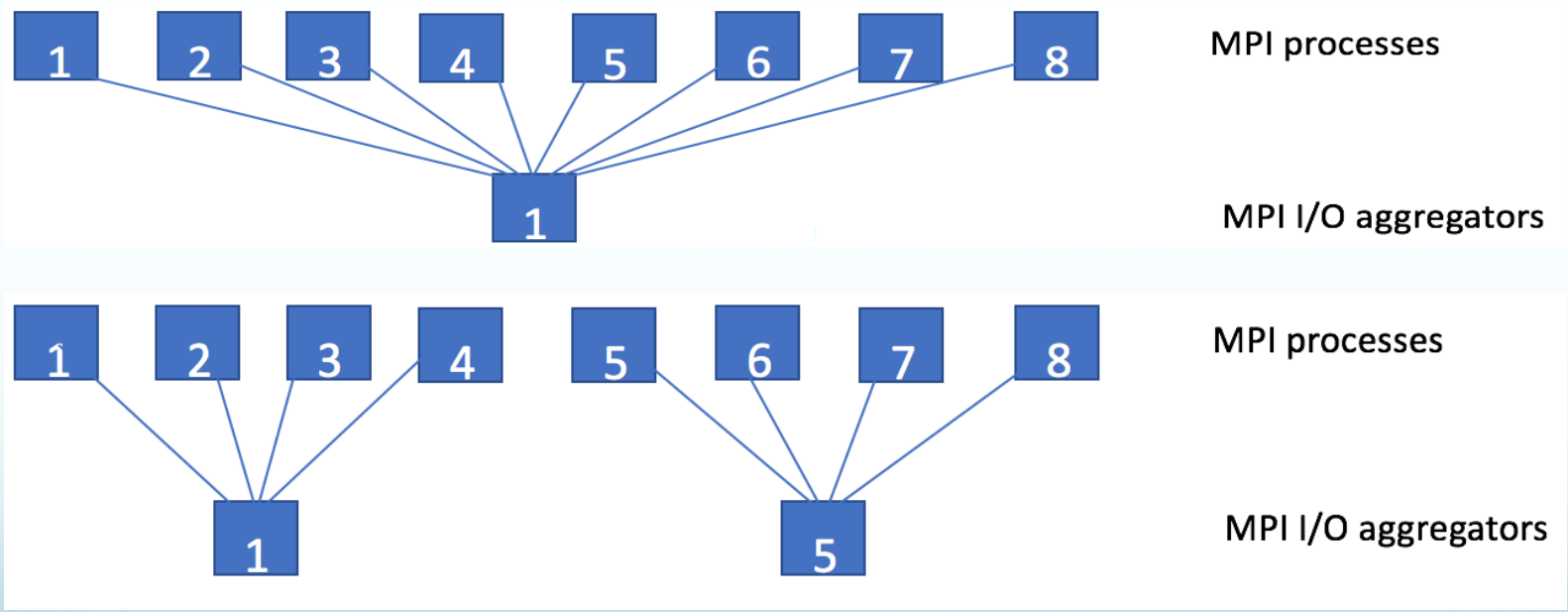
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Useful MPI environment variables

- export MPICH_ENV_DISPLAY=1
  - Displays all settings used by the MPI during execution
- export MPICH_VERSION_DISPLAY=1
  - Displays MPI version
- export MPICH_MPIIO_HINTS_DISPLAY=1
  - Displays all the available I/O hints and their values
- export MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1
  - Display the ranks that are performing aggregation when using MI-I/O collective buffering
- export MPICH_MPIIO_STATS=1
  - Statistics on the actual read/write operations after collective buffering
- export MPICH_MPIIO_HINTS="..."
  - Declare I/O hints
- export MPICH_MPIIO_TIMERS=1
  - Timing statistics for each phase of MPI I/O (requires MPICH v7.5.1)

Collective Buffering – MPI I/O aggregators

- During a collective write, the buffers on the aggregated nodes are buffered through MPI, then these nodes write the data to the I/O servers.
- The default MPI I/O aggregators on Burst Buffer, is the number of BB nodes. So, if we have 8 MPI processes and 1 BB node, then we have 1 MPI I/O aggregator, if we have 2 BB nodes, then we have 2 MPI I/O aggregators.



export MPICH_MPIIO_HINTS="wrfrst*:**cb_nodes**=80,wrfout*:**cb_nodes**=40”

This environment variable is supported on DataWarp since Cray-MPICH v7.4

Extract the list of the MPI I/O aggregators nodes

- export **MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY**=1
- First case:

AGG	Rank	nid
0	0	nid04184

- Second case:

AGG	Rank	nid
0	0	nid00292
1	8	nid00294
...		
63	1144	nid04592

Profiling MPI I/O on BB

Question: Using 160 nodes with 1 MPI process per node and 2TB of DW space (6 DW nodes) with MPI I/O through PnetCDF, how many MPI I/O aggregators are saving the NetCDF file on BB?

Answer: 6!

Table 6: File Output Stats by Filename

Write Time	Write MBytes	Write Rate MBytes/sec	Writes	Bytes/ Call	File Name PE
710.752322	988,990.668619	1,391.470191	671,160.0	1,545,133.62	Total
263.824253	369,720.282763	1,401.388533	46,690.0	8,303,272.98	wrfrst_d01_2009-12-18_00_30_00
45.299442	61,624.000000	1,360.369943	7,798.0	8,286,412.85	pe.96
44.410365	61,616.000000	1,387.423860	7,795.0	8,288,525.83	pe.160
43.762797	61,623.999999	1,408.136675	7,763.0	8,323,772.69	pe.32
43.708663	61,616.148647	1,409.701068	7,762.0	8,323,784.42	pe.0
43.532686	61,616.134117	1,415.399323	7,764.0	8,321,638.26	pe.128
43.110299	61,624.000000	1,429.449598	7,808.0	8,275,800.13	pe.64
0.000000	0.000000	--	0.0	--	pe.1
0.000000	0.000000	--	0.0	--	pe.2
0.000000	0.000000	--	0.0	--	pe.3
0.000000	0.000000	--	0.0	--	pe.4
0.000000	0.000000	--	0.0	--	pe.5
0.000000	0.000000	--	0.0	--	pe.6
0.000000	0.000000	--	0.0	--	pe.7

How do we choose the number of MPI I/O aggregators on BB?

- In this example we have parallel I/O and we can adjust the number of the MPI processes for simulating an application
- MPICH_MPIIO_HINTS
  - export MPICH_MPIIO_HINTS="wrfrst*:**cb_nodes**=80,wrfout*:**cb_nodes**=40”
  - This environment variable is supported on DataWarp since Cray-MPICH v7.4
- In this case we select 80 MPI I/O aggregators for the files starting with the name wrfrst*, and 40 MPI I/O aggregators for the files starting with the name wrfout*.
- Although this depends on the application, according to our experience, if you have one MPI I/O aggregator per DW node (default behavior), the performance is not always good. In order to stress the SSDs of the DW node, more than one MPI process should write data per DW node, and this happens with the MPI I/O aggregators.
- Depending on the size of the file, some times we need to use different number of MPI I/O aggregators per file.

Tips on selecting MPI I/O aggregators and number of BB nodes

- Tips:
  - The number of the MPI I/O aggregators should divide the number of total MPI processes for better load balancing. For example, If you have 1024 MPI processes, do not declare 100 MPI I/O aggregators, but 128 or 64. However, in some cases it does not matter.
  - The number of the requested DW nodes, should divide the number of the MPI I/O aggregators for better load balancing also.
  - Of course the requested DW nodes should provide enough data for all of your experiments, thus there is a minimum amount of needed DW nodes.

- $MPI\_IO\_aggregators = \begin{cases} DW\_nodes, if\ we\ use\ one\ aggregator\ per\ DW\ node \\ k * DW\_nodes, where\ k \in \mathbb{N}, 2 \leq k \leq 16 \end{cases}$
- Number_of_total_MPI_processes= $l * MPI\_IO\_aggregators, where\ l \in \mathbb{N}, l \geq 2$

How to compile your code for better performance on DW

- Because of the MPICH_MPIIO_HINTS environment variable, Cray-MPICH v**7.4** and later is required.
- Load the appropriate versions through cdt module:
  - **module load cdt/16.08**

  Switching to atp/2.0.2.
  Switching to cce/8.5.2.
  Switching to cray-libsci/16.07.1.
  Switching to cray-mpich/7.4.2.
  Switching to craype/2.5.6.
  Switching to modules/3.2.10.4.
  Switching to pmi/5.0.10-1.0000.11050.0.0.ari.

- Then compile your application

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Using Darshan tool

- Have you ever used Darshan tool?
  - If the answer is “I don’t know, probably not”, then maybe you have used it, as it is enabled automatic on Shaheen II and Cori.
- KAUST Supercomputing Laboratory (KSL) provides a framework to provide you easy access to performance data from Darshan:
  - Visit web page https://kaust-ksl.github.io/HArshaD/ for instructions. The framework is supported on both Shaheen and Cori, Darshan v2.x and v3.x.

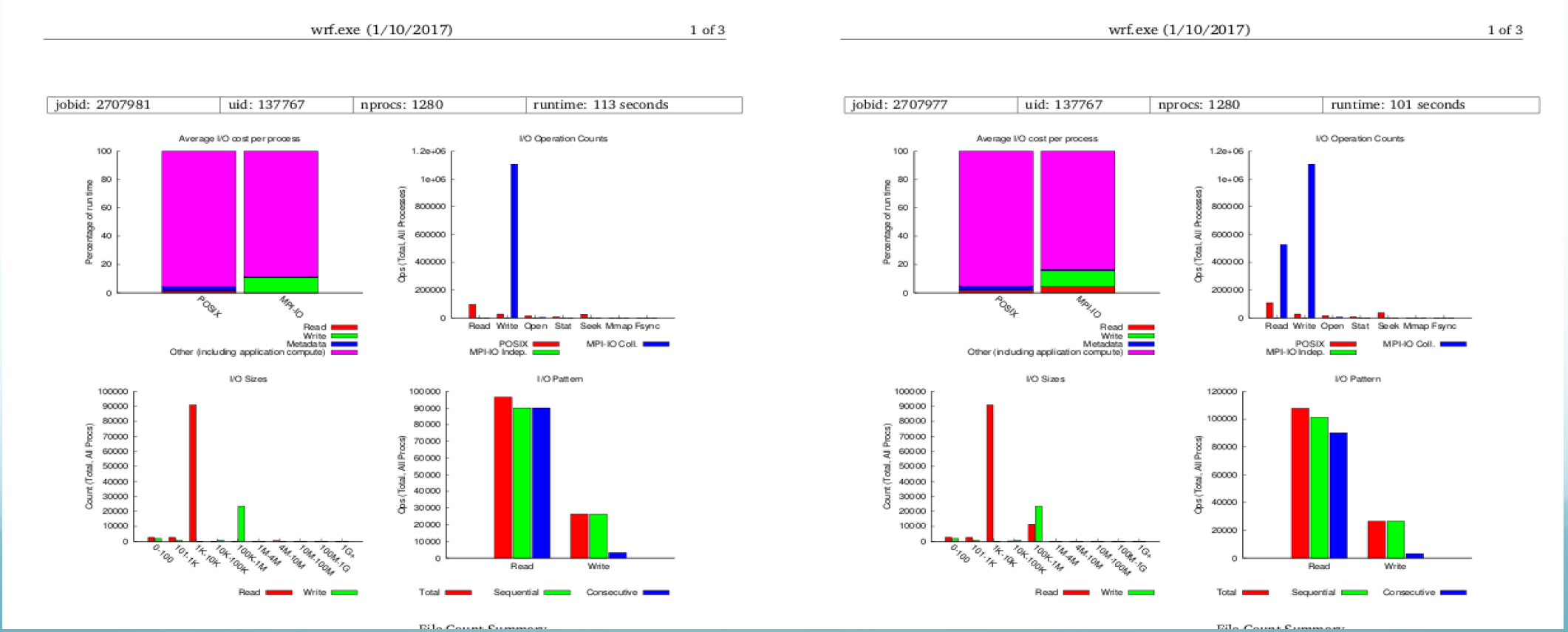
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

HArsaD I

- Get the Darshan performance data from your last experiment, execute:
  - ./open_darshan.sh
- Get the Darshan performance data from the job id 65447, execute:
  - ./open_darshan.sh 65447
- Compare Darshan perfromance data from job id 65447 and 65448, execute:
  - ./compare_darshan 65447 65448

HArshaD II - Comparison

- In case that you want to compare the execution of two applications, execute:
  - *compare_darshan.sh job1_id job2_id*
  - **One** PDF file, with the Darshan performance data of both executions, is created



Applications/Benchmarks

Study-case NAS BTIO

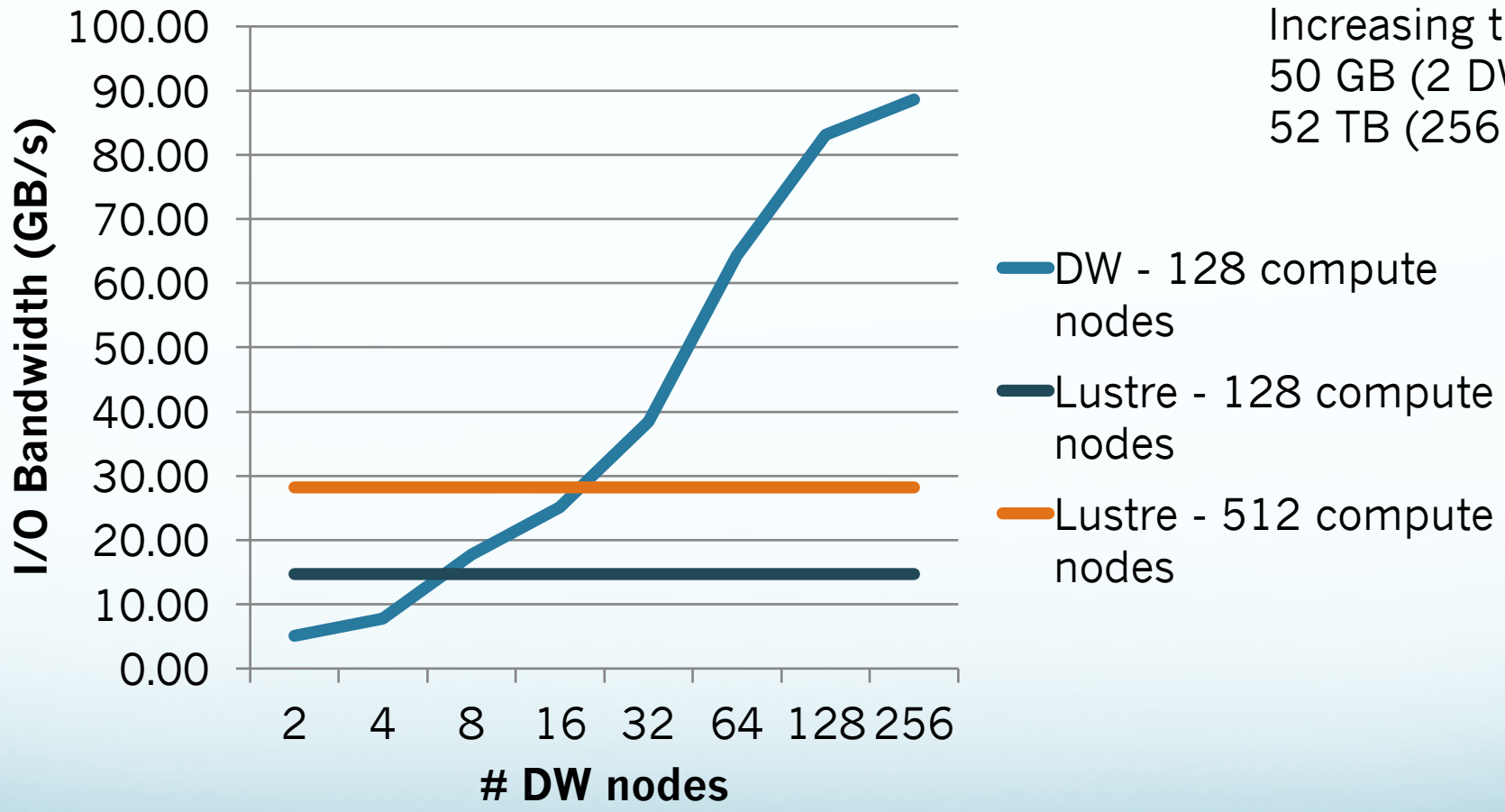
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Applications

NAS BTIO

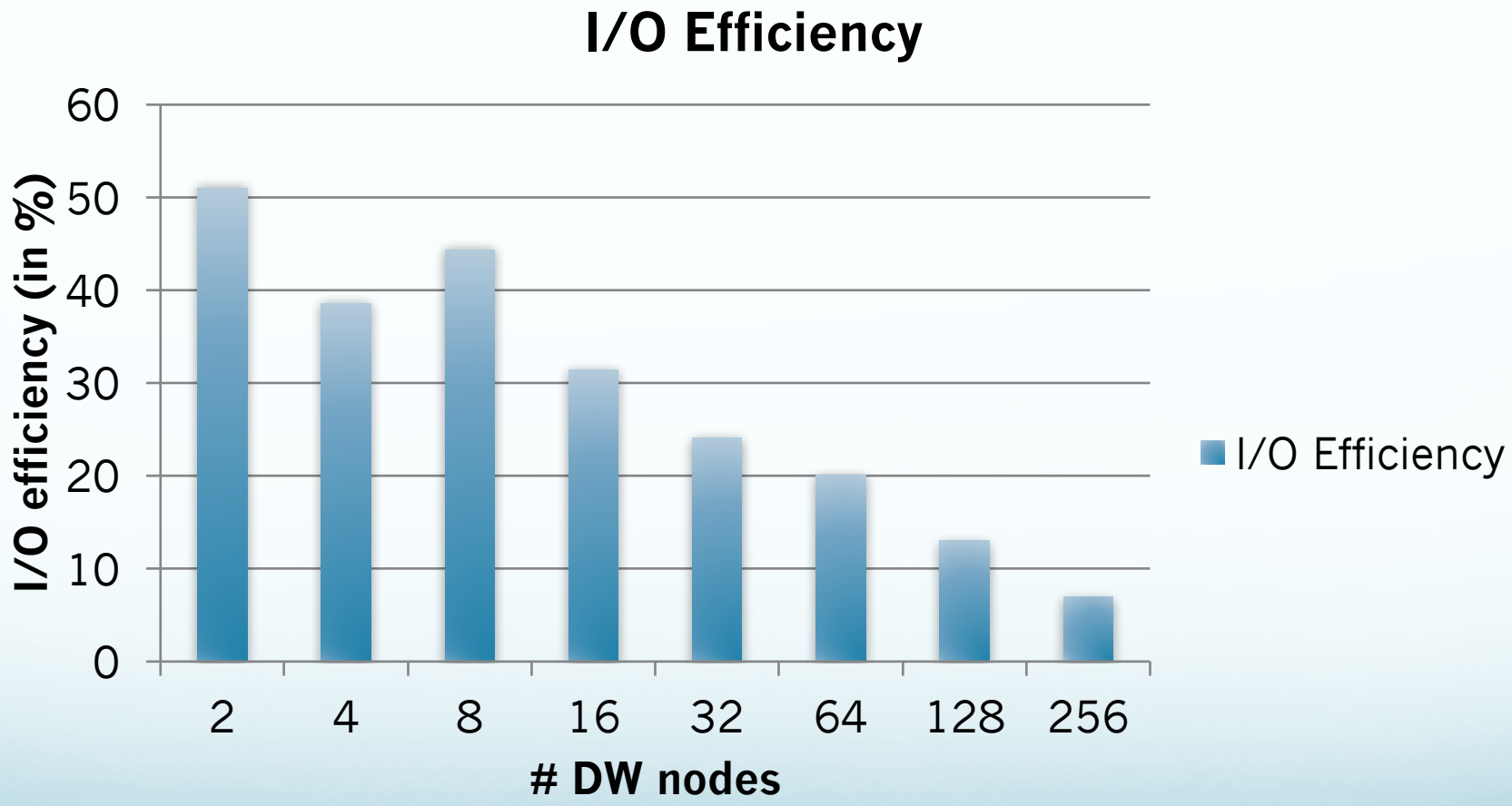
"As part of the NAS parallel benchmark set an IO benchmark has been developed which is based on one of the computational kernels. The BT benchmark is based on a CFD code that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations."

NAS – BT I/O Benchmark - PNetCDF



Increasing the file size from 50 GB (2 DW nodes) up to 52 TB (256 DW nodes)

NAS – BT I/O Benchmark - PNetCDF



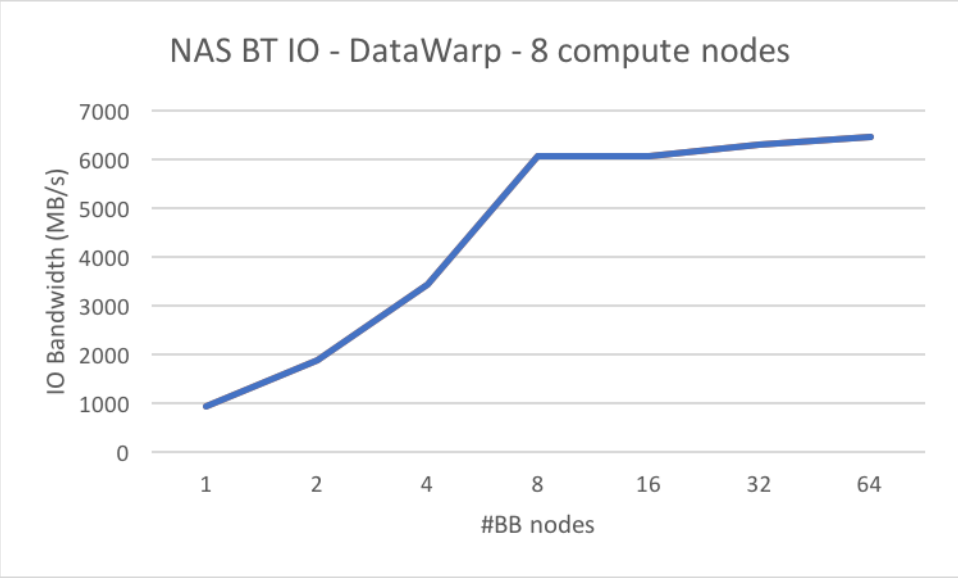
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

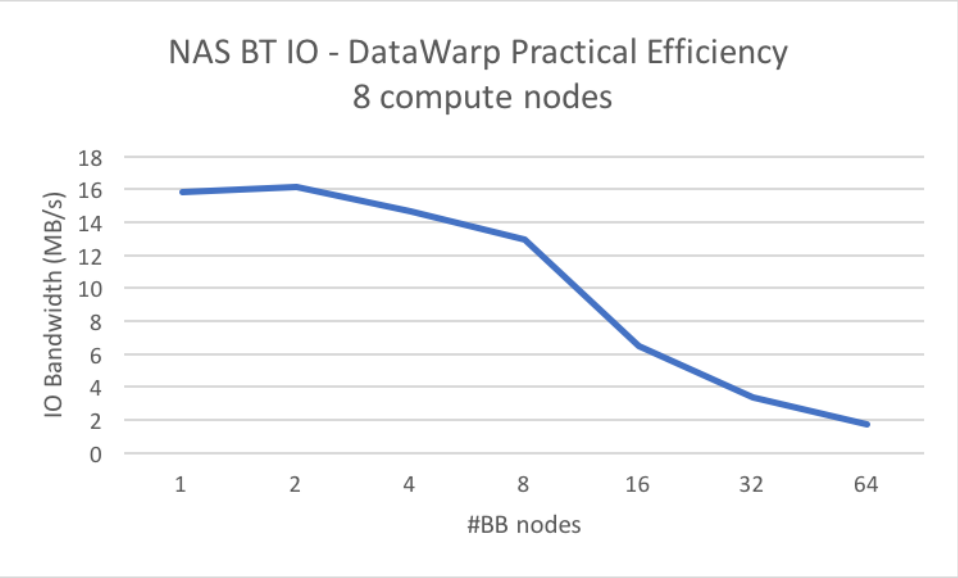
Applications

- NAS BT I/O
- Domain size: 1024 x 512 x 256
- 256 to 1024 MPI processes, 8 – 32 nodes
- Size of output file: 50 GB

Burst Buffer nodes

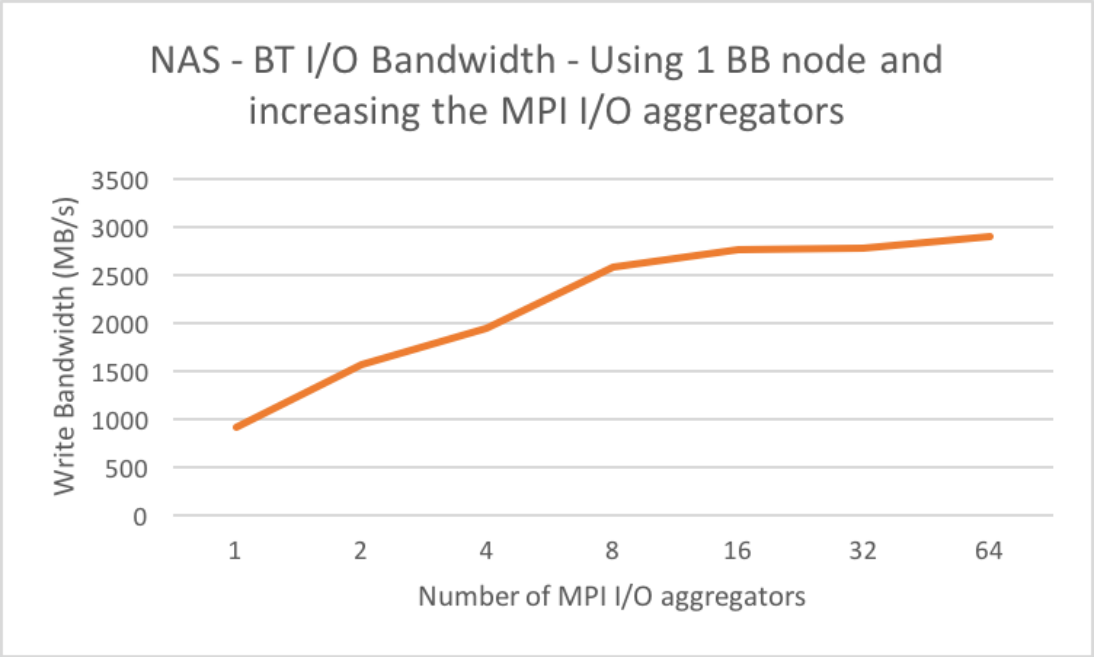
- A user tries to scale his application on Burst Buffer by just increasing the BB nodes and this does not always provide the best results.
- Increasing the BB nodes by 64 times, provide less than 8 times better performance and the practical efficiency is less than 2%!





Collective Buffering – MPI I/O aggregators II

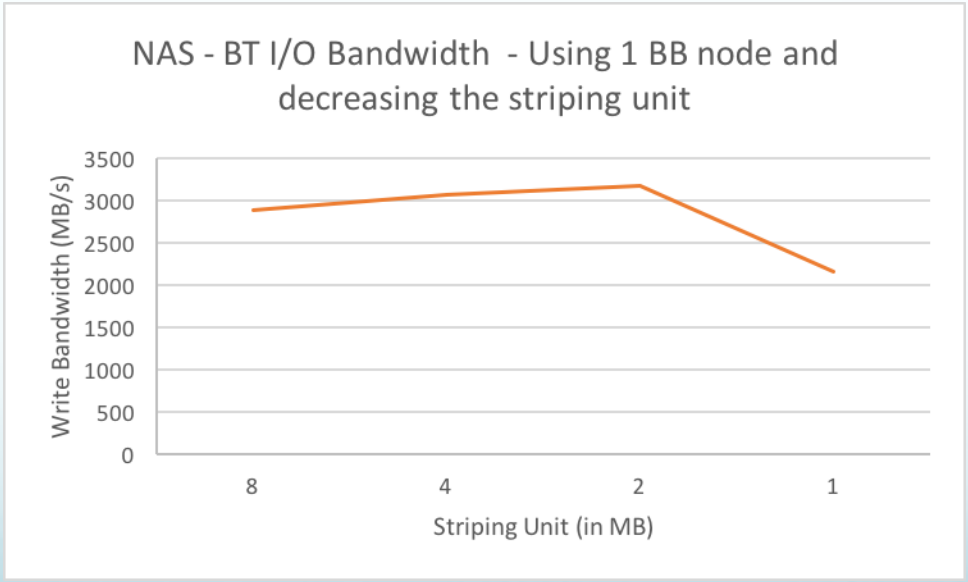
- Using optimized MPI I/O aggregators improved the performance up to 3,11 times on just one BB node.
- We achieved best performance with 64 MPI I/O aggregators



Use 64 MPI I/O aggregators for the file btio.nc: export MPICH_MPIIO_HINTS=btio.nc:**cb_nodes=64**

Striping Unit

- The stripe units are the segments of sequential data written to or read from a disk before the operation continues to the next disk
- For NAS BT IO, decreasing the striping unit up to 2 MB, increases the performance by 10%.



Change striping unit of file btio.nc to 2MB:
export MPICH_MPIIO_HINTS="btio.nc:cb_nodes=64:**striping_unit=2097152"**

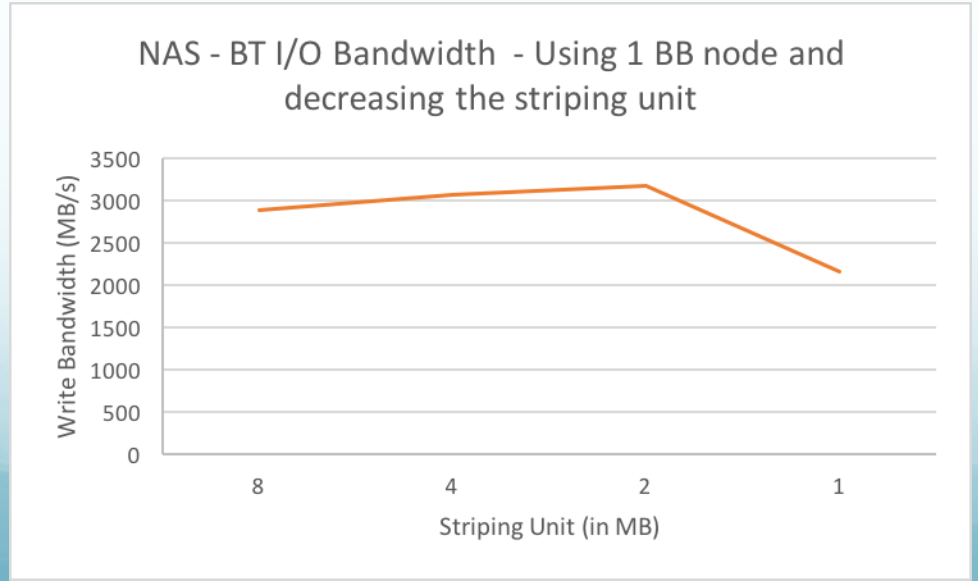
NAS BT I/O - Understanding striping unit

2897 MB/s

```
| MPIIO write access patterns for
|/var/opt/cray/dws/mounts/batch/3129772/ss//btio.nc
|   independent writes      = 11
|   collective writes       = 40960
|   independent writers     = 1
|   aggregators             = 64
|   stripe count            = 1
|   stripe size             = 8388608
|   system writes           = 6411
|   stripe sized writes     = 6400
|   total bytes for writes  = 53687091532 = 51200 MiB = 50 GiB
|   ave system write size   = 8374214
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 21
|   ave write gap size      = 23336707978
```

2165 MB/s

```
| MPIIO write access patterns for
|/var/opt/cray/dws/mounts/batch/3151099/ss//btio.nc
|   independent writes      = 11
|   collective writes       = 40960
|   independent writers     = 1
|   aggregators             = 64
|   stripe count            = 1
|   stripe size             = 1048576
|   system writes           = 51211
|   stripe sized writes     = 51200
|   total bytes for writes  = 53687091532 = 51200 MiB = 50 GiB
|   ave system write size   = 1048350
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 21
|   ave write gap size      = 23297910666
```



By **decreasing** the **striping unit** by 8 times, the **system writes** were increased by 8 times.

Doubling the number of **BB nodes** from 1 to 2, the I/O bandwidth from 2165 MB/s becomes 3850 MB/s, **78.2%** improvement.

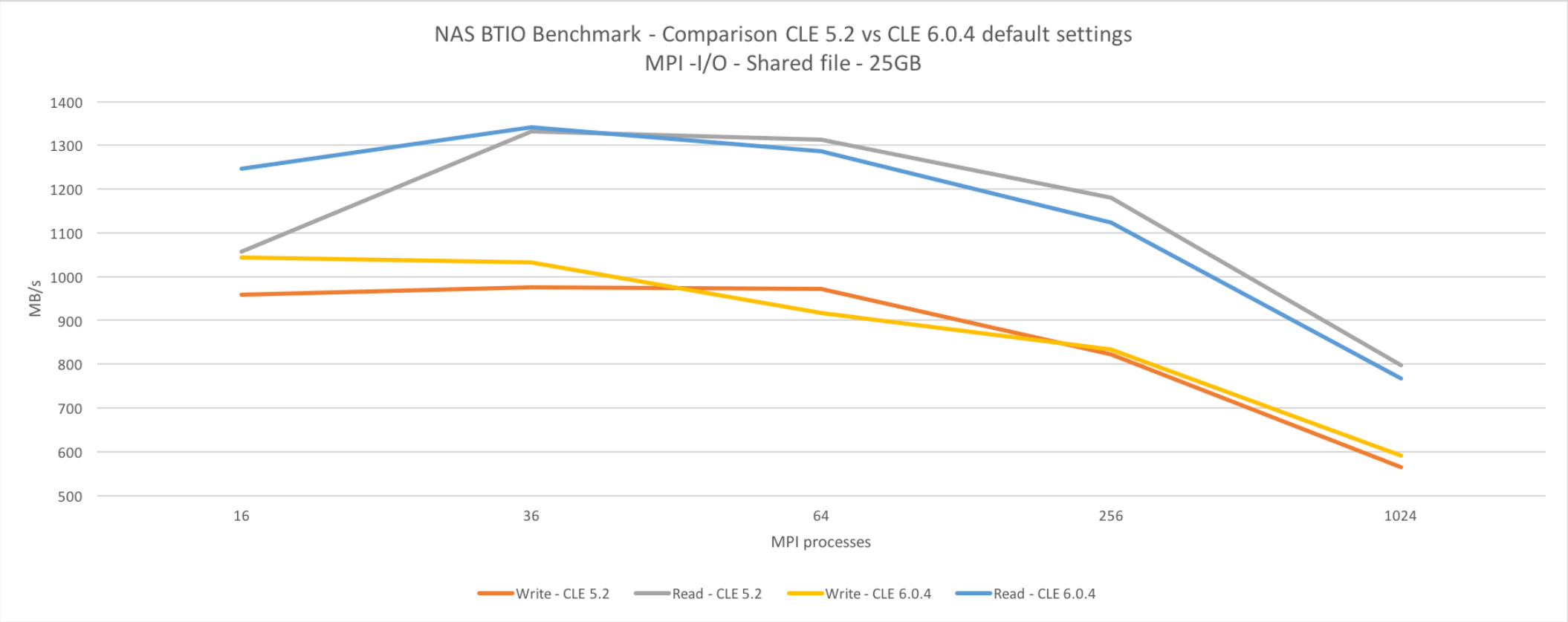
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CLE comparison

- Cray provides the CLE 6 with new functionalities and performance improvements.
- In the next slides we compare the CLE 5.2 (Shaheen II) vs 6.0.4 (Cori).
- We use the cdt/17.02 as it is the last available on both sites
- We use NAS BTIO, with a domain which leads to a shared output file of 25GB.
- We use 1 BB node

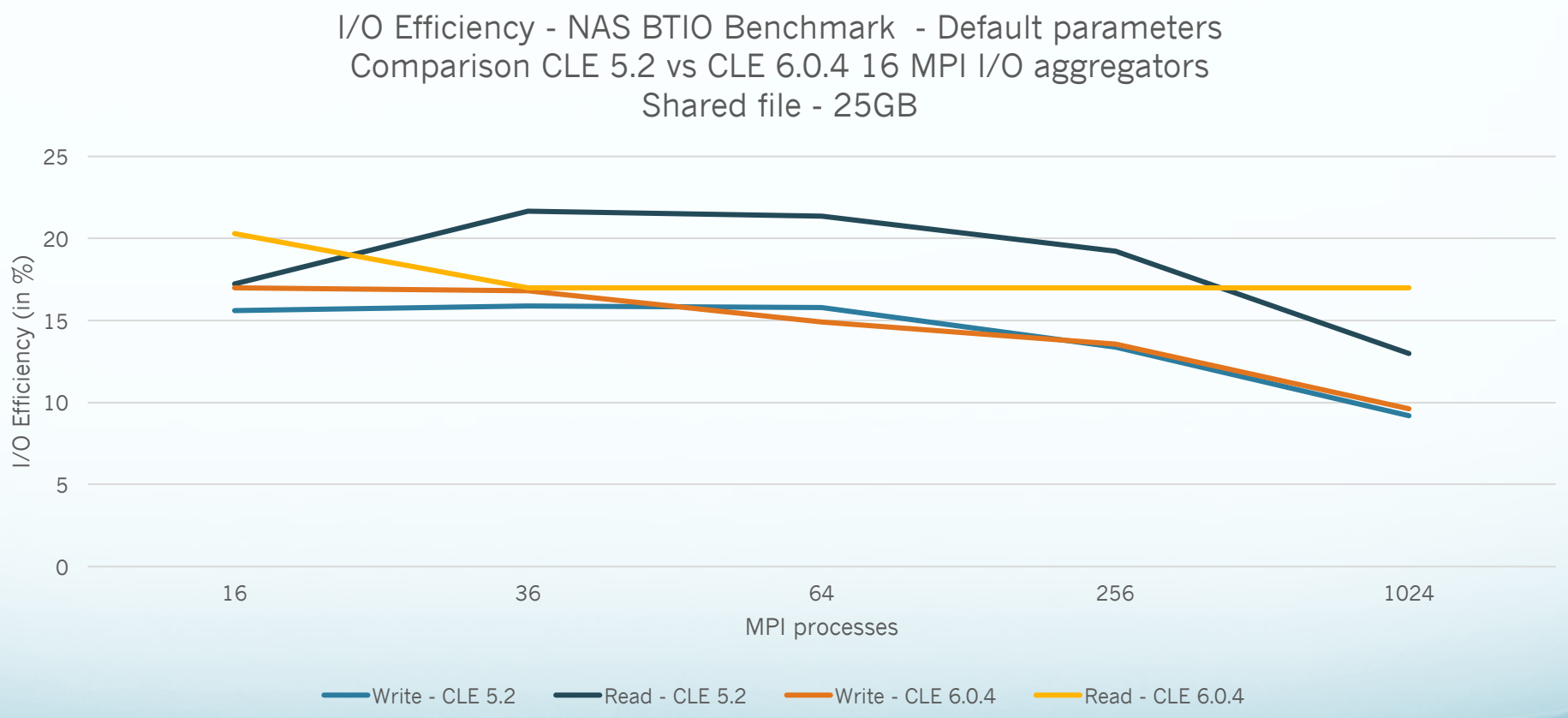
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CLE 5.2 vs 6.0.4 – default settings

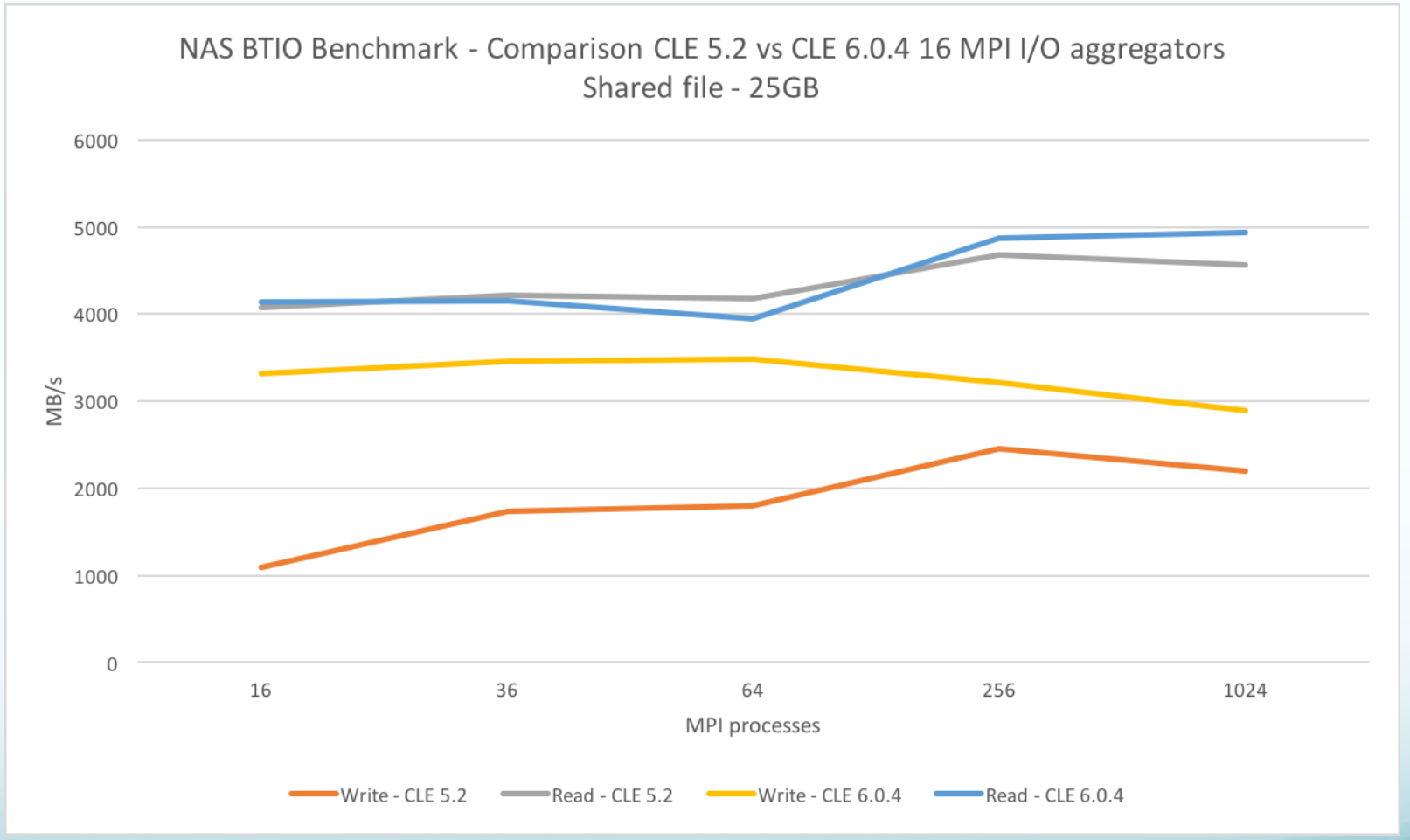


With default settings, there is no significant performance difference between the CLE 5.2 and 6.0.4 in this specific case.

I/O Efficiency – Default parameters

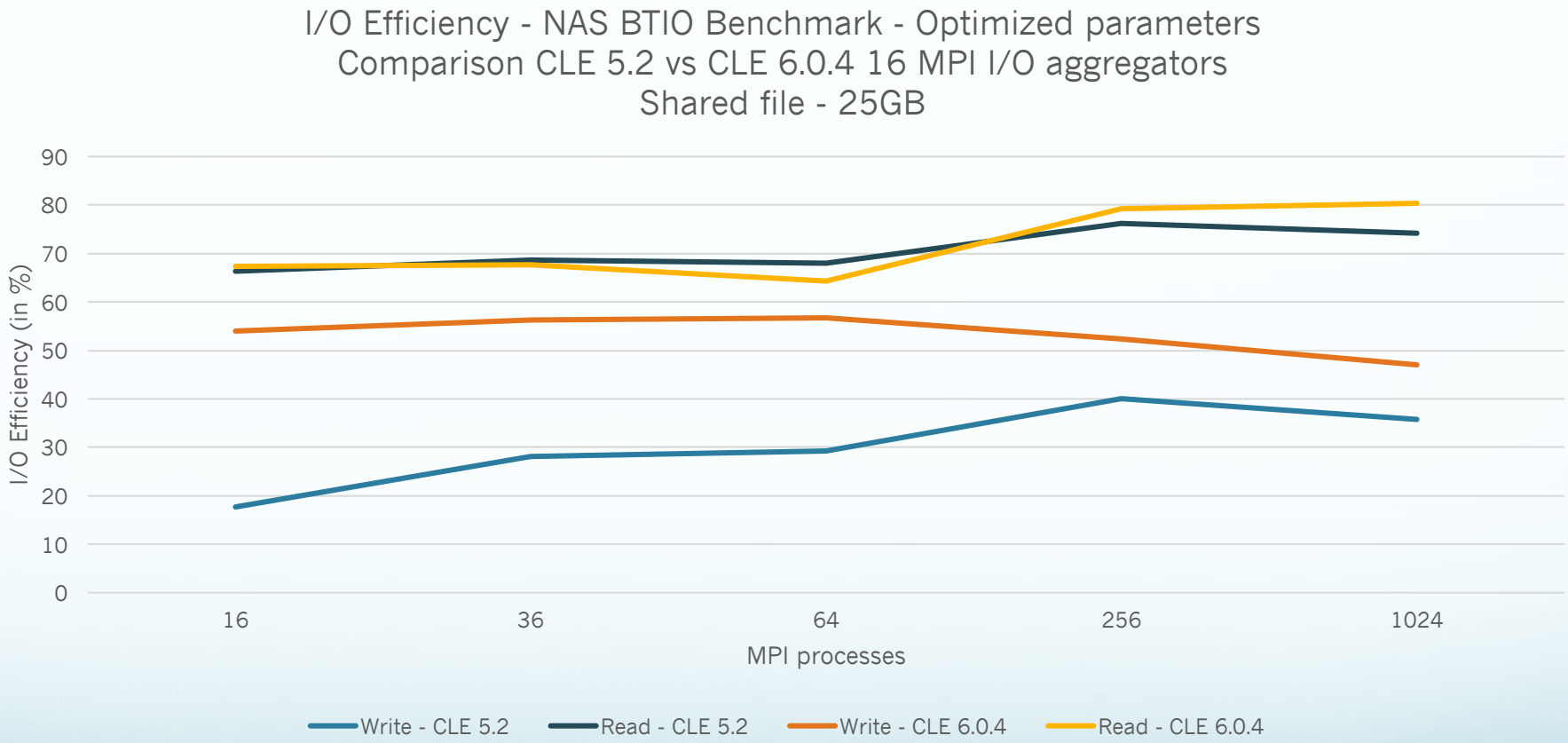


CLE 5.2 vs 6.0.4 – optimized parameters



By using more MPI I/O aggregators, CLE 6.0.4 achieves up to 3 times better write speed. The performance of reading a file seems similar between the CLE.

I/O Efficiency – Optimized parameters



Study-case Neuromap

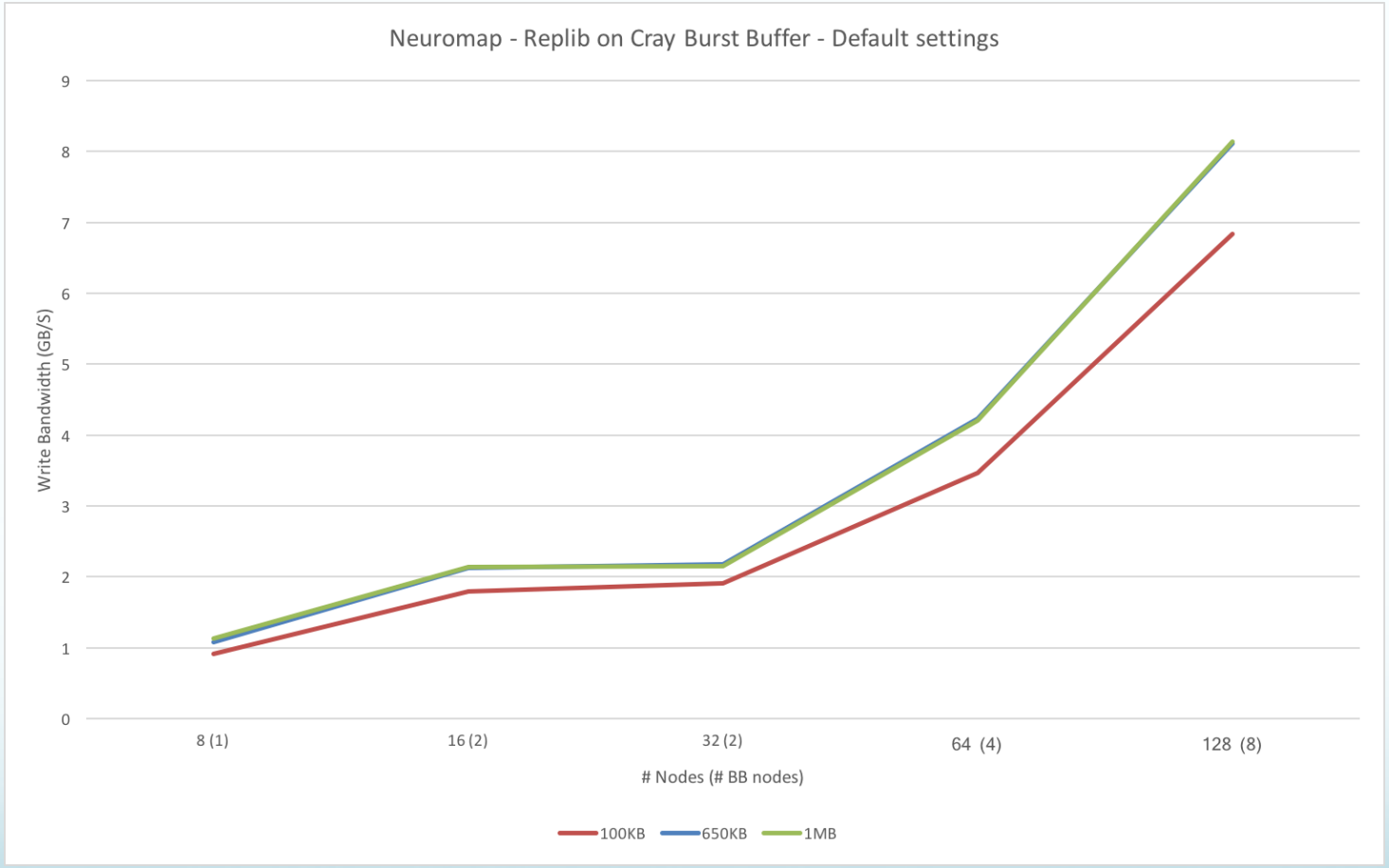
Application provided for the SC17 tutorial

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Neuromap - Replib

- The Neuronm(ini)app(lication) library reproduces the algorithms of the main software of the Blue Brail Project as a collection of mini-apps For its first release, the Neuromapp framework focuses on CoreNeuron application.
- Replib is a miniapp that mimics the behavior of Neuron's ReportingLib. It uses MPI I/O collective calls to write a fake report to a shared file. The miniapp provides several options to distribute data across ranks in different ways.
- Contact person: Judit Planas
- Information: https://insidehpc.com/2017/08/video-io-challenges-brain-tissue-simulation/

Neuromap – Replib on Cray Burst Buffer
Default parameters



- We have three cases, writing data in chunks of 100KB, 650KB, and 1MB
- We save the data in a shared file, each MPI process saves its own data and the size of the output file varies from 9 GB up to 400 GB

MPI I/O Statistics – Default parameters

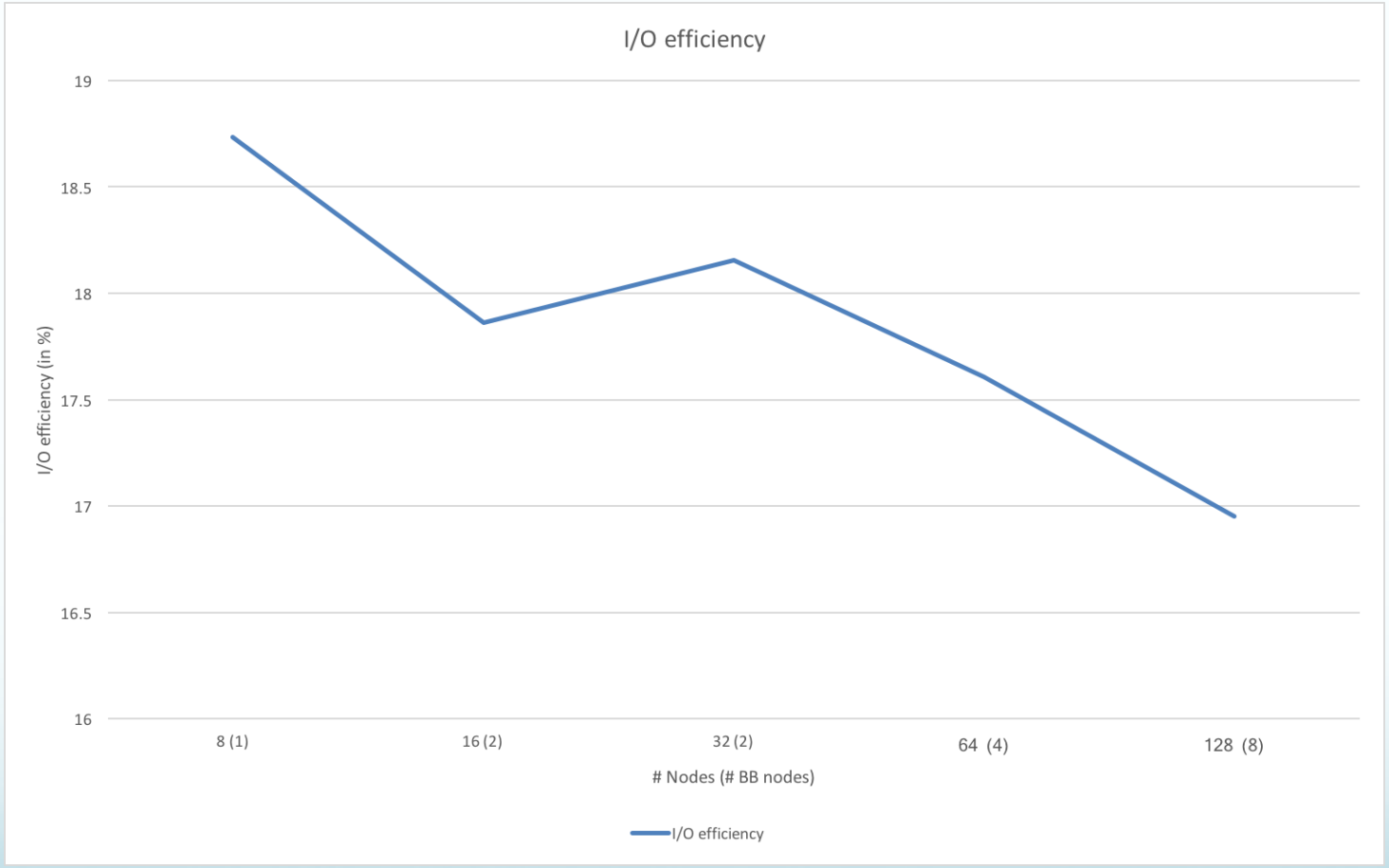
For 32 nodes with default settings:

```
+-----------------------------------------------------------------------+
| MPIIO write by phases, writers only, for /var/opt/cray/dws/mounts/batch/3774697/ss//out2
|                                  min         max         ave
|                              ----------  ----------  ----------
|   file write        time      =     22.92       23.58       23.25
|   time scale: 1 = 2**7     clock ticks    min         max         ave
|   total                     =                    523689105
|   imbalance                 =     148522      248791      198657         0%
|   local compute             =    4516667     4527122     4521894         0%
|   wait for coll             =    1225864     7717977     4471921         0%
|   collective                =    1092307     1149546     1120927         0%
|   exchange/write            =     890825      908929      899877         0%
|   data send                 =   90654295    96061956    93358125        17%
|   file write                =  412044716   423894304   417969510        79%
|   other                     =     568026      633007      600516         0%
|   data send BW (MiB/s)          =                      24.445
|   raw write BW (MiB/s)          =                    2795.602
|   net write BW (MiB/s)          =                    2231.241
+-----------------------------------------------------------------------+
```

The data send bandwidth is quite slow because all the MPI processes send data to just two MPI I/O aggregators (2 BB nodes) and it takes 17% of the total time. The net write BW is 2231 MB/s because we have one MPI process per BB node that writes data.

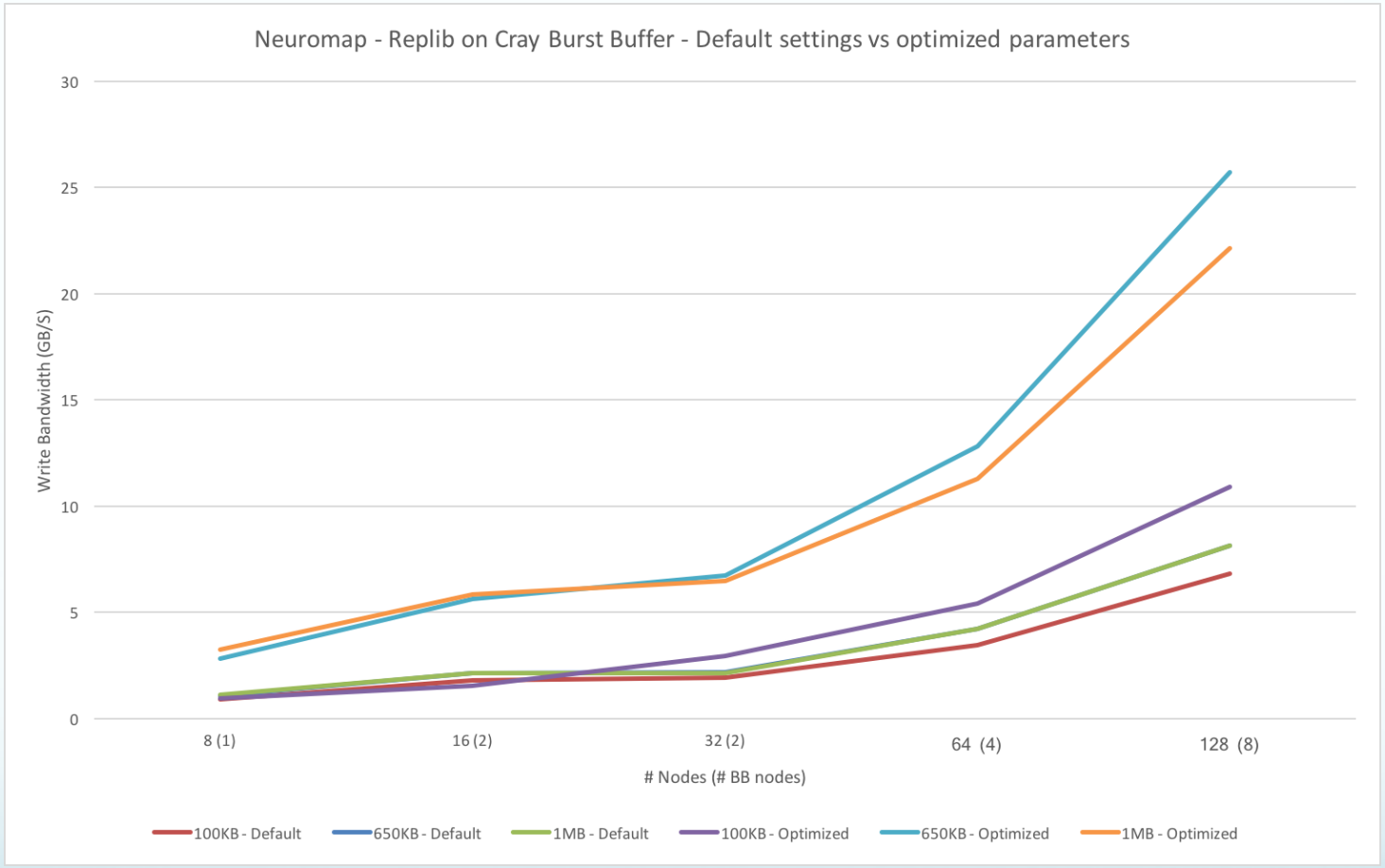
Neuromap – Replib on Cray Burst Buffer

I/O Efficiency – Default parameters



The maximum I/O efficiency is less than 19% for all the cases. We will tune the parameters for better performance.

Neuromap – Replib on Cray Burst Buffer Comparison with optimized parameters



- In order to stress the SSDs, we increase the MPI I/O aggregators, according to our tests we can even disable the collective I/O. Optimization declaration for the case of 650KB:

MPICH_MPIIO_HINTS="$DW_JOB_STRIPED/out2*:romio_ds_write=disable:romio_cb_write=disable:striping_unit=665600"

- The performance was improved up to 3,16 times.

MPI I/O Statistics – Optimized parameters

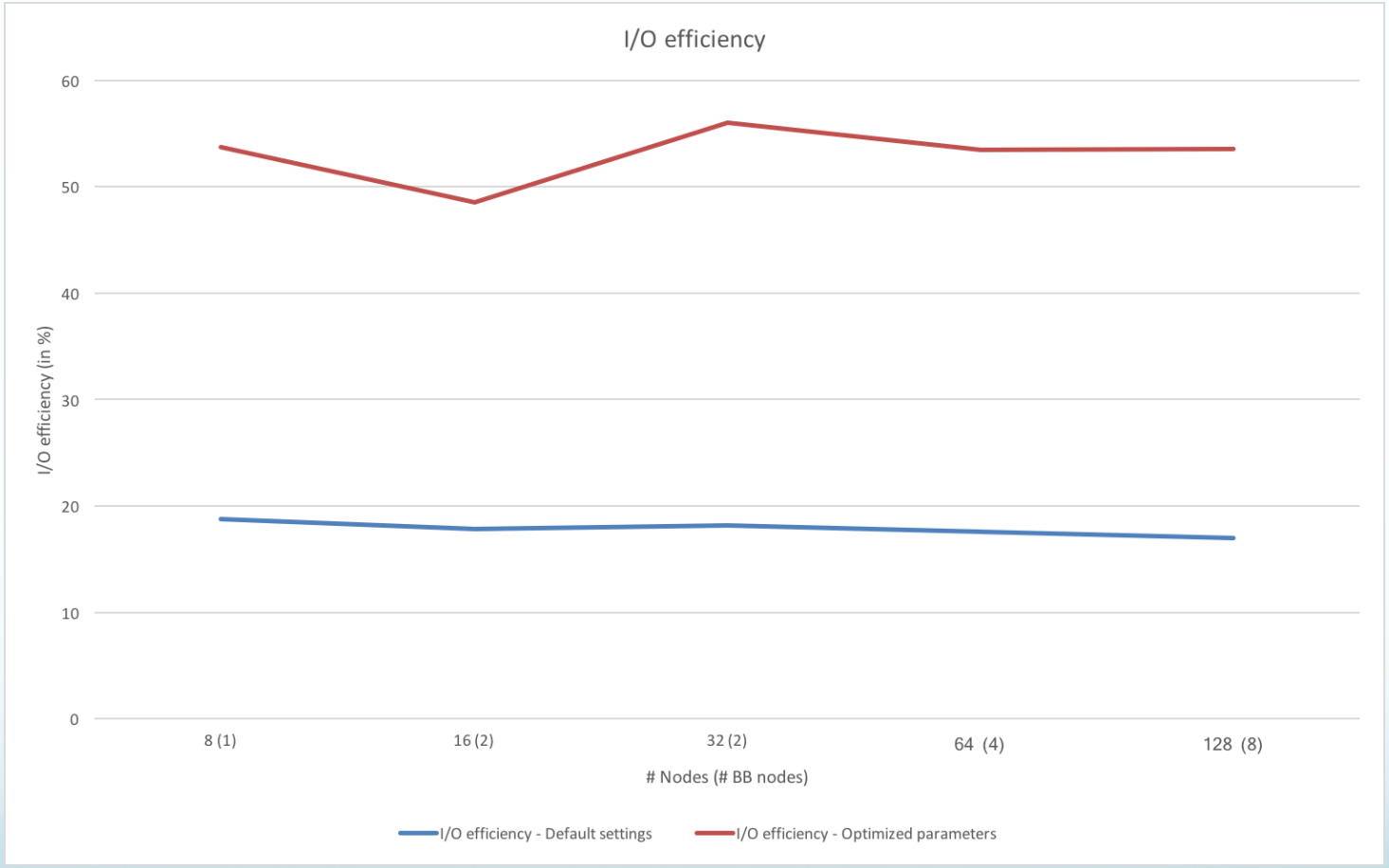
```
+-------------------------------------------------------------------+
| MPIIO write by phases, all ranks, for /var/opt/cray/dws/mounts/batch/3774937/ss//out2
|   number of ranks writing       =  1024
|   number of ranks not writing   =     0
|                                    min         max         ave
|                                ----------  ----------  ----------
|   open/close/trunc  time       =      0.02        0.04        0.03
|   file write        time       =      0.45        6.37        4.46
|   time scale: 1 = 2**5     clock ticks    min         max         ave
|                                ----------  ----------  ---------- ---
|   total                        =                   707678293
|   imbalance                    =    394311     1217998      866269        0%
|   open/close/trunc             =   1734852     2731840     2312575        0%
|   local compute                =    329159    29586804    13915952        1%
|   wait for coll                =  226899601   671583722   369929561       52%
|   file write                   =  32257150   457948530   320653934       45%
|   other                        =         0           0           0        0%
|   raw write BW (MiB/s)         =                   14576.170
|   net write BW (MiB/s)         =                    6604.564
+-------------------------------------------------------------------+
```

The bottleneck of the data send does not exist anymore because each MPI process saves its data independent from the other ones aggregators (collective I/O is disabled) and the net write BW is almost 3x times faster than the default parameters.

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

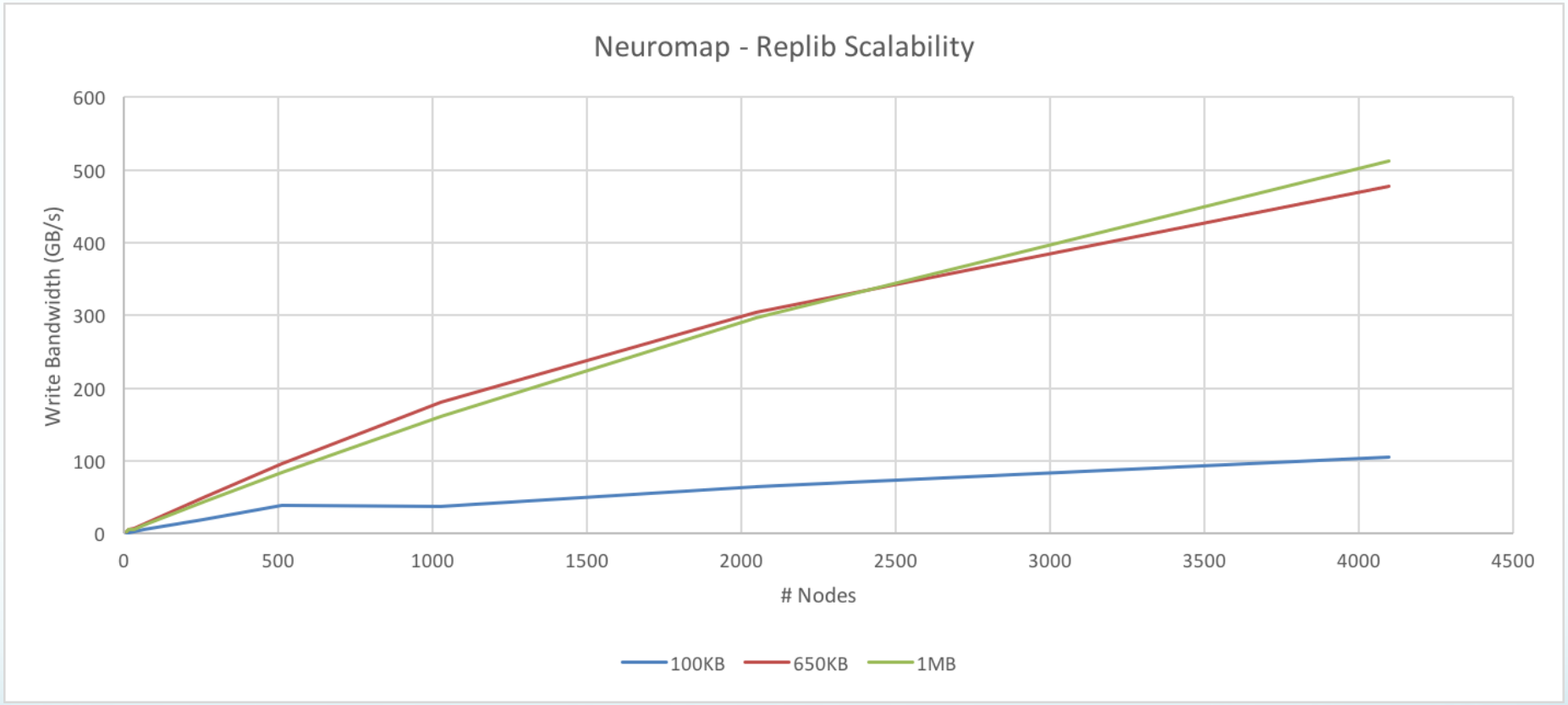
Neuromap – Replib on Cray Burst Buffer

I/O efficiency comparison with optimized parameters



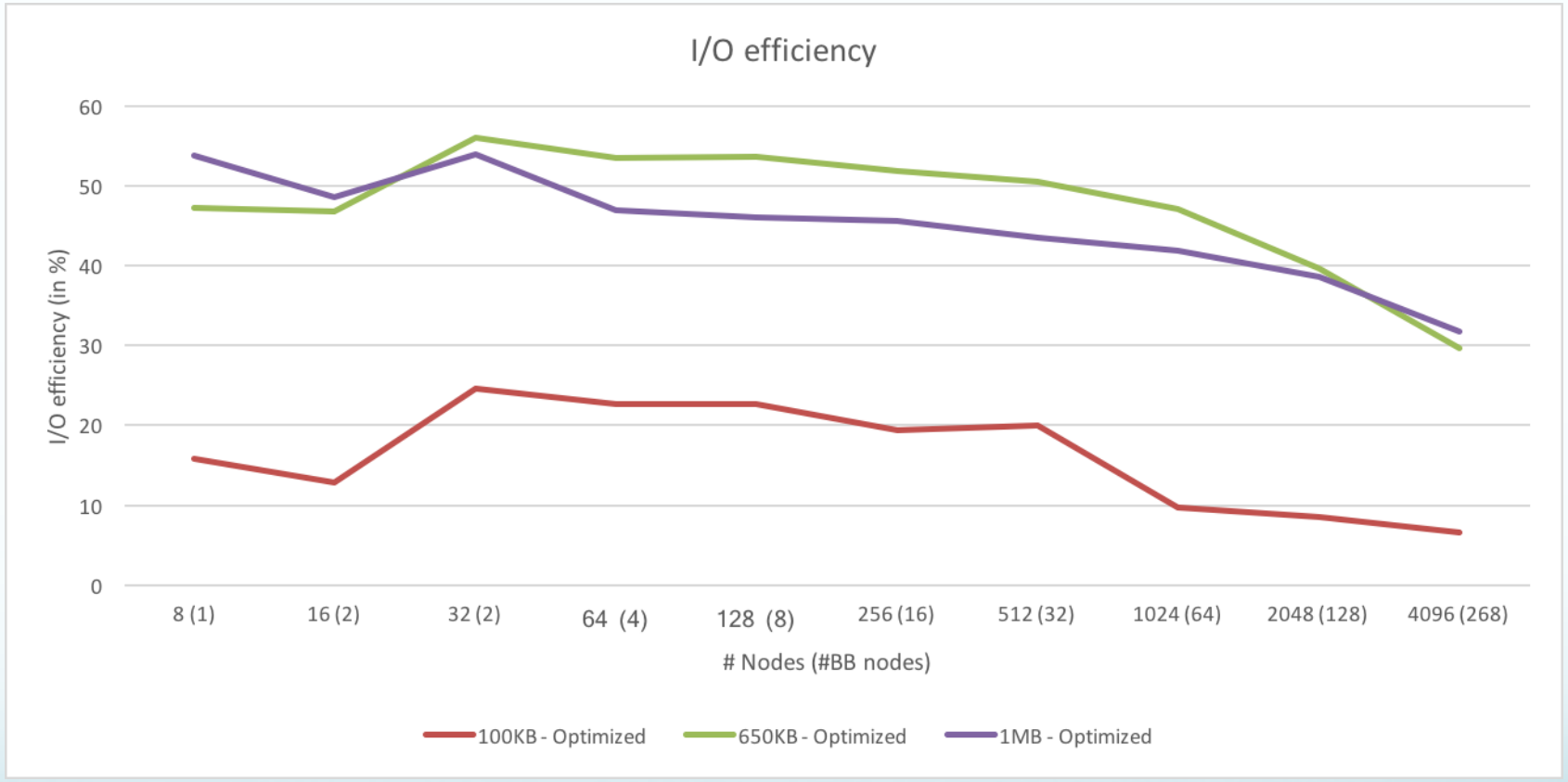
Similar, the I/O efficiency is improved maximum by 3,16 times.

Neuromap – Replib on Cray Burst Buffer



- We save the data in a shared file, each MPI process saves its own data and the size of the output file varies from 9 GB up to 12,5 TB
- We use 1 up to 268 BB nodes
- We achieve up to 0.5 TB/s with 4096 compute nodes and 268 BB nodes.

Neuromap – Replib on Cray Burst Buffer
I/O Efficiency – Optimized parameters



- For the case of chunks of 100KB, the I/O efficiency is between 6,54% and 24,6%
- However, for the cases of 650KB and 1MB, the I/O efficiency varies from 29,7% till 56%

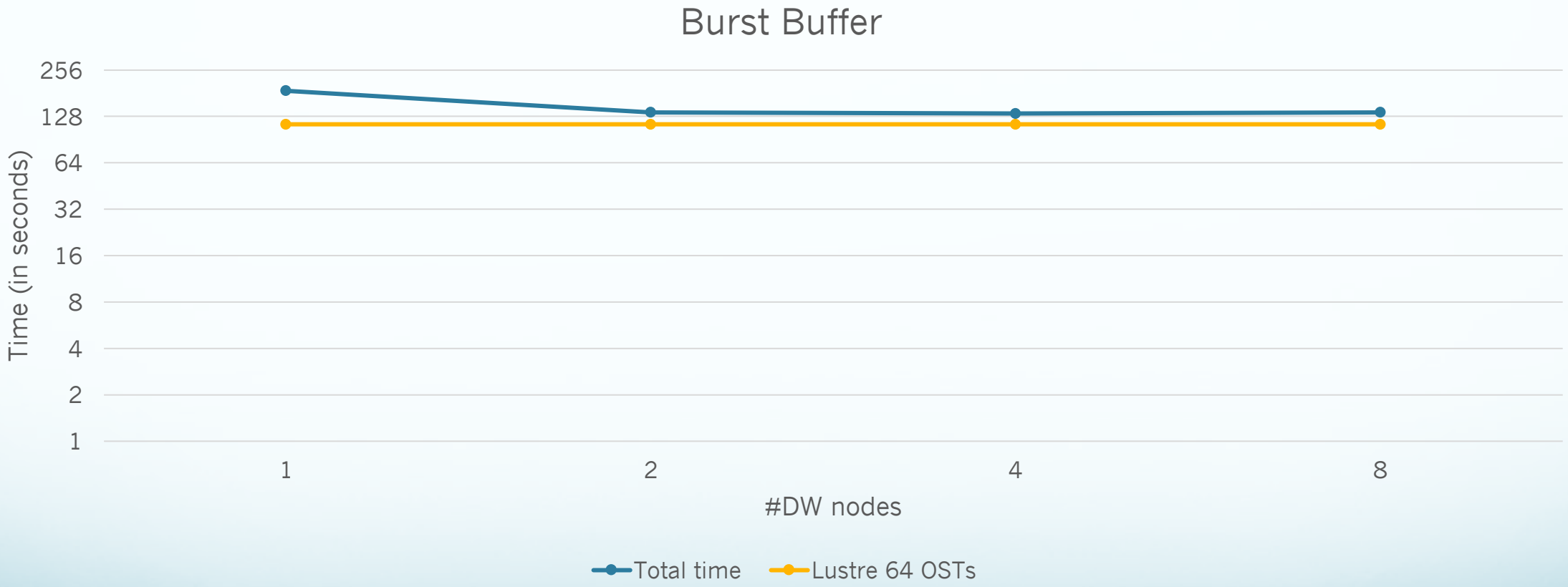
Study-case WRF-CHEM (on Cori)

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

WRF-CHEM on Burst Buffer

- Weather Research and Forecasting Model coupling with Chemistry
- Small domain: 330 x 275
- Size of input file: 804 MB
- Size of output file: 2.9GB, it is saved every one hour of simulation
- Output file quite small
- For all the WRF-CHEM experiments we use 1280 MPI processes (40 nodes), as this is the optimum for the computation/communication
- For the default case, we stage-in all the files and we execute the simulation from BB

Total execution time and I/O on BB without MPICH_MPIIO_HINTS (default)

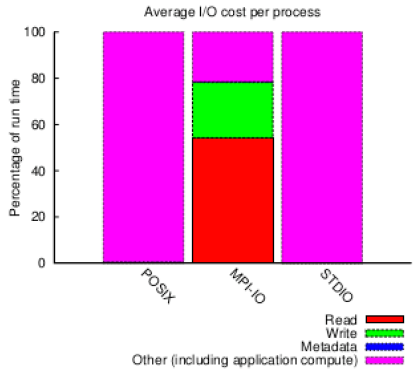


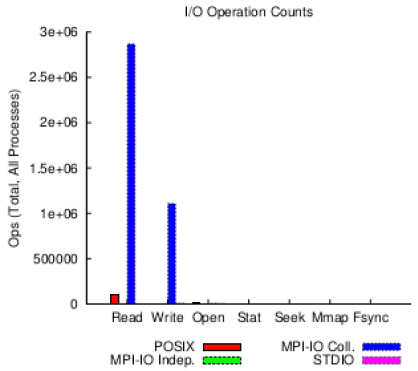
- The best total execution time is provided when we have 4 DW nodes
- On Lustre the best execution time is with 64 OSTs and it is 15% faster than BB

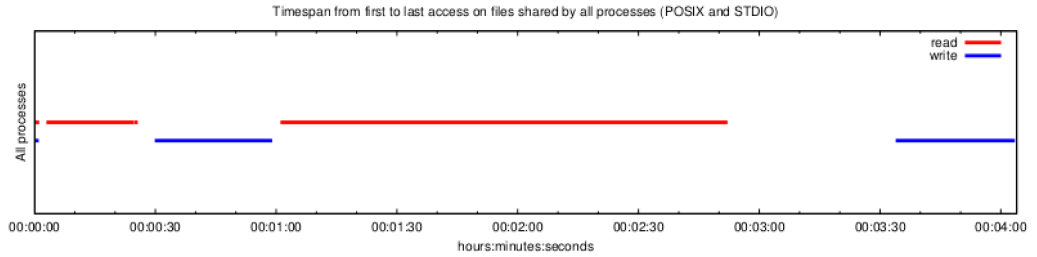
Darshan – WRFChem
1 BB node – default parameters

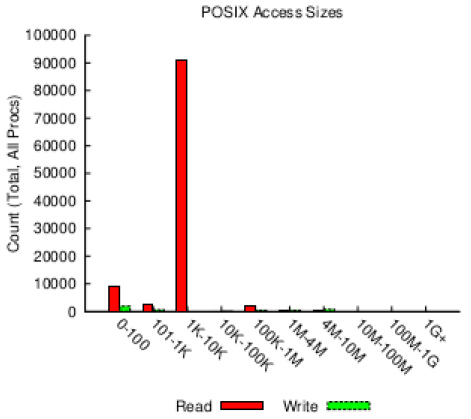
jobid: 6568872	uid: 74747	nprocs: 1280	runtime: 244 seconds

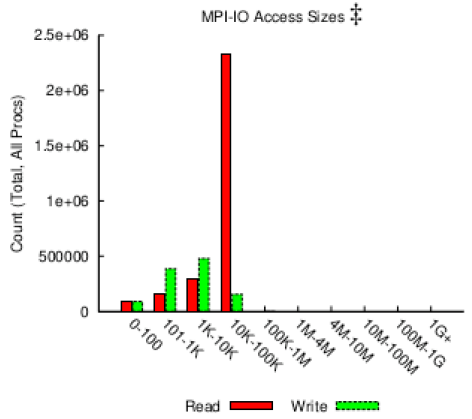
I/O performance *estimate* (at the MPI-IO layer): transferred 182344 MiB at 218.44 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.2 MiB at 1.19 MiB/s











MPI I/O phases Statistics (MPICH_MPIIO_TIMERS=1) I

```
+-------------------------------------------------------------------------+
| MPIIO read by phases, readers only, for wrfinput_d01
|                                  min         max         ave
|                               ----------  ----------  ----------
|   file read          time       =      1.54        1.54        1.54
|   time scale: 1 = 2**6       clock ticks    min         max         ave
|                               ----------  ----------  ---------- ---
|   total                      =                  773580678
|   imbalance                  =    284814      284814      284814        0%
|   local compute              =  91505804    91505804    91505804       11%
|   wait for coll              =   2398813     2398813     2398813        0%
|   collective                 =   3646301     3646301     3646301        0%
|   read/exchange              =  18196022    18196022    18196022        2%
|   file read                  =  55222120    55222120    55222120        7%
|   data receive               =  588775983   588775983   588775983      76%
|   other                      =  12888553    12888553    12888553        1%
|   data receive BW (MiB/s)       =                    0.146
|   raw read  BW (MiB/s)          =                 1819.310
|   net read  BW (MiB/s)          =                  129.872
+-------------------------------------------------------------------------+
Timing for processing wrfinput file (stream 0) for domain        1:   21.68633 elapsed seconds
```

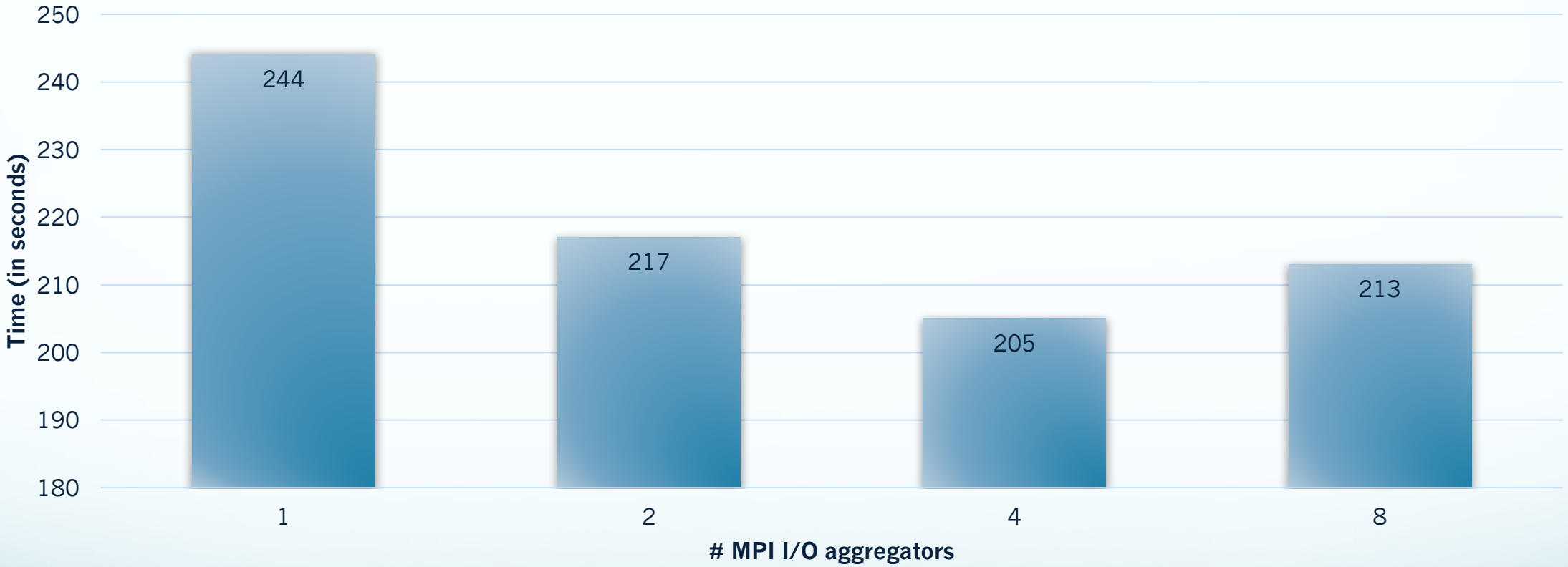
MPI I/O phases Statistics (MPICH_MPIIO_TIMERS=1) II

```
+-------------------------------------------------------------------------+
| MPIIO write by phases, writers only, for wrfout_d01_2007-04-03_01_00_00
|                                 min         max         ave
|                              ----------  ----------  ----------
|   file write        time      =      2.30        2.30        2.30
|   time scale: 1 = 2**7     clock ticks    min         max         ave
|                              ----------  ----------  ---------- ---
|   total                     =                   532124046
|   imbalance                 =    158972      158972      158972        0%
|   local compute             =  48146033    48146033    48146033   9%
|   wait for coll             =    855958      855958      855958        0%
|   collective                =   1589992     1589992     1589992       0%
|   exchange/write            =   9748711     9748711     9748711       1%
|   data send                 =  418919605   418919605   418919605 78%
|   file write                =   41345308    41345308    41345308    7%
|   other                     =   10140527    10140527    10140527    1%
|   data send BW (MiB/s)        =                     0.107
|   raw write BW (MiB/s)        =                  1262.748
|   net write BW (MiB/s)        =                    98.114
+-------------------------------------------------------------------------+
Timing for Writing wrfout_d01_2007-04-03_01_00_00 for domain        1:   30.25151 elapsed seconds
```

MPI I/O phases Statistics (MPICH_MPIIO_TIMERS=1) III

```
+-------------------------------------------------------------------------+
| MPIIO read by phases, readers only, for wrfbdy_d01
|                                  min         max         ave
|                               ----------  ----------  ----------
|   file read         time      =      1.31        1.31        1.31
|   time scale: 1 = 2**8      clock ticks    min         max         ave
|                               ----------  ----------  ---------- ---
|   total                     =                995854066
|   imbalance                 =     349921      349921      349921          0%
|   local compute             =  133146526   133146526   133146526    13%
|   wait for coll             =    1342000     1342000     1342000          0%
|   collective                =    4455424     4455424     4455424          0%
|   read/exchange             =   22374792    22374792    22374792       2%
|   file read                 =   11742536    11742536    11742536       1%
|   data receive              =  803299250   803299250   803299250    80%
|   other                     =   18892054    18892054    18892054       1%
|   data receive BW (MiB/s)       =                    0.210
|   raw read  BW (MiB/s)          =                  271.116
|   net read  BW (MiB/s)          =                    3.197
+-------------------------------------------------------------------------+
Timing for processing lateral boundary for domain        1:  111.10603 elapsed seconds
```

Compare the total execution time on single DW nodes across various MPI I/O aggregators



- Example for declaring 4 MPI I/O aggregators
  export MPICH_MPIIO_HINTS="wrfinput:**cb_nodes=4**,wrfout*:**cb_nodes=4,***
  wrfb:**cb_nodes=4**"*
- Tip: You can declare different MPI I/O aggregators per file

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) I

```
+-----------------------------------------------------+
| MPIIO read access patterns for wrfinput_d01
|   independent reads       = 1
|   collective reads        = 527360
|   independent readers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 8388608
|   system reads            = 762
|   stripe sized reads      = 108
|   total bytes for reads   = 2930104643 = 2794 MiB = 2 GiB
|   ave system read size    = 3845281
|   number of read gaps     = 2
|   ave read gap size       = 0
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+-----------------------------------------------------+
```

We have 4 MPI I/O aggregators
We use one BB node (stripe count)
Default stripe size 8 MB
Only 14.17% of the reads are striped (100*108/762)

The average system read size is less than 4MB,
the stripe size should be close to the average system read size

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) II

```
+-----------------------------------------------------+
| MPIIO write access patterns for wrfout_d01_2007-04-03_00_00_00
|   independent writes      = 2
|   collective writes       = 552960
|   independent writers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 8388608
|   system writes           = 797
|   stripe sized writes     = 114
|   aggregators active      = 234240,0,0,318720 (1, <= 1, > 1, 2)
|   total bytes for writes  = 3045341799 = 2904 MiB = 2 GiB
|   ave system write size   = 3821006
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 2
|   ave write gap size      = 4194300
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+-----------------------------------------------------+
```

Similar 14.3% of the writes are striped (100*114/797)

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) III

```
+-----------------------------------------------------+
| MPIIO read access patterns for wrfbdy_d01
|   independent reads       = 2
|   collective reads        = 2338560
|   independent readers     = 1
|   aggregators             = 2
|   stripe count            = 1
|   stripe size             = 8388608
|   system reads            = 1876
|   stripe sized reads      = 0
|   total bytes for reads   = 371398962 = 354 MiB
|   ave system read size    = 197973
|   number of read gaps     = 6
|   ave read gap size       = 0
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+-----------------------------------------------------+
```

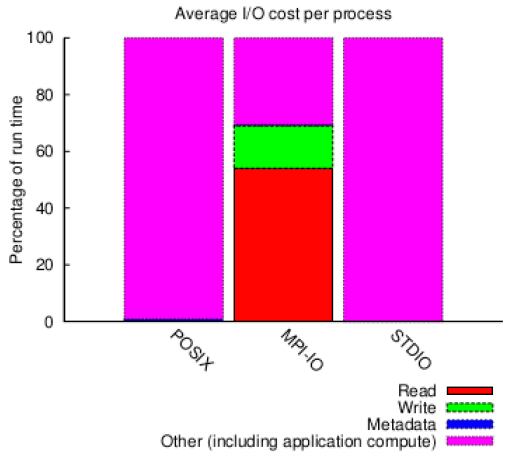
All the reads are not striped which mean this I/O is not efficient.
The average system read size is 197973 bytes

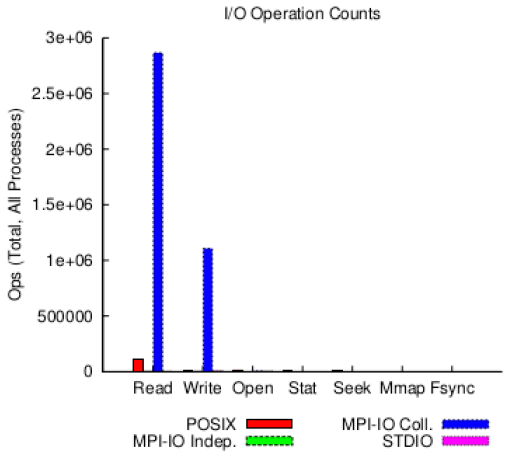
Declaring MPICH MPIIO HINTS parameters based on the previous data

MPICH_MPIIO_HINTS="wrfinput*:cb_nodes=4:striping_unit=2097152,
wrfout*:cb_nodes=4:striping_unit=2097152,
wrfb*:cb_nodes=4:striping_unit=197973"

jobid: 6575384	uid: 74747	nprocs: 1280	runtime: 172 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 182112 MiB at 350.40 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.2 MiB at 1.34 MiB/s





The execution time was decreased by almost 30%

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) IV

```
+-----------------------------------------------------+
| MPIIO read access patterns for wrfinput_d01
|   independent reads       = 1
|   collective reads        = 527360
|   independent readers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 2097152
|   system reads            = 1810
|   stripe sized reads      = 1141
|   total bytes for reads   = 2930104643 = 2794 MiB = 2 GiB
|   ave system read size    = 1618842
|   number of read gaps     = 2
|   ave read gap size       = 0
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+-----------------------------------------------------+

Timing for processing wrfinput file (stream 0) for domain        1:    9.56521 elapsed seconds
```

We have 4 MPI I/O aggregators
We use one BB node (stripe count)
New stripe size 2 MB
Only 63% of the reads are striped
The number of the operations increase (1810 reads)

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) V

```
+--------------------------------------------------------+
| MPIIO write access patterns for wrfout_d01_2007-04-03_00_00_00
|   independent writes      = 2
|   collective writes       = 552960
|   independent writers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 2097152
|   system writes           = 1886
|   stripe sized writes     = 1183
|   aggregators active      = 208640,33280,0,311040 (1, <= 2, > 2, 4)
|   total bytes for writes  = 3045341799 = 2904 MiB = 2 GiB
|   ave system write size   = 1614709
|   read-modify-write count = 0
|   read-modify-write bytes = 0
|   number of write gaps    = 2
|   ave write gap size      = 1048572
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+--------------------------------------------------------+
Timing for Writing wrfout_d01_2007-04-03_00_00_00 for domain        1:    12.99924 elapsed seconds
```

62.7% of the writes are striped

Understand the MPI I/O statistics on BB (MPICH_MPIIO_STATS=1) VI

```
+-----------------------------------------------------+
| MPIIO read access patterns for wrfbdy_d01
|   independent reads       = 2
|   collective reads        = 2338560
|   independent readers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 197973
|   system reads            = 3705
|   stripe sized reads      = 114
|   total bytes for reads   = 371398962 = 354 MiB
|   ave system read size    = 100242
|   number of read gaps     = 5
|   ave read gap size       = 444575572
| See "Optimizing MPI I/O on Cray XE Systems" S-0013-20 for explanations.
+-----------------------------------------------------+
```

3% of the reads are striped

Timing for processing lateral boundary for domain 1: 83.90572 elapsed seconds

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

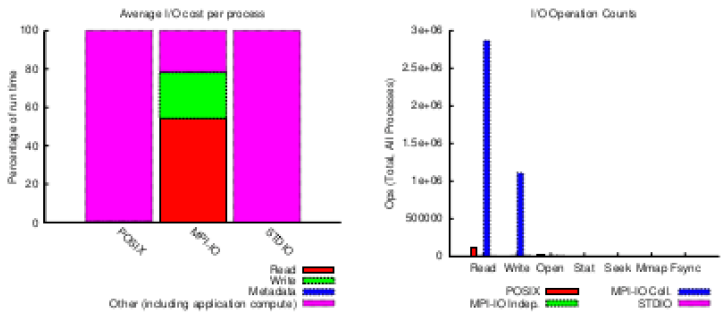
Looking for the optimum parameters

- We executed more experiments and tested various parameters according to the MPI IO statistics data.
- If the performance does not increase while we decrease the value of the striping unit, increase the number of the MPI I/O aggregators.
- While we decrease the value of the striping unit, the number of reads/writes is increasing. Maybe there is a need to use more BB nodes to achieve better performance.

WRF-CHEM – Final results

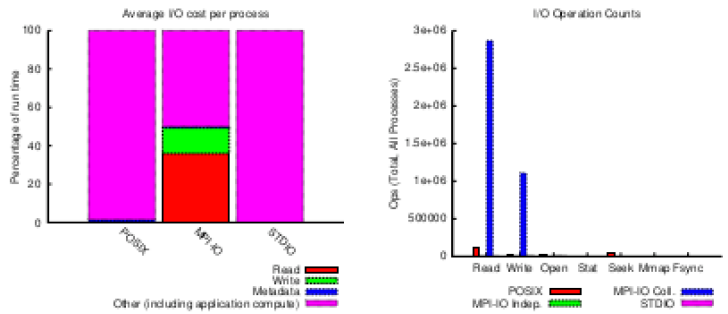
jobid: 6568872	uid: 74747	nprocs: 1280	runtime: 244 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 182344 MiB at 218.44 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.2 MiB at 1.19 MiB/s



jobid: 6575835	uid: 74747	nprocs: 1280	runtime: 105 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 182756 MiB at 794.25 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.2 MiB at 1.38 MiB/s



The execution time was decreased by 57% on just on BB node!

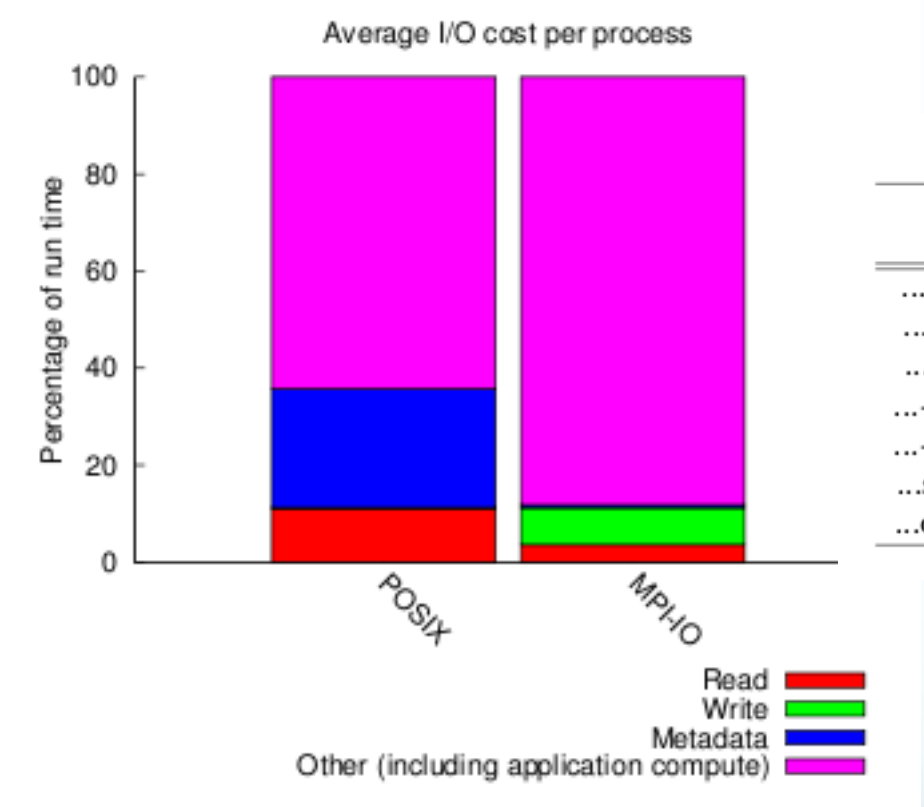
Optimum parameters
MPICH_MPIIO_HINTS="wrfinput*:cb_nodes=16:striping_unit=262144,\
wrfout*:cb_nodes=16:striping_unit=262144,\
wrfb*:cb_nodes=16:striping_unit=50482"

Studying MPI I/O aggregators and striping size

Parameters	I/O duration for wrfinput (in sec.)	I/O duration for wrfout (in sec.)	I/O duration for wrbdy_d01 (in sec.)
Default	21,68	30,25	111,10
Optimized	5,51	7,27	32,8

The I/O bandwidth was improved between 3.4 and 4.1 times

What happens if we try to scale on BB with this example?



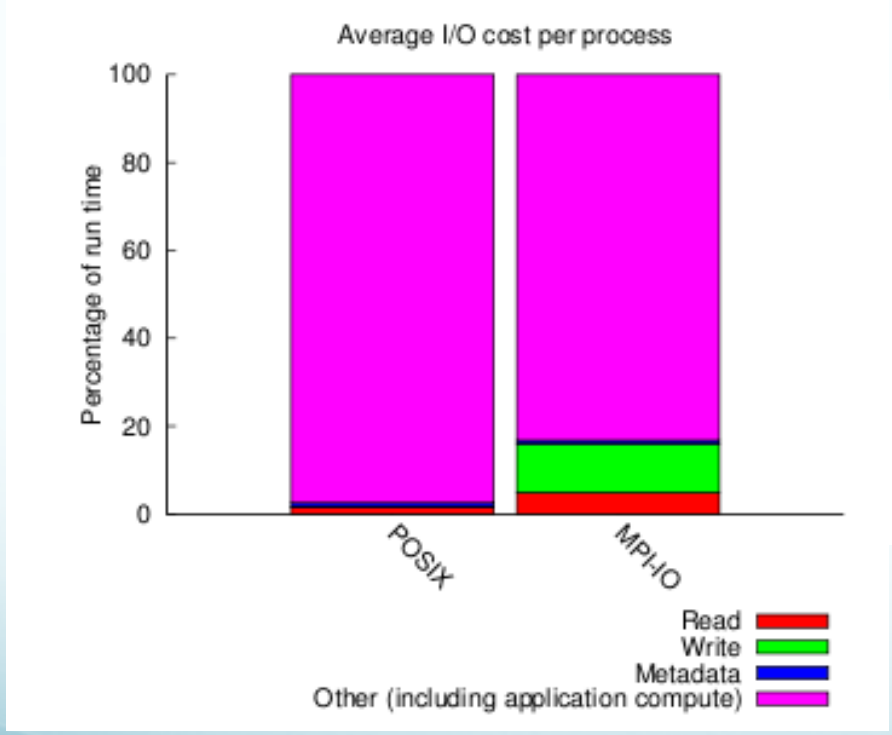
Variance in Shared Files

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
.../namelist.input	1280	572	20.092293	22K	0	41.251590	215K	4.9	5.51e+03
..._plev.formatted	1280	358	1.260848	708	255	16.833767	708	1.75	0
...e_lat.formatted	1280	750	0.023262	536	1258	9.408035	536	2.34	0
...-04-03_01_00_00	1280	1019	6.188566	0	0	6.809746	46M	0.331	1.04e+07
...-04-03_00_00_00	1280	1189	6.245521	0	0	6.275130	46M	0.331	1.04e+07
...ss/wrfinput_d01	1280	4	5.919294	0	1279	6.212909	0	0.327	6.06e+06
...ozone.formatted	1280	1107	0.016222	531K	459	1.896104	531K	0.565	0

For this experiment, we use 64 DW nodes, the file namelist.input is 12KB (Darshan 2.x reports wrong file size) but 1280 MPI processes access it for reading and there is significant metadata issue.

Solution

- Use private access mode
- Hybrid execution, we save small files on Lustre and large ones on BB



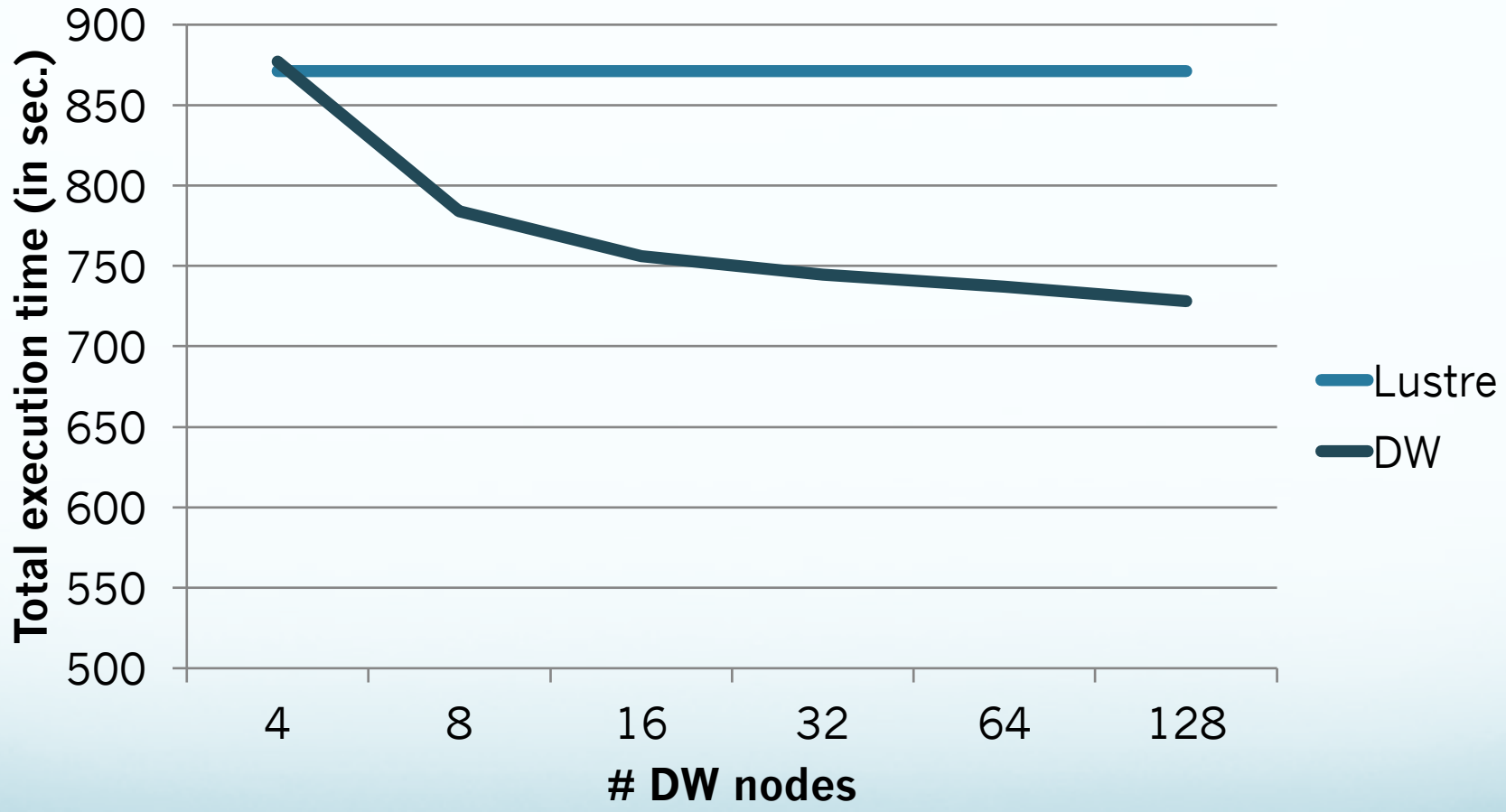
Variance in Shared Files

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
...-04-03_01_00_00	1280	1019	6.508006	0	0	7.171755	46M	0.334	1.04e+07
...st/wrfinput_d01	1280	4	4.773887	0	1279	6.874528	0	0.0271	6.43e+06
...-04-03_00_00_00	1280	307	6.485894	0	0	6.540200	46M	0.348	1.04e+07
.../namelist.input	1280	778	1.059317	22K	972	2.180347	22K	0.258	5.53e+03
...ozone.formatted	1280	105	0.013726	531K	1133	0.687412	531K	0.163	0
..._plev.formatted	1280	583	0.031235	708	932	0.585686	708	0.126	0
...e_lat.formatted	1280	603	0.005603	536	262	0.549975	536	0.142	0

Tip: You need to declare the path of the files in the MPICH_MPIIO_HINTS
export MPICH_MPIIO_HINTS="**$DW_JOB_STRIPED/wrfinput***:cb_nodes=40 ...
$DW_JOB_STRIPED/wrfout*:cb_nodes=40..."

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

WRF – Lustre vs DW



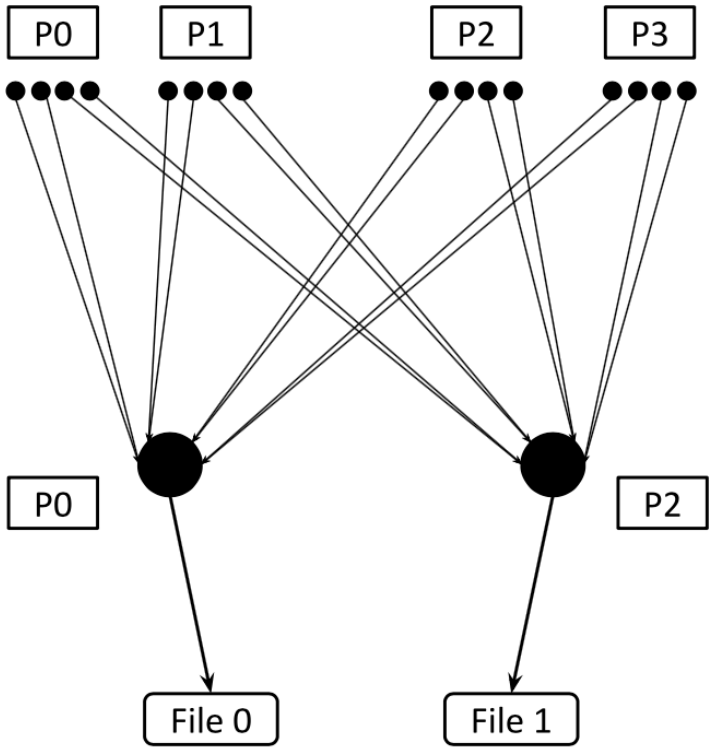
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

PIDX

- PIDX is an efficient parallel I/O library that reads and writes multiresolution IDX data files
- It can provide high scalability up to 768k cores
- Successful integration with several simulation codes
  - KARFS (KAUST Adaptive Reacting Flow Solvers) on Shaheen II
  - Uintah with production runs on Mira
  - S3D

https://www.sci.utah.edu/software/pidx.html

PIDX description



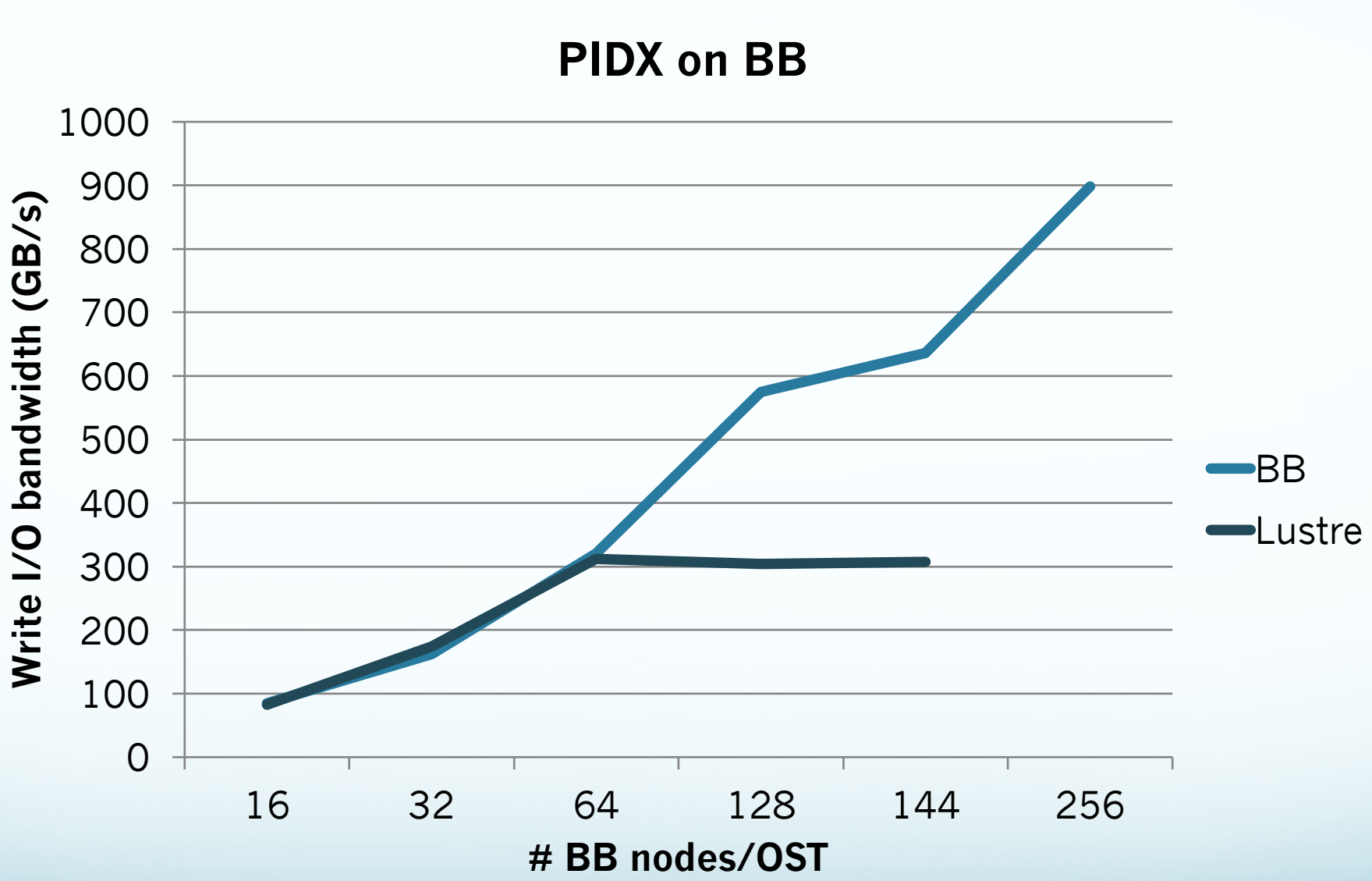
Synchronous reducing mode of communication using one-sided MPI

To balance the data movement, the aggregators are uniformly placed across the ranks.

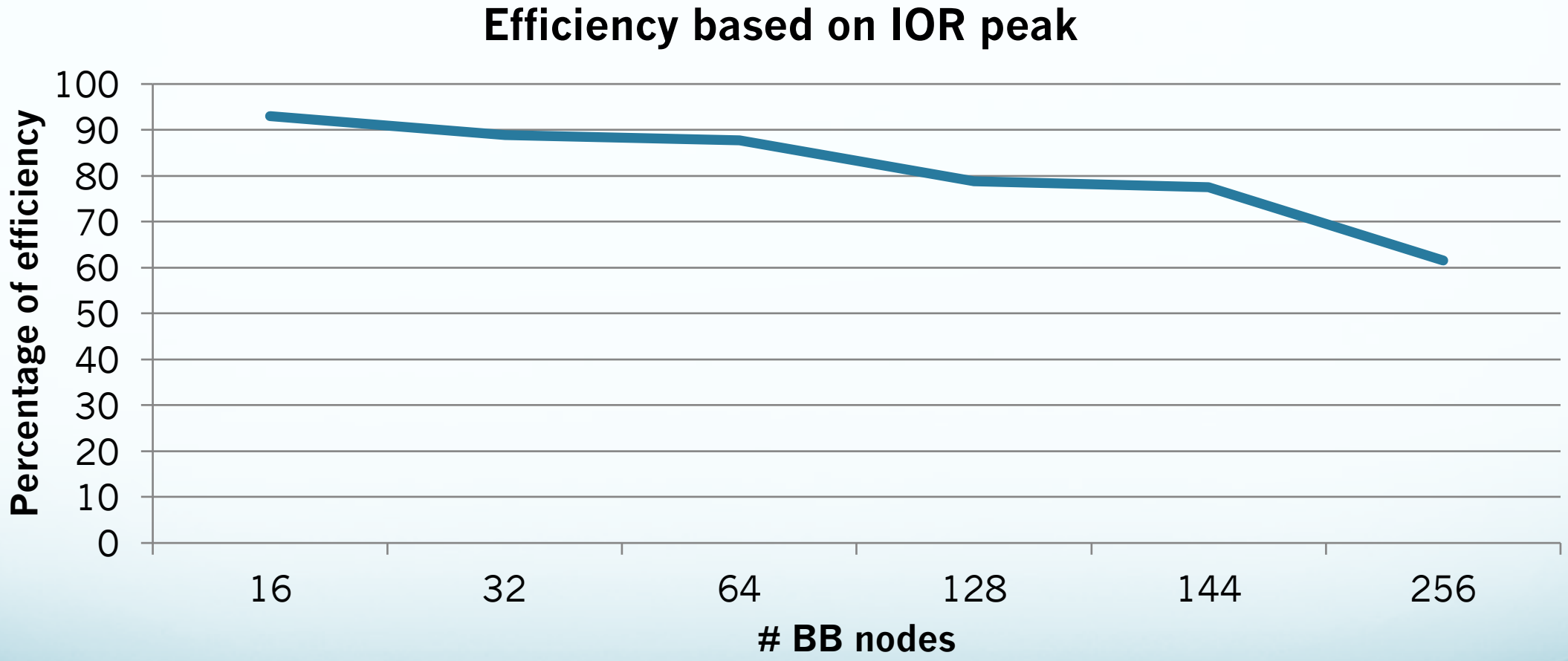
File locking contention is reduced by having an aggregator to write to only one file

File count are tunable (based on the file system in use).

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology



The library achieves up to 900 GB/s on Burst Buffer, while we save 64 MB (2x32) per MPI process


Efficiency based on IOR peak
Percentage of efficiency
100
90
80
70
60
50
40
30
20
10
0
16
32
64
128
144
256
BB nodes

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

SeisSol I

- SeisSol is a software package for simulating wave propagation and dynamic rupture based on the arbitrary high-order accurate derivative discontinuous Galerkin method
- Using 128 DataWarp nodes with 256 compute nodes. Developer provided an I/O kernel benchmark called checkpoint and it is available in the corresponding github repository.
- Many back-ends to be tests, MPI I/O, POSIX, HDF5, the SIONLIB had some issues.

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

SeisSol II

The developers have already integrated many advanced parameters such as:

SEISSOL_CHECKPOINT_ALIGNMENT=8388608
SEISSOL_CHECKPOINT_BLOCK_SIZE=8388608
SEISSOL_CHECKPOINT_SION_BACKEND=ansi
SEISSOL_CHECKPOINT_SION_NUM_FILES=1
SEISSOL_CHECKPOINT_SION_COLL_SIZE=0
SEISSOL_CHECKPOINT_CB_NODES=256
SEISSOL_CHECKPOINT_ROMIO_CB_WRITE=disable
SEISSOL_CHECKPOINT_ROMIO_DS_WRITE=disable
SEISSOL_CHECKPOINT_MPIO_LARGE_BUFFER=0

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

SeisSol Results

Filesystem	Back-end	I/O write performance (GB/s)
Lustre	MPI I/O	100
DataWarp	MPI I/O	472
DataWarp	POSIX	503
DataWarp	HDF5	449

In this case, DataWarp is 4.72 times faster than Lustre and around to 60% I/O efficiency

Complex Workflows

Case 1: WRF-CHEM

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Outline

- Motivation
- In-depth explanation
- Demo - video

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Motivation

- Using compute resources, while producing wrong results, costs time and money (even in electricity)
- Spending core-hours from team project
- You are not sure if the simulation has any issue

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Study case – WRF-CHEM

- This is a real case of a ShaheenII user at KAUST.
- 40 compute nodes are used
- Around to 3GB of data are saved for specific time-steps.

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Methodology

- First, we declare the required Burst Buffer (BB) space in persistent mode (**create_persistent.sh**).
- Then we start the execution of the model, using the BB persistent space
- Then we start the execution of the tool **plot_and_stage_out.sh** that does the following:
  - Check the existence of any output file (we know the filename pattern)
  - When an output file exists (NetCDF format), we use a script in Python with NetCDF and Matplotlib libraries to read the output file and save one variable to an image file (with same filename pattern)
  - Then a tool which uses DataWarp API, stages out **only** the image into the Lustre parallel filesystem.
- The same moment with the plot_stage_out.sh, we execute the **wait.sh** script which runs on the login node. This script recognizes when an image has been stage-out and it visualizes it for the user. Then, the user observes if the simulation is correct or not and can stop the simulation if it is required.

Instructions here:
https://github.com/KAUST-KSL/SC17_BB_Tutorial
folder: complex_workflow/persistent_vis

Creating Persistent BB allocation

- File: **create_persistent.sh**
- Execution: sbatch create_persistent.sh

```
#!/bin/bash –x
#SBATCH --partition=workq
#SBATCH -t 1
#SBATCH -A k01
#SBATCH --nodes=1
#SBATCH -J create_persistent_space

#BB create_persistent name=george_test capacity=600G access=striped
type=scratch
exit 0
```

Executing the main application I

- File: **wrfchem_bb_persistent.sh**
- Execution: sbatch wrfchem_bb_persistent.sh (check the job id)

```
#SBATCH --partition=workq
#SBATCH -t 60
#SBATCH -A k01
#SBATCH  --ntasks=1280
#SBATCH --ntasks-per-node=32
#SBATCH   -J WRF_CHEM_PERSISTENT
#SBATCH -o out_%j
#SBATCH -e err_%j

#DW persistentdw name=george_test
#DW stage_in type=directory
source=/project/k01/.../forburst  destination=$DW_PERSISTENT_STRIPED_george_test

export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY=1
export MPICH_MPIIO_HINTS_DISPLAY=1
export MPICH_STATS_DISPLAY=1
```

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Executing the main application II

```
export MPICH_MPIIO_HINTS="$DW_PERSISTENT_STRIPED_george_test/
wrfinput*:cb_nodes=40:striping_unit=131072,
$DW_PERSISTENT_STRIPED_george_test/wrfout*:cb_nodes=40:striping_unit=65536”
export MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1
export MPICH_MPIIO_STATS=2

cd $DW_PERSISTENT_STRIPED_george_test
chmod +x wrf.exe

time srun -n 1280 --hint=nomultithread wrf.exe
```

Create an image of the output NetCDF file

- File: **plot_persistent.sh**
- Execute: ./plot_persistent.sh filename_netcdf

```
#!/.../python
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import netCDF4
import sys
nc = netCDF4.Dataset(str(sys.argv[1]))

# read all the data
topo = nc.variables['T2'][::1,::1]

# make image
plt.figure(figsize=(10,10))
plt.imshow(topo.squeeze(),origin='lower')

#plt.title(nc.title)
output=str(sys.argv[1])+'.png'
plt.savefig(output, bbox_inches=0)
```

Stage out using DataWarp API

- File: **stage_out.c**
- Compile:
  - module load datawarp
  - cc –o stage_out stage_out.c

```
#include <stdio.h>
#include <datawarp.h>

int main(int argc, char **argv)
{    char *infile, *outfile;
    int stage_out;
    infile = argv[1];
    outfile = argv[2];
    stage_out = dw_stage_file_out(infile, outfile, DW_STAGE_IMMEDIATE);
    return 0;
}
```

- Execute: srun -n 1 stage_out $DW_PERSISTENT_STRIPED_george_test/filename.png /project/k01/markomg/wrfchem_stage_out/filename.png

Script to plot and stage out I

- File: **plot_stage_out.sh**
- Execute: sbatch --dependency=after:app_job_id plot_stage_out.sh

```
#!/bin/bash

#SBATCH --partition=workq
#SBATCH -t 30
#SBATCH -A k01
#SBATCH  --ntasks=32
#SBATCH --ntasks-per-node=32
#SBATCH   -J PLOT_AND_STAGE_OUT
#SBATCH -o out_%j
#SBATCH -e err_%j

#DW persistentdw name=george_test
#DW stage_in type=directory
source=/project/k01/markomg/burstbuffer/complex/stage_in_bb/  destination=$DW_PERSIST
ENT_STRIPED_george_test
```

Script to plot and stage out II

```
module load python/2.7.11
cd $DW_PERSISTENT_STRIPED_george_test
chmod +x plot_persistent.sh
chmod +x stage_out

let i=0
while [ $i -lt 24 ]
do
k=$(printf %02d $i)

if [ -f wrfout_d01_2007-04-03_${k}_00_00 ]; then
    check_lsof=`lsof wrfout_d01_2007-04-03_${k}_00_00 | wc -l`
        while [ $check_lsof -eq 2 ]
        do
                sleep 30
            check_lsof=`lsof wrfout_d01_2007-04-03_${k}_00_00 | wc -l`
        done
    ./plot_persistent.sh wrfout_d01_2007-04-03_${k}_00_00
        srun -n 1 stage_out $DW_PERSISTENT_STRIPED_george_test/wrfout_d01_2007-04-03_${k}_00_00.png
/project/k01/markomg/wrfchem_stage_out/wrfout_d01_2007-04-03_${k}_00_00.png
        let i=$i+1
else
        sleep 30
fi
done
```

Visualize images when they arrive on the Lustre

- File: **wait.sh**
- Execute: ./wait.sh number_of_images /path_to_Lustre_stage_out_folder/

```
#!/bin/bash

let i=0
while [ $i -lt $1 ]
do
        if [ -f $2/wrfout_d01_2007-04-03_$(printf "%02d" $i)_00_00.png ]; then
                  display  $2/wrfout_d01_2007-04-03_$(printf "%02d" $i)_00_00.png &
                let i=i+1
                sleep 15
        else
                sleep 60
        fi
done
```

Delete Persistent BB allocation

- File: **delete_persistent.sh**
- Execution: sbatch delete_persistent.sh

```
#!/bin/bash
#SBATCH --partition=workq
#SBATCH -t 1
#SBATCH -A k01
#SBATCH --nodes=1
#SBATCH  -J delete_persistent_space

#BB destroy_persistent name=george_test
exit 0
```

markomg@cdl4:

Case 2: In situ processing and visualization

(collaboration with KVL)

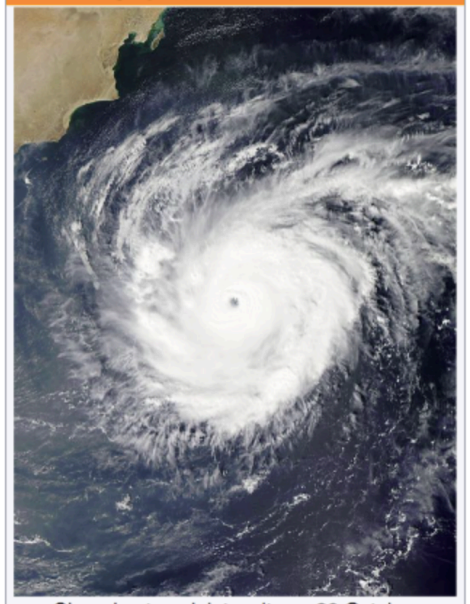
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Cyclone Chapala

Extremely Severe Cyclonic Storm Chapala was the second strongest tropical cyclone on record in the Arabian Sea, according to the American-based Joint Typhoon Warning Center (JTWC). The third named storm of the 2015 North Indian Ocean cyclone season, it developed on 28 October off western India from the monsoon trough. Fueled by record warm water temperatures, the system quickly intensified and was named *Chapala* by the India Meteorological Department (IMD). By 30 October, the storm developed an eye in the center of a well-defined circular area of deep convection. That day, the IMD estimated peak three-minute sustained winds of 215 km/h (130 mph), and the JTWC estimated one-minute winds of 240 km/h (150 mph); only Cyclone Gonu in 2007 was stronger in the Arabian Sea.

Extremely Severe Cyclonic Storm Chapala

Extremely severe cyclonic storm (IMD scale)
Category 4 (Saffir–Simpson scale)



Chapala at peak intensity on 30 October

Formed	28 October 2015
Dissipated	4 November 2015
Highest winds	*3-minute sustained:* 215 km/h (130 mph) *1-minute sustained:* 240 km/h (150 mph)
Lowest pressure	940 hPa (mbar); 27.76 inHg
Fatalities	9 confirmed
Damage	Unknown
Areas affected	Oman, Somalia, Yemen

Part of the **2015 North Indian Ocean cyclone season**

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Description

- We execute Inshimtu and WRF on the same nodes (Inshimtu uses only the last core), one extra node for the post-process
- When a NetCDF file is written, then it is converted to VTK format but only the area that we are interested in, so we save less data
- In our largest case, by removing variables that we do not need and chopping specific area, from 28.2TB of NetCDF files, we save on Lustre 97GB
- Files are downloaded and visualized

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Results

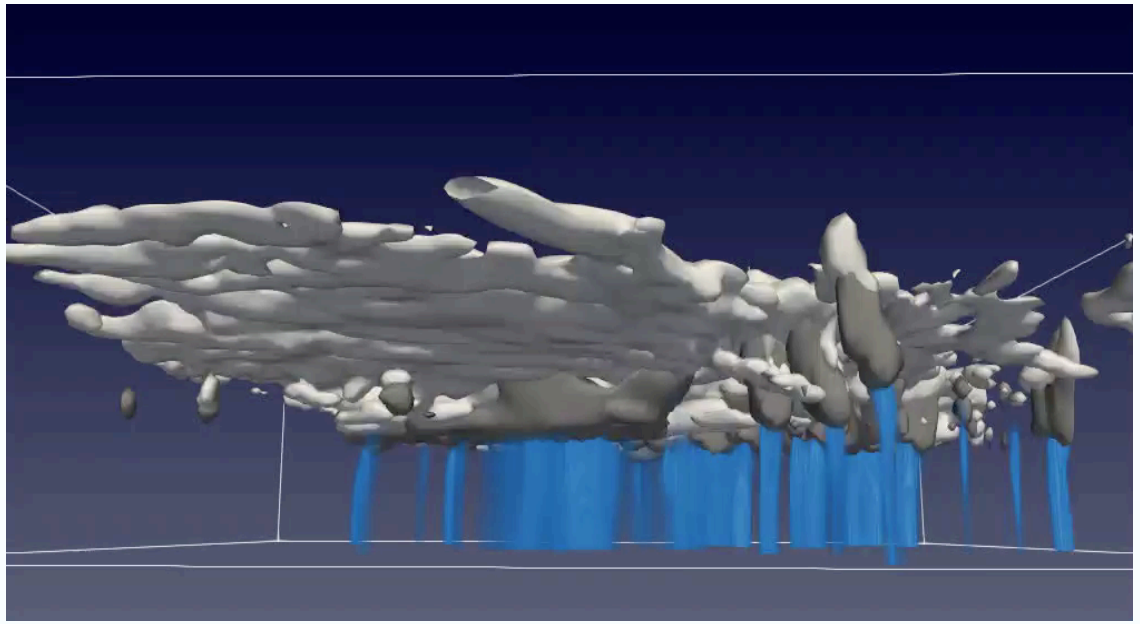
- We use two domains, one small (1100x1000x34) and one larger (3500x3000x34).
- In order to increase the details in the available data, we are testing two cases, saving data every one hour and every 10 minutes.
- A post-processing tool chops from the whole area only the cyclone region and saves this file on BB.

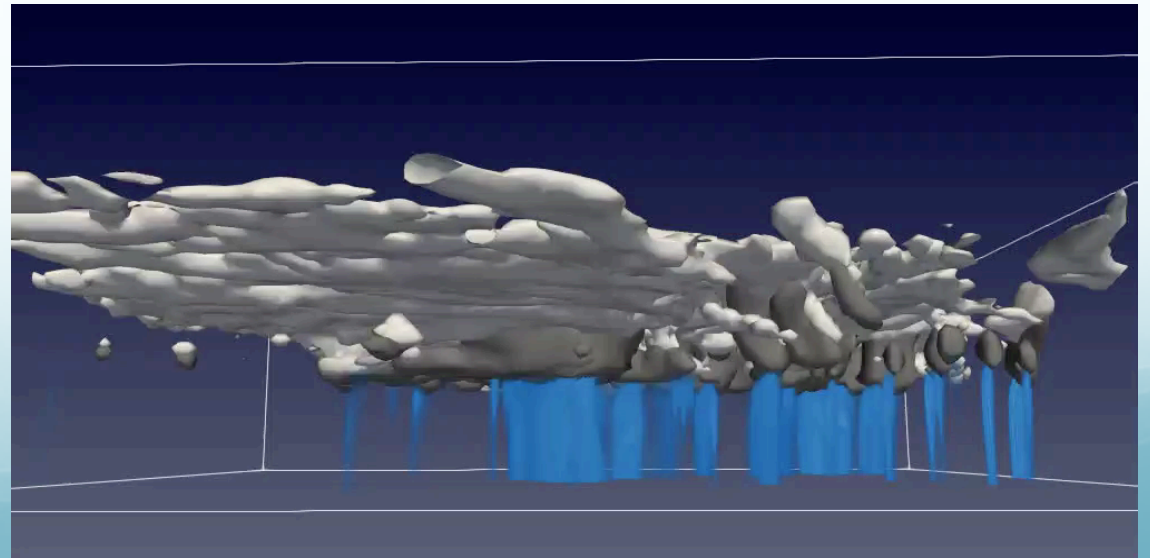
Videos here:
https://github.com/KAUST-KSL/SC17_BB_Tutorial
folder: complex_workflow/cyclone_vis

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Visualization

- Executing the simulation on Burst Buffer and save data every 10 minutes with manual tuning (**6x times more data**). Total execution time is 5% faster than Lustre.





جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Conclusions

- Using Burst Buffer is not difficult but achieving significant performance requires some effort.
- Burst Buffer boosts the performance for many demonstrated applications
- Many parameters need be investigated for the optimum performance
- Be careful to compile your application with the appropriate Cray MPICH version
- CLE 6.0 solves some BB issues but still needs optimizations
- Implementing a complex workflow has several steps and it could combine persistent allocation, multiple applications having access to same files, external scripts to handle same files, and DataWarp API

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Evaluation

http://bit.ly/sc17-eval

Burst Buffer and NVMe Virtual Organization Registration

- Join the virtual organization and let all contribute
- https://goo.gl/XyeDRo

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Thank you!
Questions?

georgios.markomanolis@kaust.edu.sa