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

Burst Buffer: From Alpha to Omega

Intel Extreme Performance Users Group Middle East Conference 2018

25 April 2018

George Markomanolis
Computational Scientist
KAUST Supercomputing Laboratory
georgios.markomanolis@kaust.edu.sa

Saber Feki
Computational Scientist Team Lead
KAUST Supercomputing Laboratory
saber.feki@kaust.edu.sa

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

Outline

- Introduction to Parallel I/O
- Understanding the I/O performance on Lustre
- Introduction to Burst Buffer
- Accelerating the performance

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 (SSD) 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)

Software

- **Application Software**
  - Weather & Environment: WRF, WRF-Chem, HIRAM, MITgcm
  - Big Data: Mizan (in-house)
  - Biology & MD: Amber, Gromacs, LAMMPS, NAMD, VEP, BLAST, Infernal
  - Combustion: NGA, S3D, KARFS
  - CFD & Plasma: Ansys, Fluent, OpenFOAM, Plasmoid (in-house)
  - Chemistry & Materials Science: VASP, Materials Studio, Gaussian, WEIN2k, Quantum Espresso, ADF, CP2K
  - Electromagnetism: Ansys, In-house developed code
  - Oil & Gas: Madagascar, sofi2D, sofi3D, In-house developed codes
  - Seismology: SORD, SeisSol, SPECFEM_3D_GLOBE

- **Development Tool**
  - Compiler: Cray, Intel and GNU with MPICH library
  - Optimized Math Library: Cray-libsci, Intel-MKL, PETSc, FFTW, ParMetis
  - I/O library: HDF5, NetCDF, PNetCDF, ADIOS
  - Performance tools: CrayPat, Reveal, Extrae, Allinea Map
  - Debugger: Totalview, Allinea DDT

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

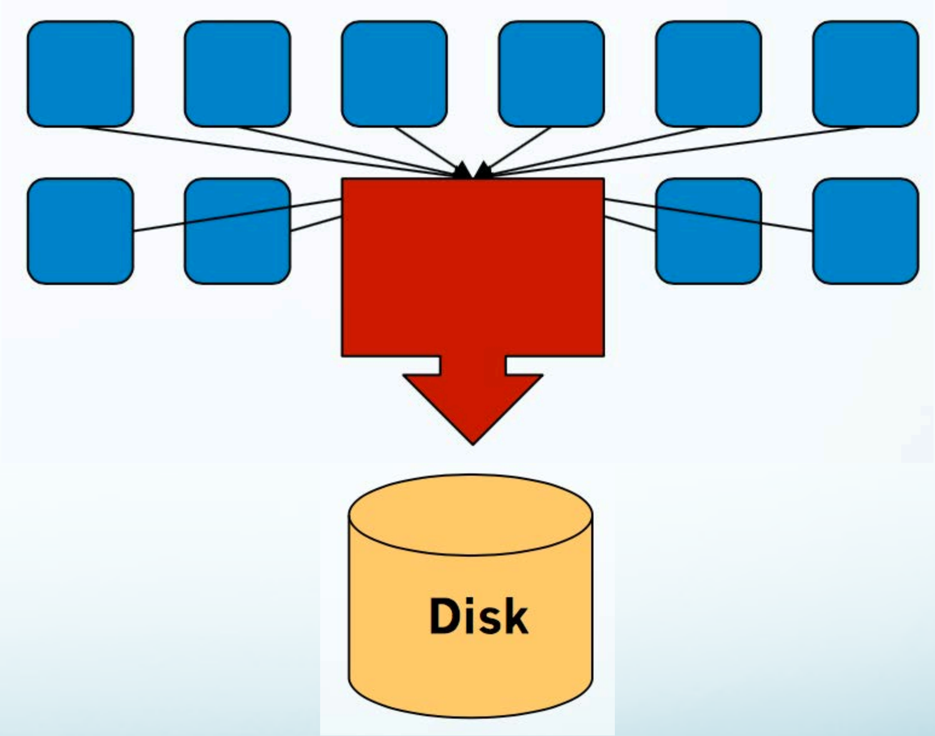
- Parallel I/O is needed to
  - Do more science than waiting files to be read/written
  - No waste of resources
  - Not stressing the file system, thus affecting other users

I/O Performance

- There is no one 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

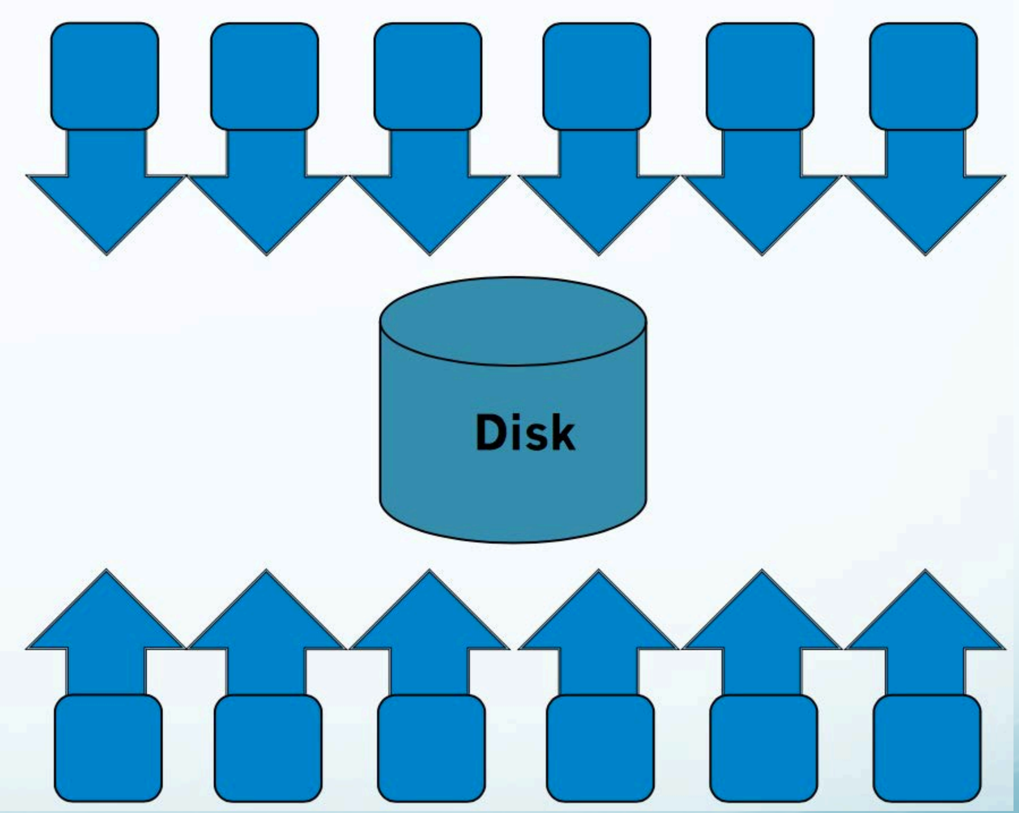
Serial I/O

- 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



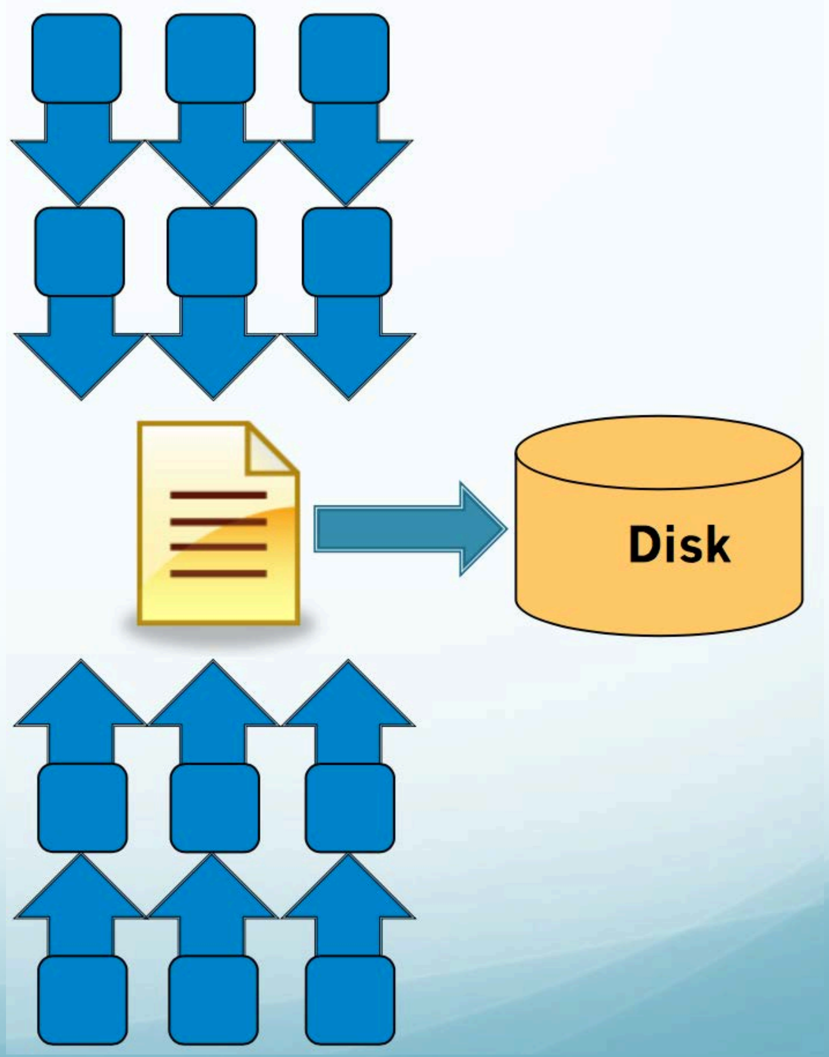
Parallel I/O: File-per-Process

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



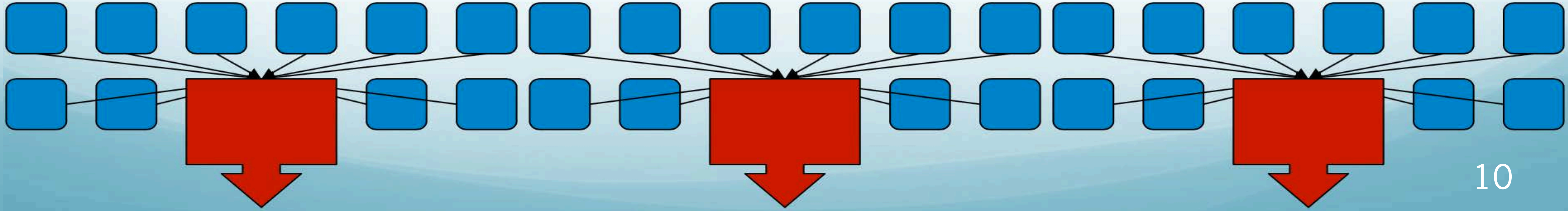
Parallel I/O: Shared File

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



Pattern Combinations

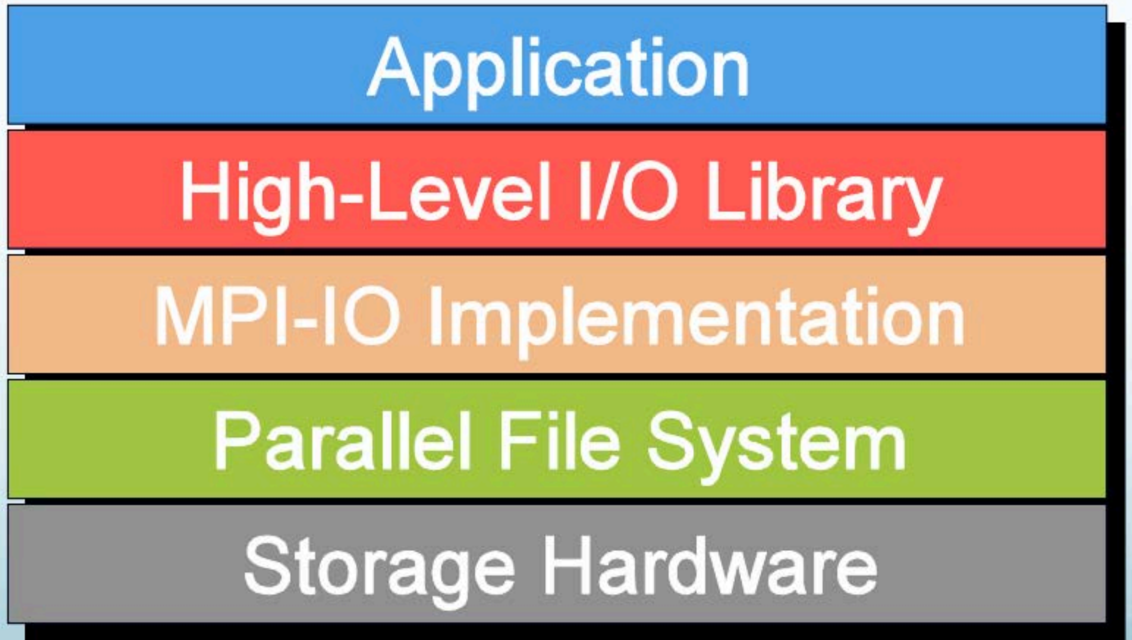
- Subset of processes perform I/O
  - **Aggregation** of a group of processes data
  - I/O process may access independent files
  - Group of processes perform parallel I/O to a shared file



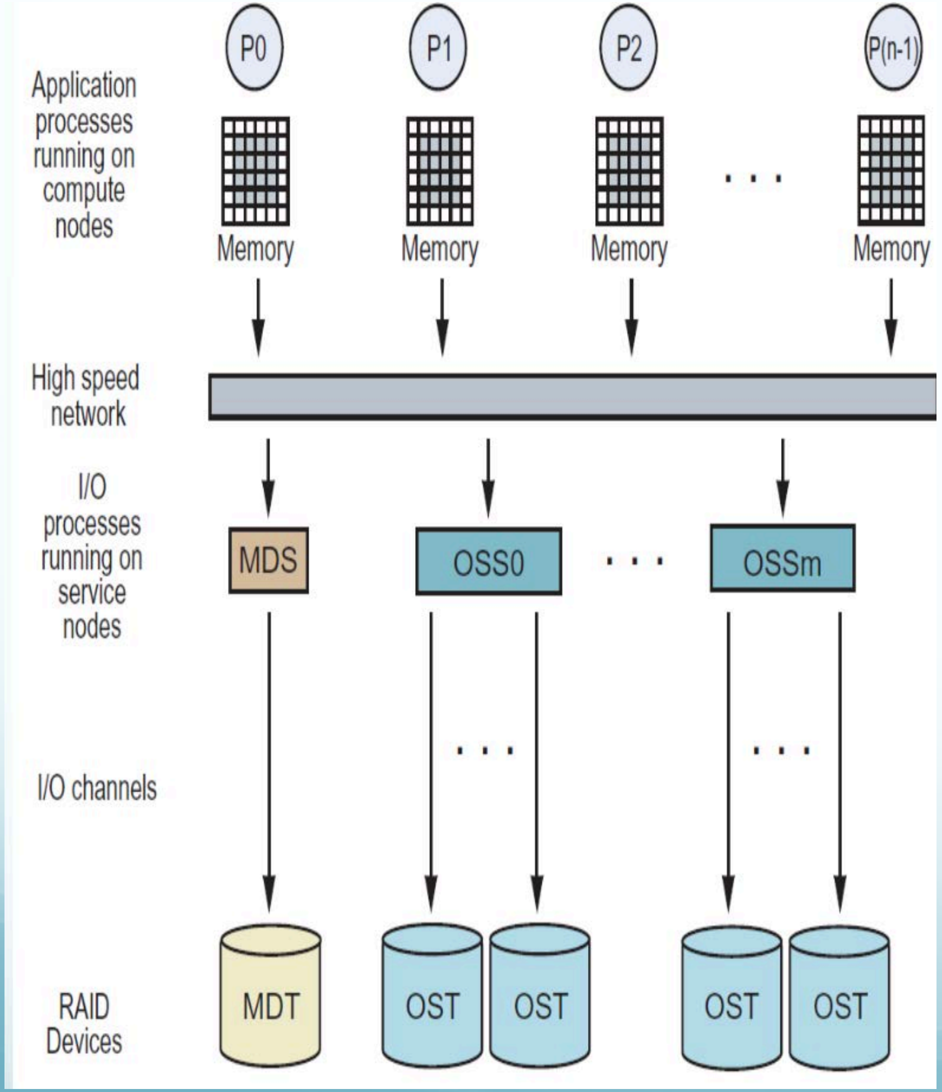
Lustre

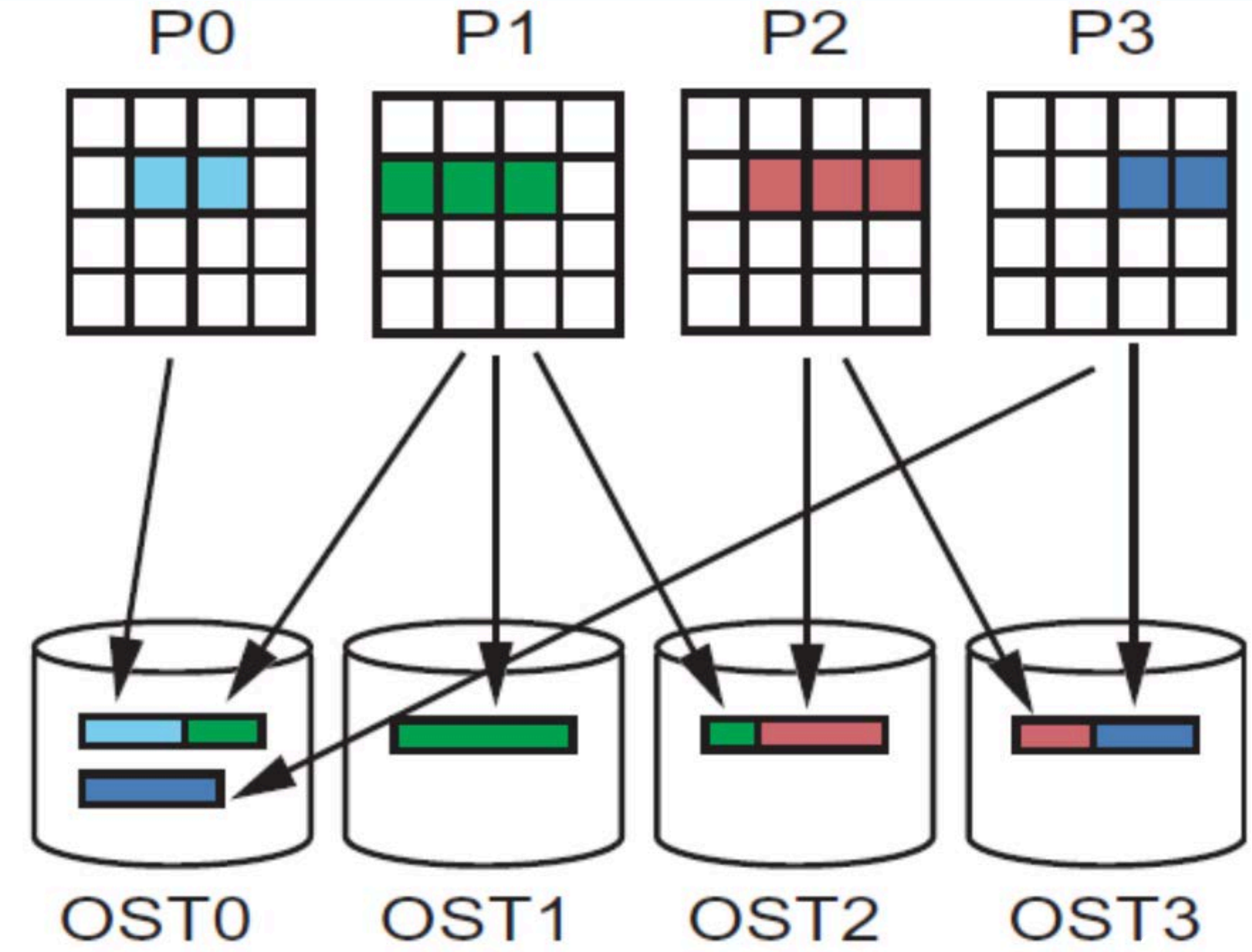
- Lustre file system is made up of an underlying:
  - Set of I/O servers called Object Storage Servers (OSSs)
  - Disks called Object Storage Targets (**OSTs**), stores file data (chunk of files). We have 144 OSTs on Shaheen

- The file metadata is controlled by a Metadata Server (MDS) and stored on a Metadata Target (MDT)



Lustre Operation



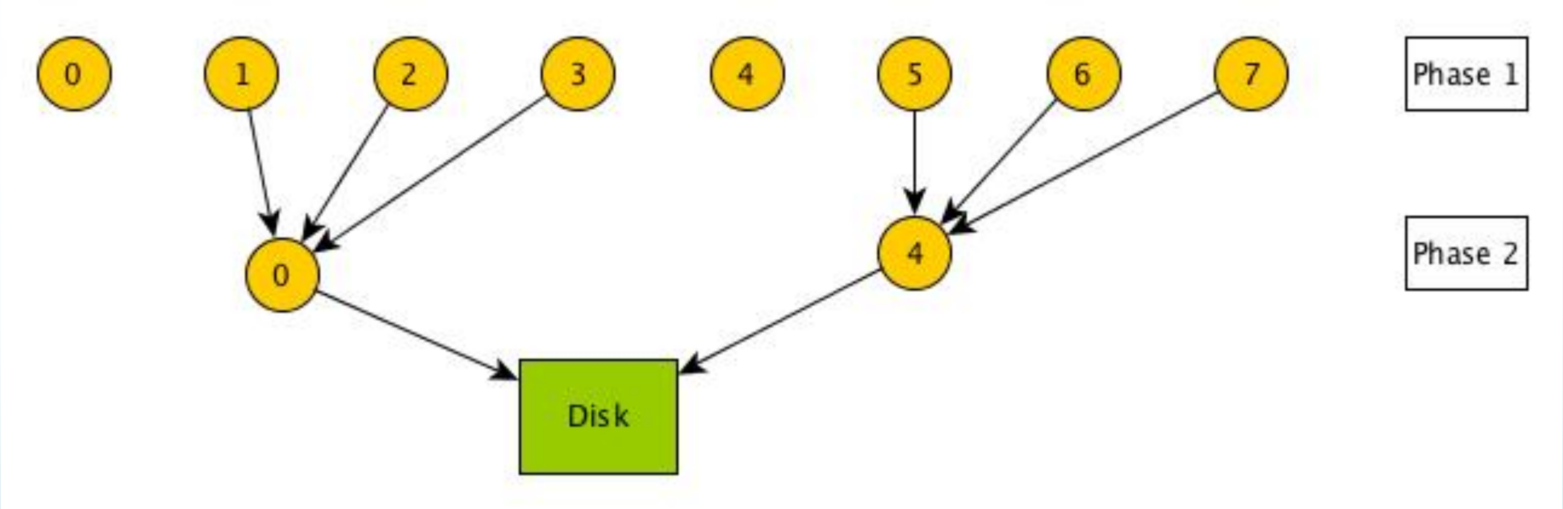


Lessons learned from Lustre

- Important factors:
  - Striping
  - Aligned data
  - But… how parallel is the I/O?

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.
- Example 8 MPI processes, 2 MPI I/O aggregators



How many MPI processes are writing a shared file?

- With CRAY-MPICH, we execute one application with 1216 MPI processes and it provides parallel I/O with Parallel NetCDF and the file's size is 360GB:

- First case (no stripping):
  - mkdir execution_folder
  - copy necessary files in the folder
  - cd execution_folder
  - run the application
  - Timing for Writing restart for domain   1:   **674.26** elapsed seconds

- **Answer**: 1 MPI process

How many MPI processes are writing a shared file?

- With CRAY-MPICH, we execute one application with 1216 MPI processes and it provides parallel I/O with Parallel NetCDF and the file’s size is 360GB:

- Second case:
  - mkdir execution_folder
  - lfs seststripe –c 144 execution_folder
  - copy necessary files in the folder
  - cd execution_folder
  - Run the application
  - Timing for Writing restart for domain 1: **10.35** elapsed seconds

- **Answer: 144 MPI processes**

Extract the list of the MPI I/O aggregators nodes

- export **MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY**=1

- First case:

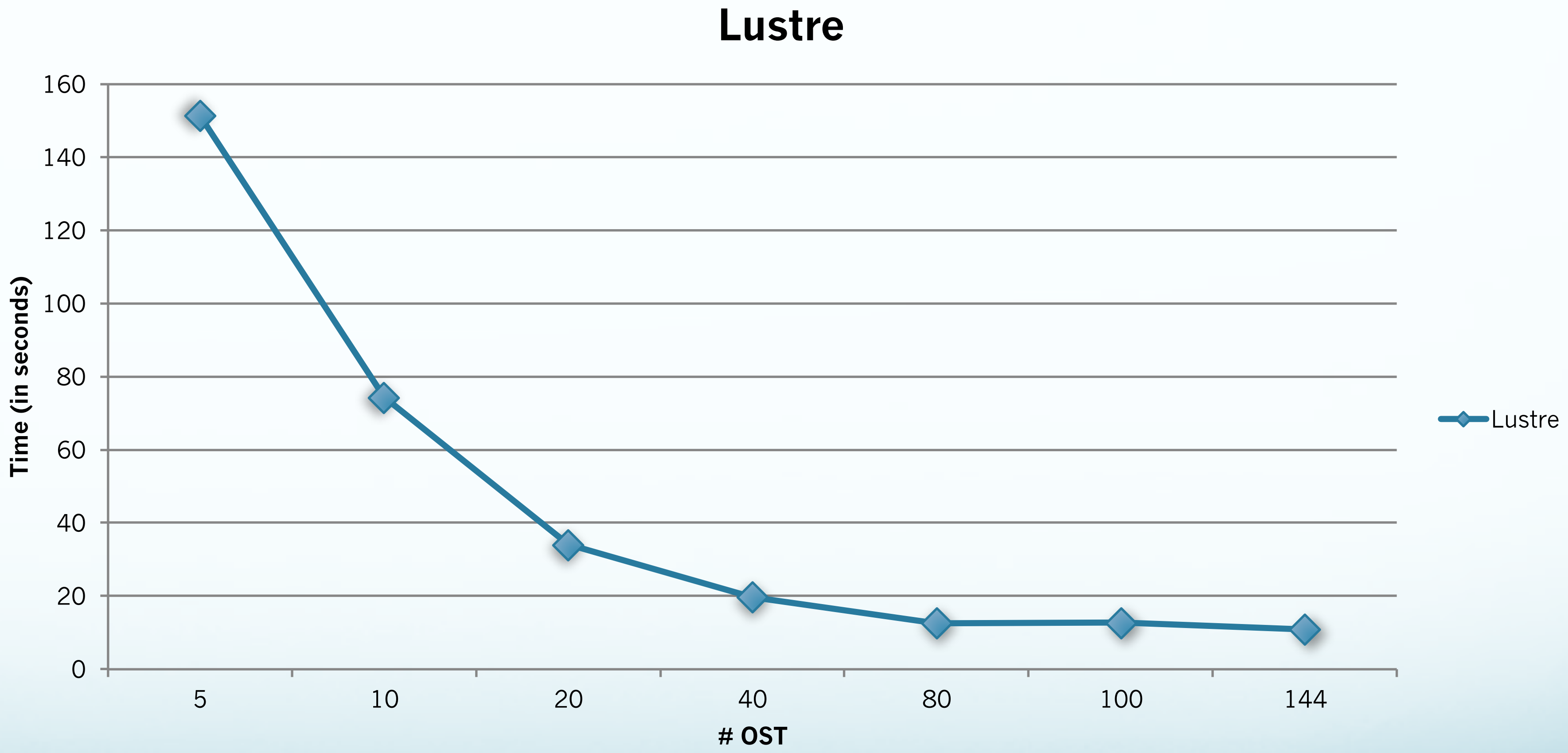
```
 AGG    Rank     nid
 ----   ------   --------
  0        0    nid04184
```

- Second case:

```
 AGG    Rank     nid
 ----   ------    --------
  0       0     nid00292
  1       8      nid00294
  ...
  143   1144  nid04592
```

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

I/O performance on Lustre while increasing OSTs



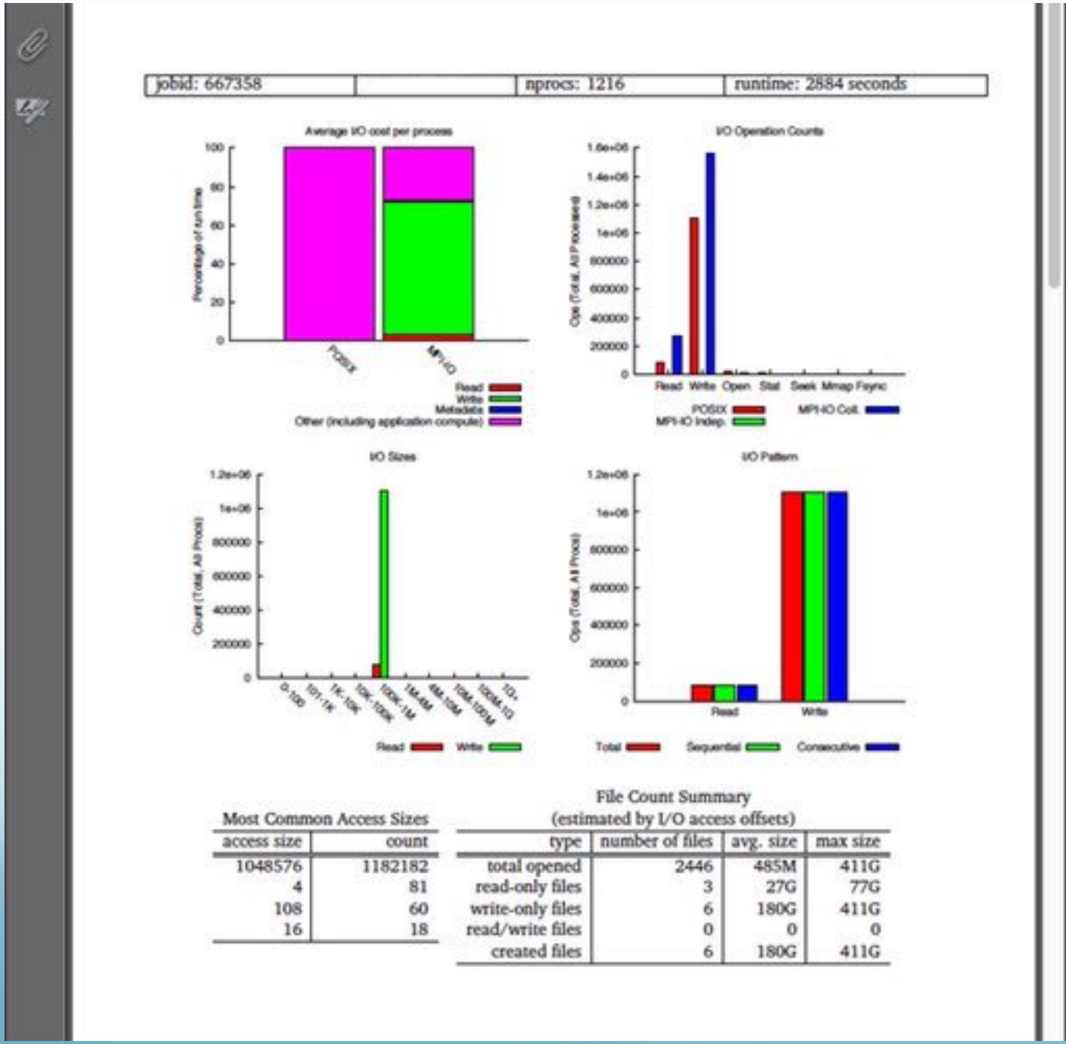
Declare the number of MPI I/O aggregators

- By default with the current version of Lustre, the number of MPI I/O aggregators is the number of OSTs.

- There are two ways to declare the striping (number of OSTs).
  - Execute the following command on an empty folder
    - lfs setstripe -c X empty_folder
      where X is between 2 and 144, depending on the size of the used files.

- Use the environment variable MPICH_MPIIO_HINTS to declare striping per files
  export MPICH_MPIIO_HINTS=
  "wrfinput*:striping_factor=64,wrfrst*:striping_factor=144,\
  wrfout*:striping_factor=144"

Using Darshan tool to visualize I/O performance



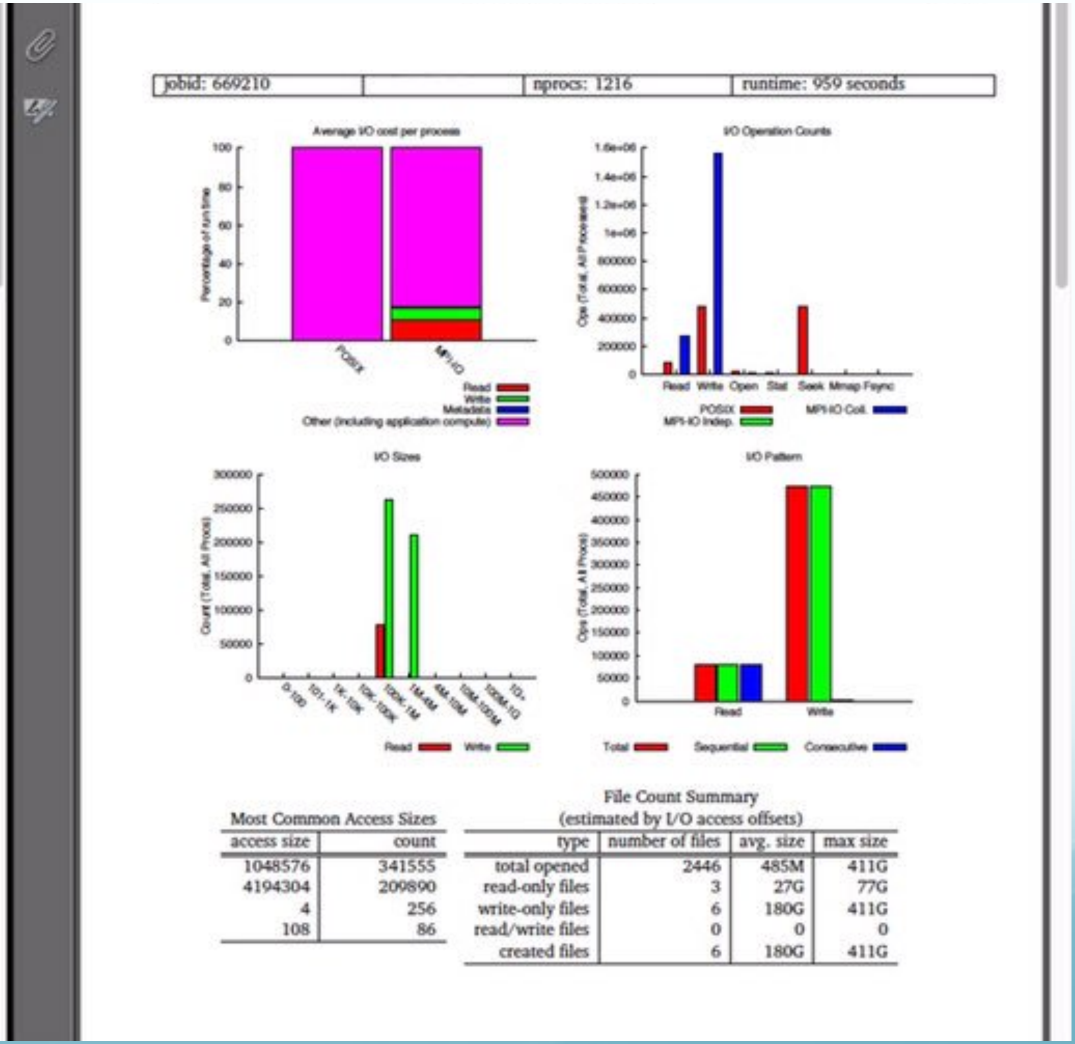
jobid: 667358		nprocs: 1216	runtime: 2884 seconds

Most Common Access Sizes

access size	count
1048576	1182182
4	81
108	60
16	18

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	2446	485M	411G
read-only files	3	27G	77G
write-only files	6	180G	411G
read/write files	0	0	0
created files	6	180G	411G



jobid: 669210		nprocs: 1216	runtime: 959 seconds

Most Common Access Sizes

access size	count
1048576	341555
4194304	209890
4	256
108	86

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	2446	485M	411G
read-only files	3	27G	77G
write-only files	6	180G	411G
read/write files	0	0	0
created files	6	180G	411G

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

Using Darshan tool

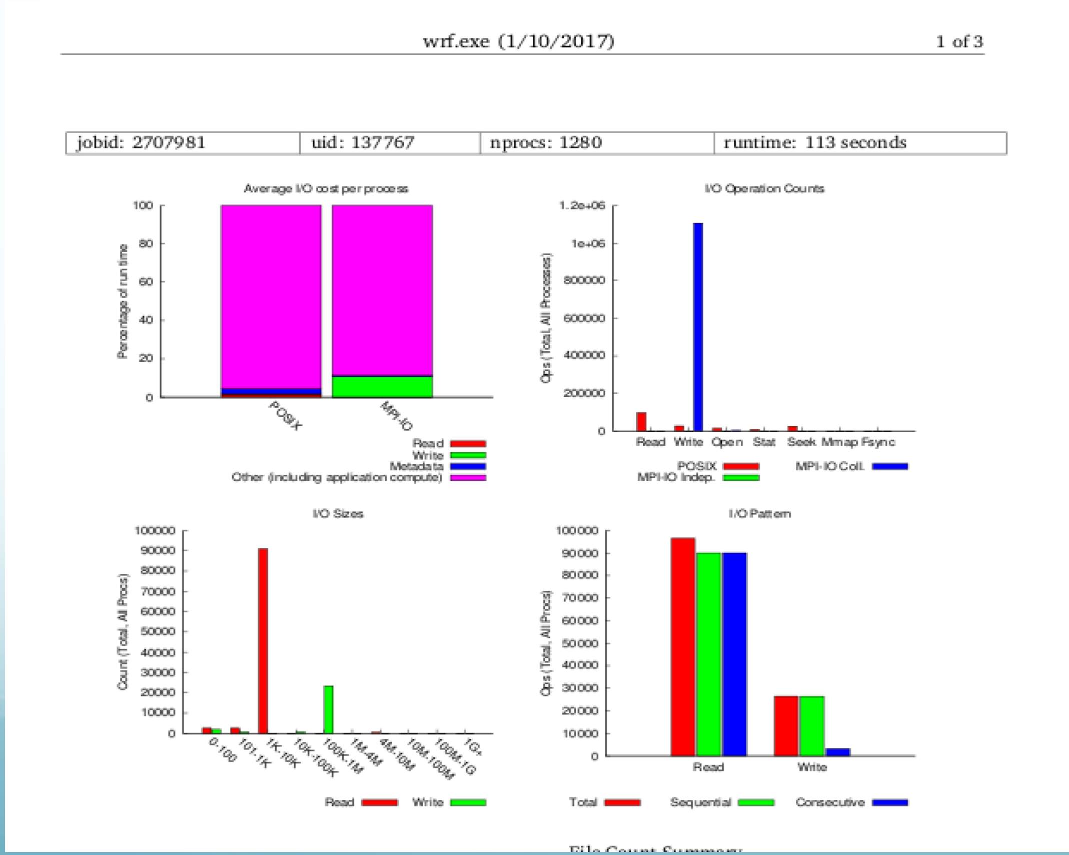
- Have you ever used Darshan tool?
  - If the answer is “I don’t know, probably not”, then maybe you have used it, as it is enabled automatic on Shaheen II and Cori.
- KAUST Supercomputing Laboratory (KSL) provides a framework to provide you easy access to performance data from Darshan:
  - Visit web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

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

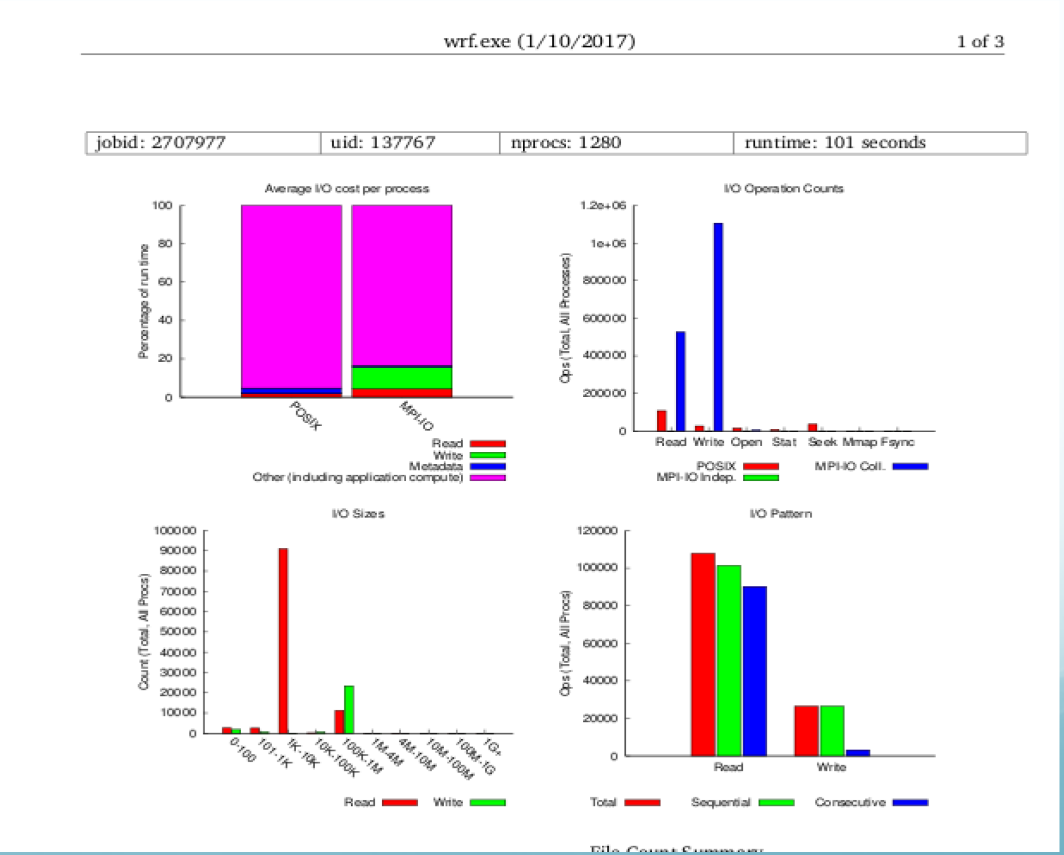
- In case that you want to compare the execution of two applications, execute:
  - *compare_darshan.sh job1_id job2_id*
  - **One** PDF file, with the Darshan performance data of both executions, is created



wrf.exe (1/10/2017) 1 of 3

jobid: 2707981	uid: 137767	nprocs: 1280	runtime: 113 seconds

File Count Summary



wrf.exe (1/10/2017) 1 of 3

jobid: 2707977	uid: 137767	nprocs: 1280	runtime: 101 seconds

File Count Summary

Discussion about Lustre

- There are many parameters to optimize Lustre, one quite interesting is the striping_unit. This declares the number of bytes to store on an OST before moving to the next OST

  - lfs setstripe -s X empty_folder   where X in bytes

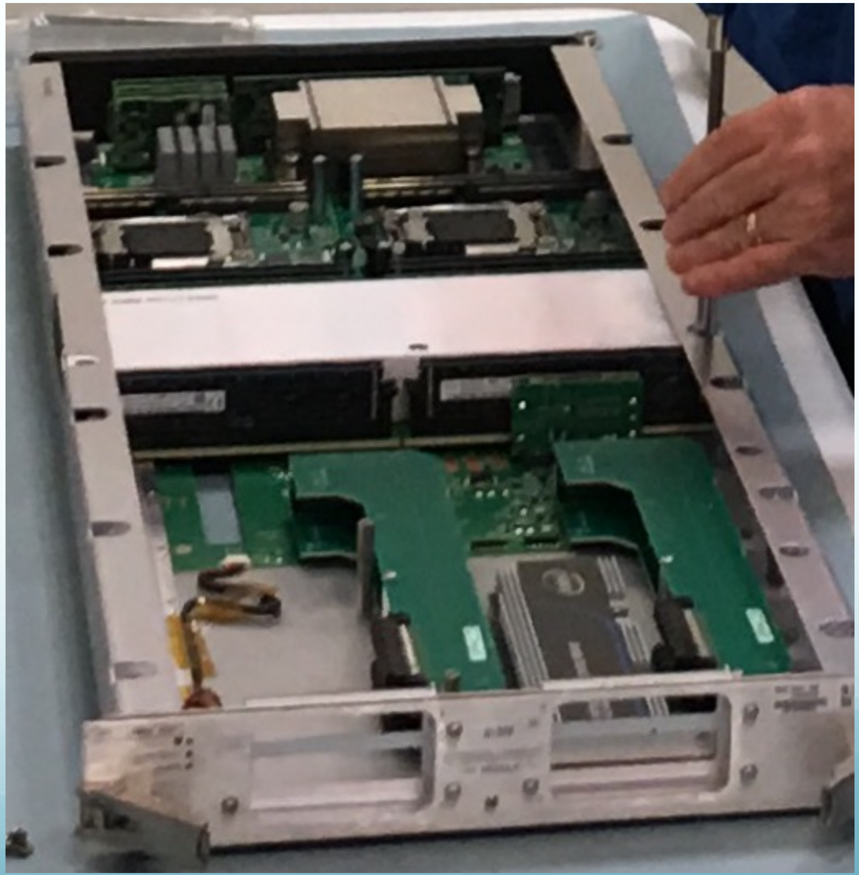
  - export MPICH_MPIIO_HINTS=
    "wrfinput*:striping_factor=64,wrfrst*:striping_factor=144:\
    striping_unit=4194304,wrfout*:striping_factor=144:\
    striping_unit=2097152”

Useful MPI environment variables

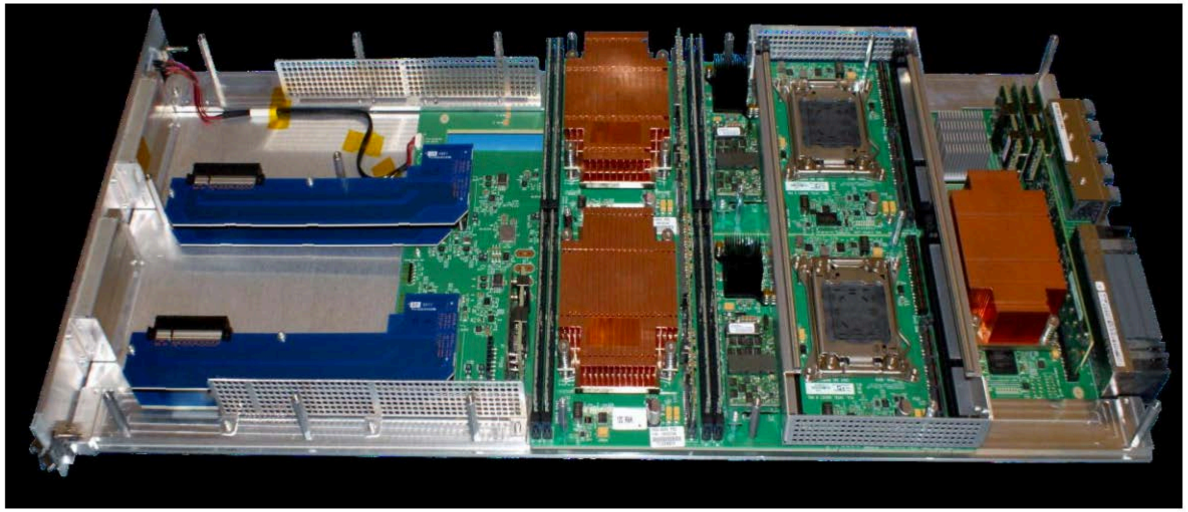
- export MPICH_ENV_DISPLAY=1
  - Displays all settings used by the MPI during execution
- export MPICH_VERSION_DISPLAY=1
  - Displays MPI version
- export MPICH_MPIIO_HINTS_DISPLAY=1
  - Displays all the available I/O hints and their values
- export MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY=1
  - Display 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)

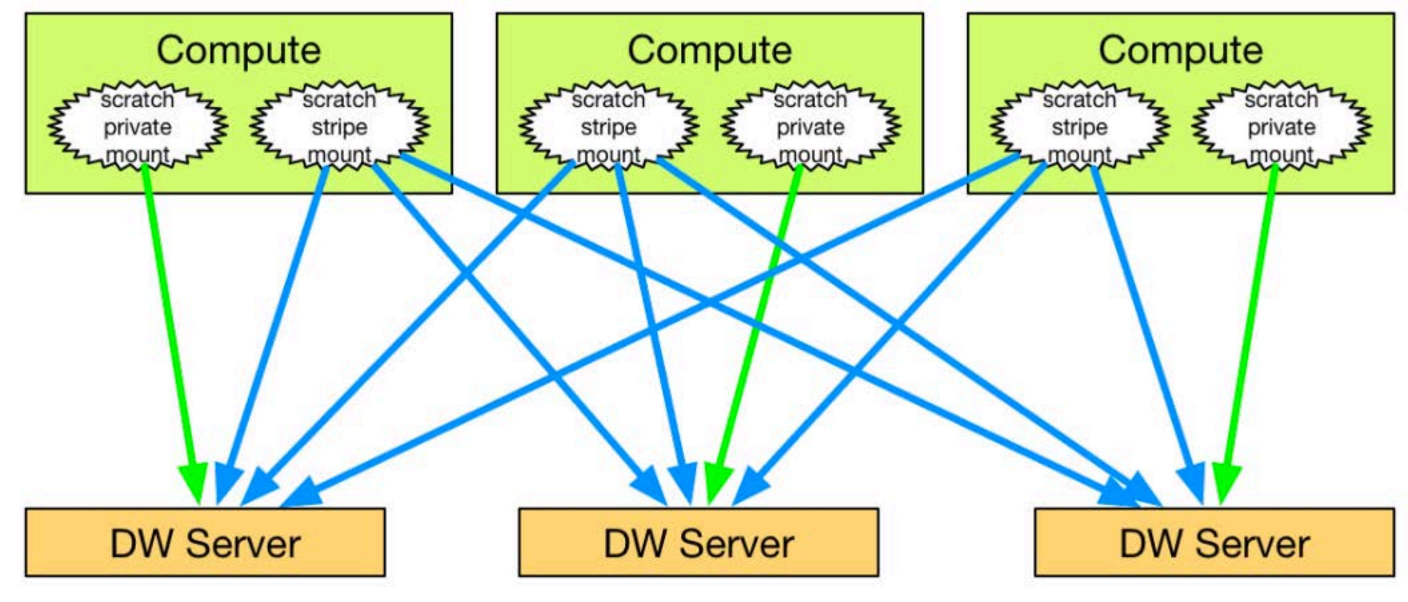
Burst Buffer

- Shaheen II: 268 Burst Buffer nodes, 536 SSDs, totally 1.52 PB, each node has 2 SSDs
- 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



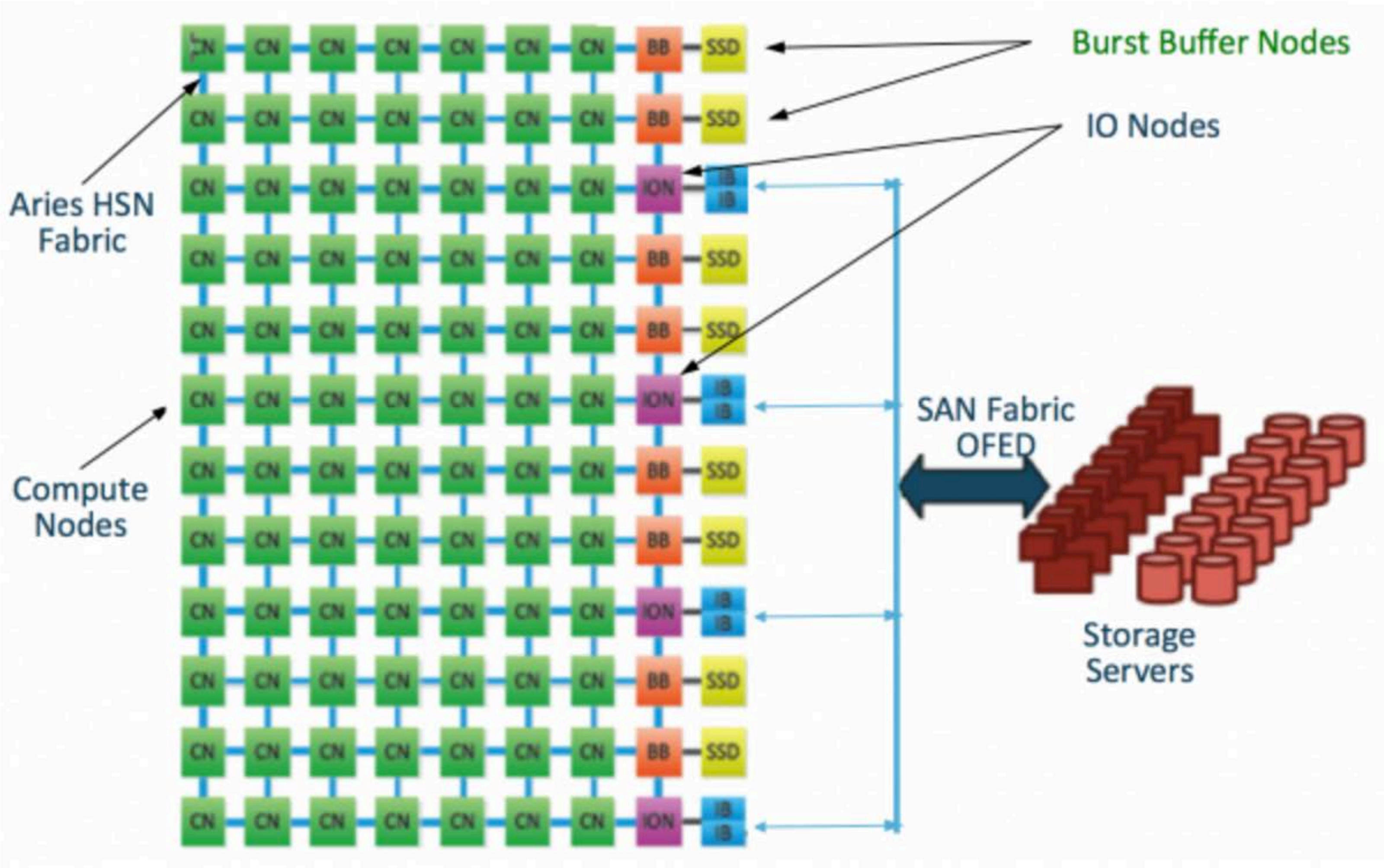
Burst Buffer Architecture





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

Burst Buffer Architecture



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

- **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 368GiB

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
```

Check if there are jobs using BB

```
> scontrol show burst
Name=cray DefaultPool=wlm_pool Granularity=406976M
TotalSpace=1636043520M UsedSpace=0
  Flags=EnablePersistent
 StageInTimeout=1800 StageOutTimeout=1800 ValidateTimeout=5
OtherTimeout=300
  AllowUsers=...markomg...
  GetSysState=/opt/cray/dw_wlm/default/bin/dw_wlm_cli
```

If your username is not in the list of AllowUsers while you have applied for BB early access, send email to help@hpc.kaust.edu.sa

```
scontrol show burst
Name=cray DefaultPool=wlm_pool Granularity=406976M
TotalSpace=1636043520M UsedSpace=813952M
 Flags=EnablePersistent
StageInTimeout=1800 StageOutTimeout=1800 ValidateTimeout=5
OtherTimeout=300
AllowUsers=...,markomg...
GetSysState=/opt/cray/dw_wlm/default/bin/dw_wlm_cli
Allocated Buffers:
JobID=2729000 CreateTime=2017-01-20T17:15:31 Pool=wlm_pool
Size=813952M State=allocated UserID=markomg(137767)
Per User Buffer Use:
UserID=markomg(137767) Used=813952M
```

Burst Buffer Nodes Allocation

- How many DW instances per node?

DW_instances_per_node = 5.82*1024/368 = **16.19**

A DW node can accommodate up to 16*368/1024 = 5.75 TB

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

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***1024/368 = **166.95**, so we will reserve **167** DW nodes.

Important: If you reserve more than 268 * 368/1024 = **96.31TB**, then some DW nodes will be used twice and this can cause I/O performance issues

Burst Buffer Modes

- DW supports two access modes

  - **Private**
  Each of the compute job has its own private space on BB and it will lot be visible to other compute jobs. For now, data is not striped over BB nodes in private mode (not tested). Each compute node has access to a BB allocation equal to the granularity size.

  - **Striped**
  The data will be striped over several Burst Buffer nodes. BB nodes are allocated on a round-robin basis. We use this mode mainly

- BB supports two reservation modes
  - **Scratch** is temporary space allocation which will be removed when the job is finished
  - **Persistent** is when you have many jobs that need to access the same files, so this mode creates a DW space that persists after a job is finished and it is available to other of your DW jobs. **Important**: Persistent space is not a backup solution, you could lose your data in case of any BB problem

Burst Buffer Workflow

- Initially the files are located on Lustre filesystem

- For the files that need to be accessed multiple times but also for any big files you should move these files on BB before your job reservation. This phase is called **stage-in.** You can stage-in either file or folder.

- When the job finishes, the created files will be returned to the folder that the user declared in the script, this is called **stage-out**.

- The files on BB are located inside the path declared by environment variable $DW_JOB_STRIPED (for striped mode)

Note: Stage-in and –out are not mandatory it depends what the user needs. Maybe there are no input files or the user wants just to measure the execution time.

Modify SLURM script

- Lustre reservation

```
#!/bin/bash
#SBATCH --partition=workq
#SBATCH -t 10:00:00
#SBATCH -A k01
#SBATCH --nodes=32
#SBATCH  --ntasks=1024
#SBATCH   -J slurm_test
```

Comment: Insert the DW commands, exactly after the SBATCH commands, do not include any other unrelated commands between SBATCH and DW declarations.

Modify SLURM script

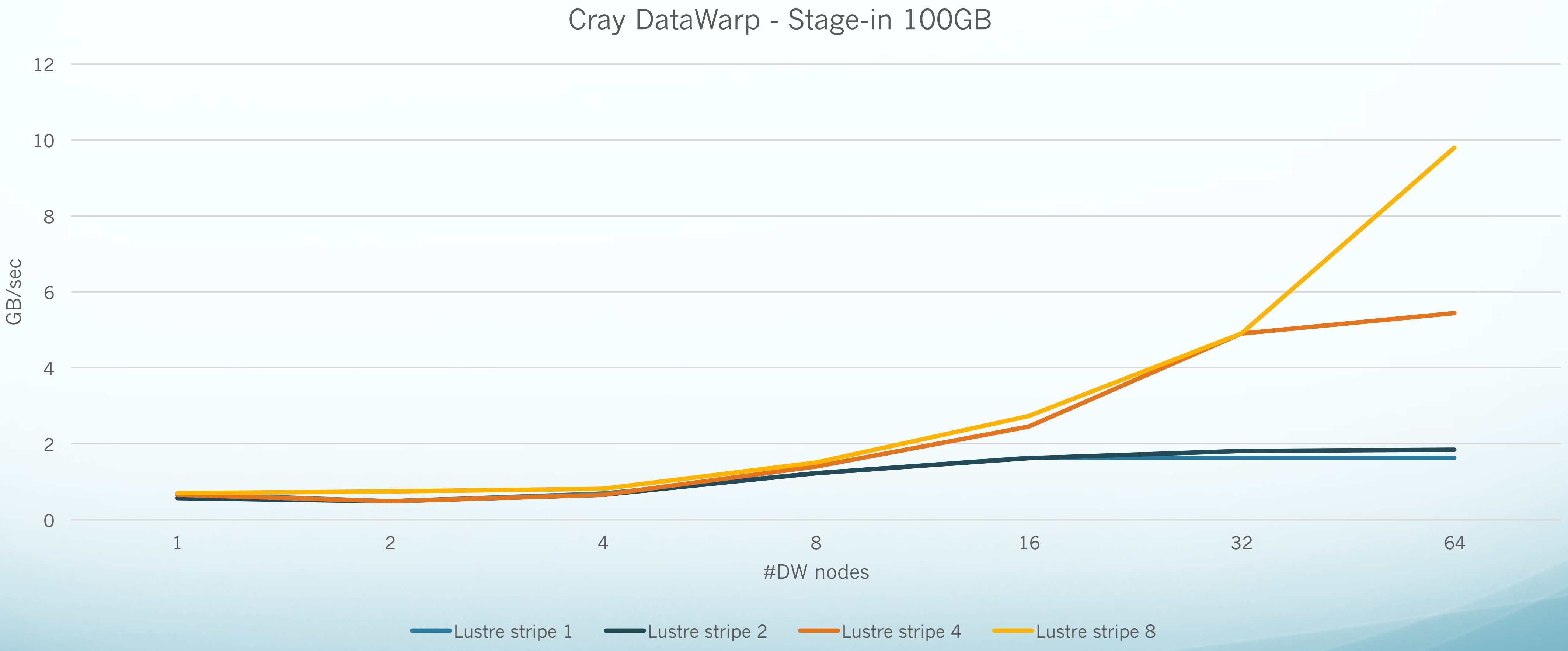
- BB reservation (2TB of DW space)

```
#!/bin/bash
#SBATCH --partition=workq
#SBATCH -t 10:00:00
#SBATCH -A k01
#SBATCH --nodes=32
#SBATCH  --ntasks=1024
#SBATCH   -J slurm_test

#DW jobdw type=scratch access_mode=striped capacity=2TiB
#DW stage_in type=directory source=/scratch/markomg/for_bb
destination=$DW_JOB_STRIPED
#DW stage_out type=directory destination=/scratch/markomg/back_up
source=$DW_JOB_STRIPED/

cd $DW_JOB_STRIPED

chmod +x executable
```

Note: You can stage-in/out also files instead of directory

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

How fast is stage-in?



DataWarp – Restrictions

- When you stage-in executables, you need to execute a command when you are on BB, that this file is executable (*chmod +x executable*)

- Symbolic links will be lost during stage-in

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 Lustre?

Answer: 6!

Table 6: File Output Stats by Filename

```
 Write Time |   Write MBytes |   Write Rate    |     Writes |  Bytes/ Call |File Name
            |                |   MBytes/sec |                |                     | 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”
- 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 out 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 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.

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

- 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.
  - 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.
  - $MPI\_IO\_aggregators = \begin{cases} DW\_nodes, if\ we\ use\ one\ aggregator\ per\ DW\ node \\ k * DW\_nodes, where\ k \in \mathbb{N}, 2 \leq k \leq 128 \end{cases}$
  - Number_of_total_MPI_processes= $l * MPI\_IO\_aggregators, where\ l \in \mathbb{N}, l \geq 2$

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. As these two numbers should be divided you can compute how many nodes you have to request.

- If you need for example 64 DW nodes, then you should calculate the requested space as follows:
  - Multiply with the DW granularity:
    - 64*368=23552

Create persistent DW space I

- Create persistent DW space

```
#!/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
```

Checking the status of the persistent DW reservation

```
> dwstat most

sess state       token creator  owner              created expiration nodes
985 CA--- george_test     CLI 137767 2017-01-20T18:01:00      never     0

inst state sess      bytes nodes              created expiration intact       label public confs
977 CA---   985 736GiB      2 2017-01-20T18:01:01       never   true george_test   true       1
```

```
> dwstat nodes

    node      pool          online drain  gran capacity insts activs
 nid01349 wlm_pool    true false 16MiB  5.82TiB      1       0
 nid01410 wlm_pool    true false 16MiB  5.82TiB      1       0
```

Use DW persistent space I

```
#!/bin/bash
#SBATCH --partition=workq
#SBATCH -t 10
#SBATCH -A k01
#SBATCH --nodes=1
#DW persistentdw name=george_test
#DW stage_in type=directory source=/project/k01/markomg/wrf
destination=$DW_PERSISTENT_STRIPED_george_test

cd $DW_PERSISTENT_STRIPED_george_test/

…

exit 0
```

Use DW persistent space II

- Now, you can execute the second job on the persistent space, however, do **not** stage-in the 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**

If the problem is not solved send us email immediately! help@hpc.kaust.edu.sa, inform also the BB users through bb_users@hpc.lists.kaust.edu.sa

Use DW persistent space III

- 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 bbf.conf with the following:
    - #DW persistentdw name=**george_test**
    - Execute:

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

Use DW persistent space IV

- 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
```

Finalize DW persistent reservation

- When the experiments are finished, then stage-out the files. Do not copy the files from the interactive mode back to Lustre as this can be much slower, depending on the file sizes.
- Finally delete the DW space

Use DataWarp for Medata 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

Applications/Benchmarks

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

Data Centric Optimizations of Seismic Natural Migration Algorithm at Scale on Parallel File Systems and Burst Buffer

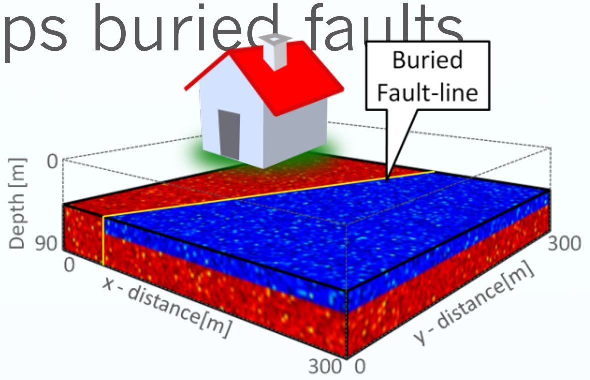
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

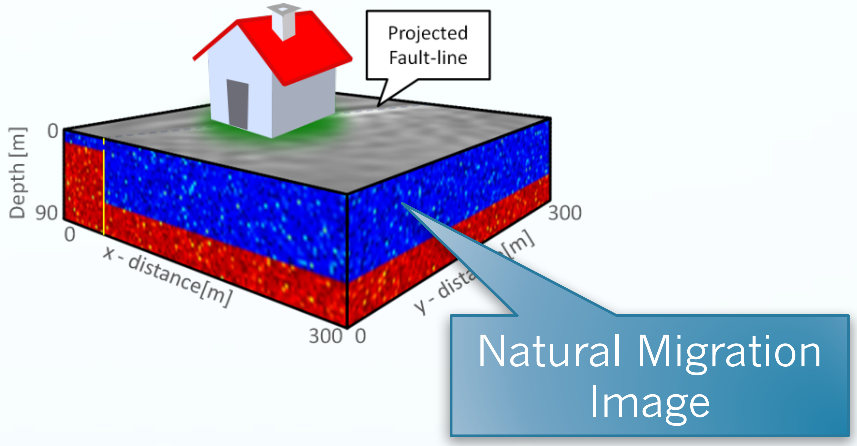
Outline

- Seismic Natural Migration
- I/O optimizations
  - on parallel filesystem
  - using Cray DataWarp Burst Buffer
- Summary and Future Work

Seismic Natural Migration

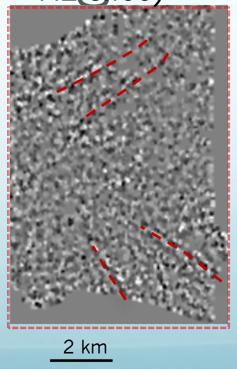
- Natural Migration is a seismic imaging tool that maps buried faults



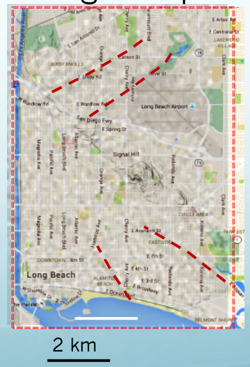


- Application to Long-Beach, CA area

Natural Migration (2.0 Hz slice)



Overlay on Google Maps



Buried fault lines in the subsurface are shown as lineaments in the images

Unknown faults are under populated LA areas.

Seismic Natural Migration

- Natural Migration is a seismic imaging tool that maps buried faults.
- The Algorithm uses recorded Green's functions G(*s*,*x*,t) to compute an image:

$$Image(\boldsymbol{x}) = \sum_{\boldsymbol{s}}^{N} \sum_{\boldsymbol{r}}^{N} [G(\boldsymbol{s}, \boldsymbol{x}, t) * G(\boldsymbol{r}, \boldsymbol{x}, t)] \cdot G_1(\boldsymbol{s}, \boldsymbol{r}, t) \ ,$$

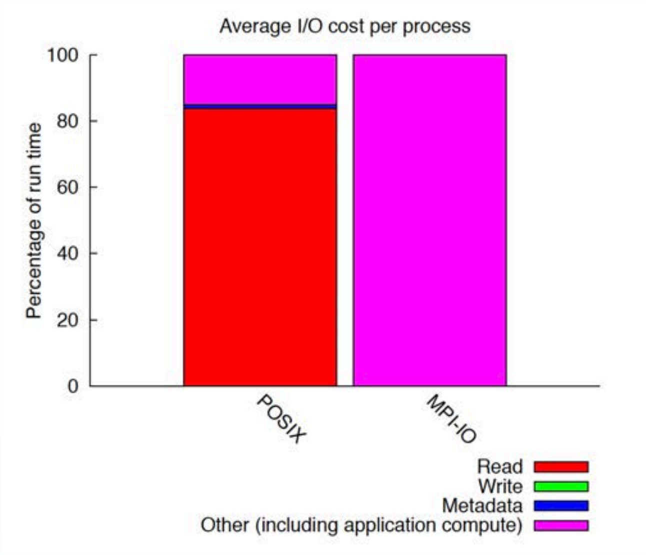
where the *s* and *r* denote seismic data coordinates, and *x* denotes image coordinates.

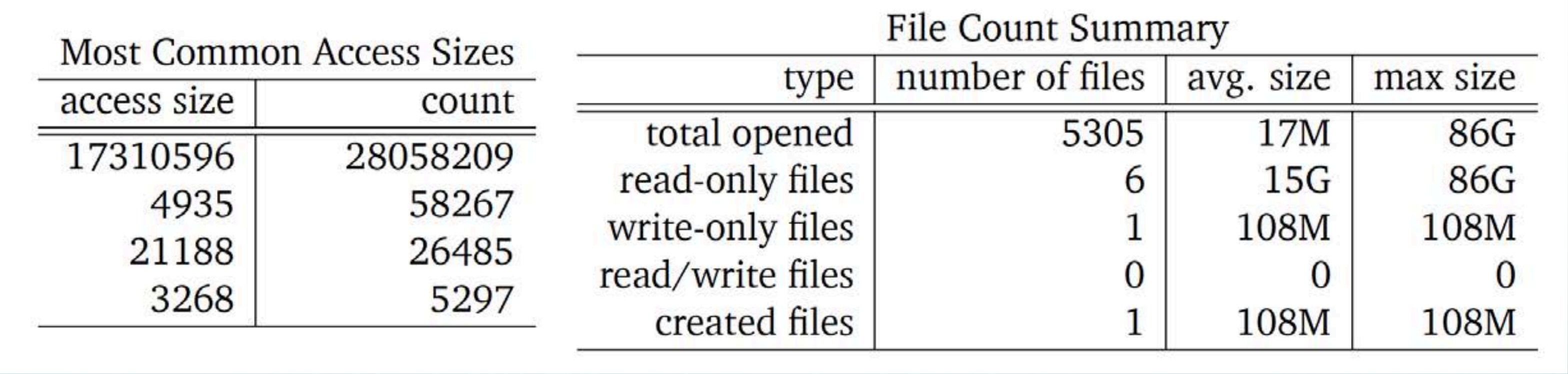
- The Green's functions are pre-computed and stored in a single file with more than 86GB of size (for this experiment)

Computational Aspects

- Natural migration equation: $Image(\boldsymbol{x}) = \sum_{s}^{N} \sum_{r}^{N} [G(\boldsymbol{s},\boldsymbol{x},t) * G(\boldsymbol{r},\boldsymbol{x},t)] \cdot G_1(\boldsymbol{s},\boldsymbol{r},t)$,
- There are N=5297 Green's functions.
- The outer summation is distributed among MPI processes
- All runs are configured with one MPI process per socket using 16 OpenMP threads.
- Each MPI process loads the whole 86GB file in parts (one Green's function at a time) to compute the inner summation.
- The time convolution and dot-product operations in the equation above are computationally cheap compared to the I/O cost for retrieving the Green's function from disks.

Natural Migration I/O Profile Before Tuning





Most Common Access Sizes

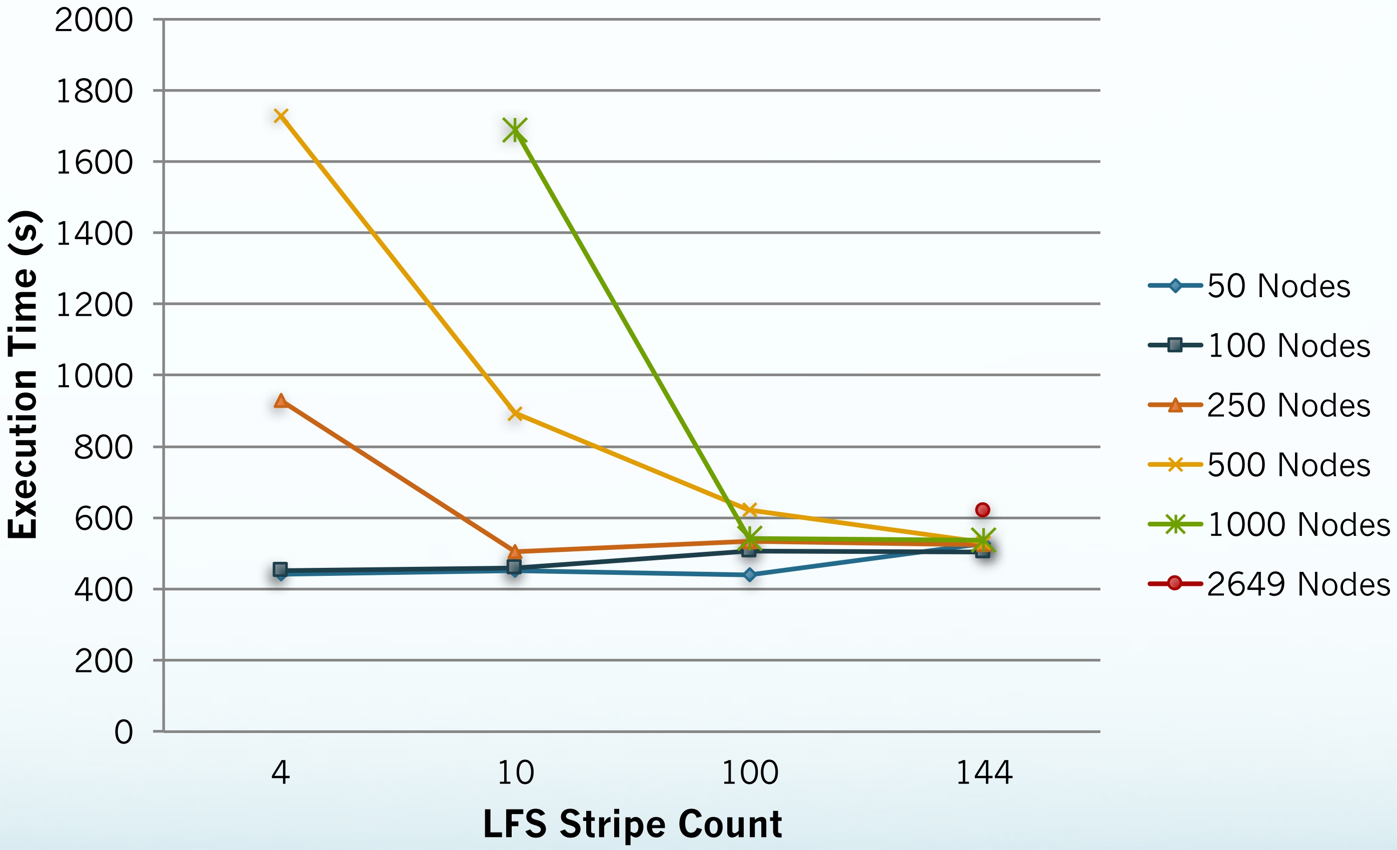
access size	count
17310596	28058209
4935	58267
21188	26485
3268	5297

File Count Summary

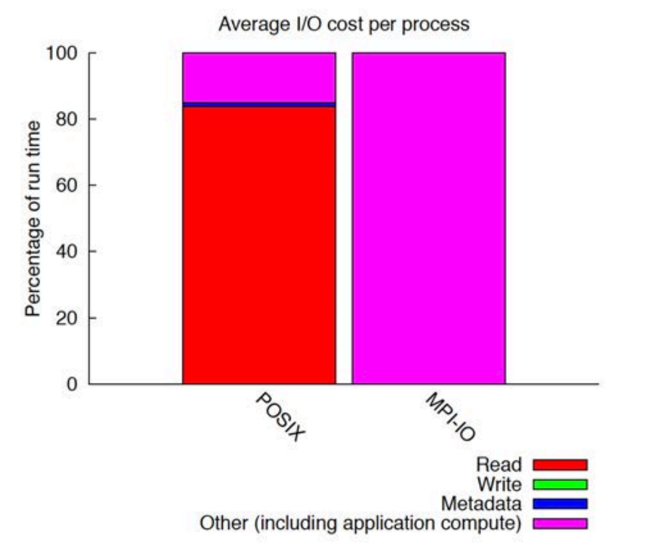
type	number of files	avg. size	max size
total opened	5305	17M	86G
read-only files	6	15G	86G
write-only files	1	108M	108M
read/write files	0	0	0
created files	1	108M	108M

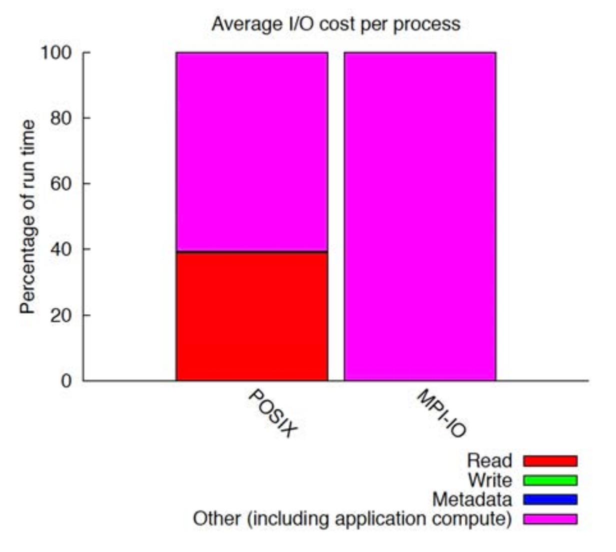
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

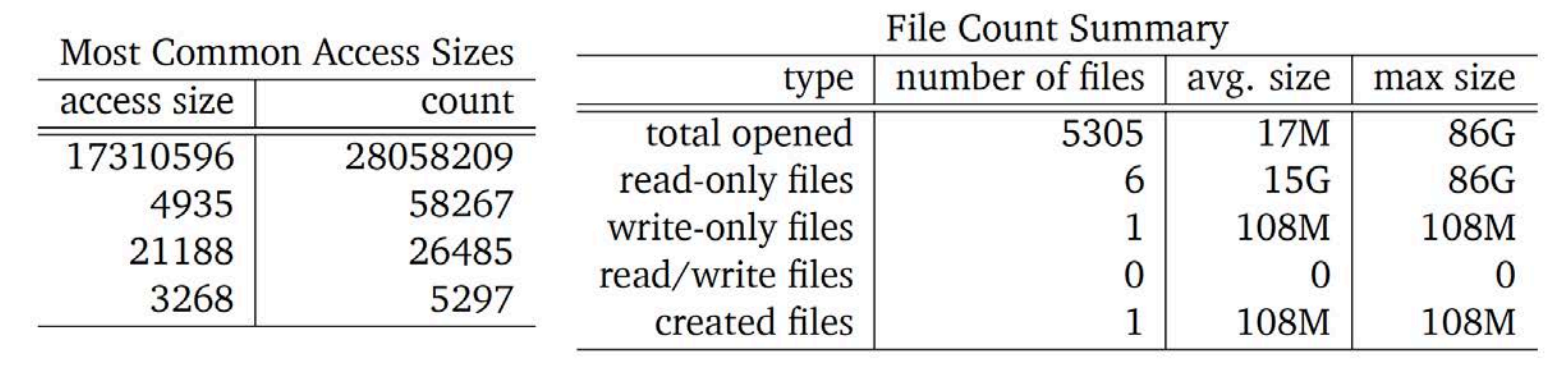
Tuning Lustre Stripe Count for Natural Migration



Natural Migration I/O Profile After Tuning







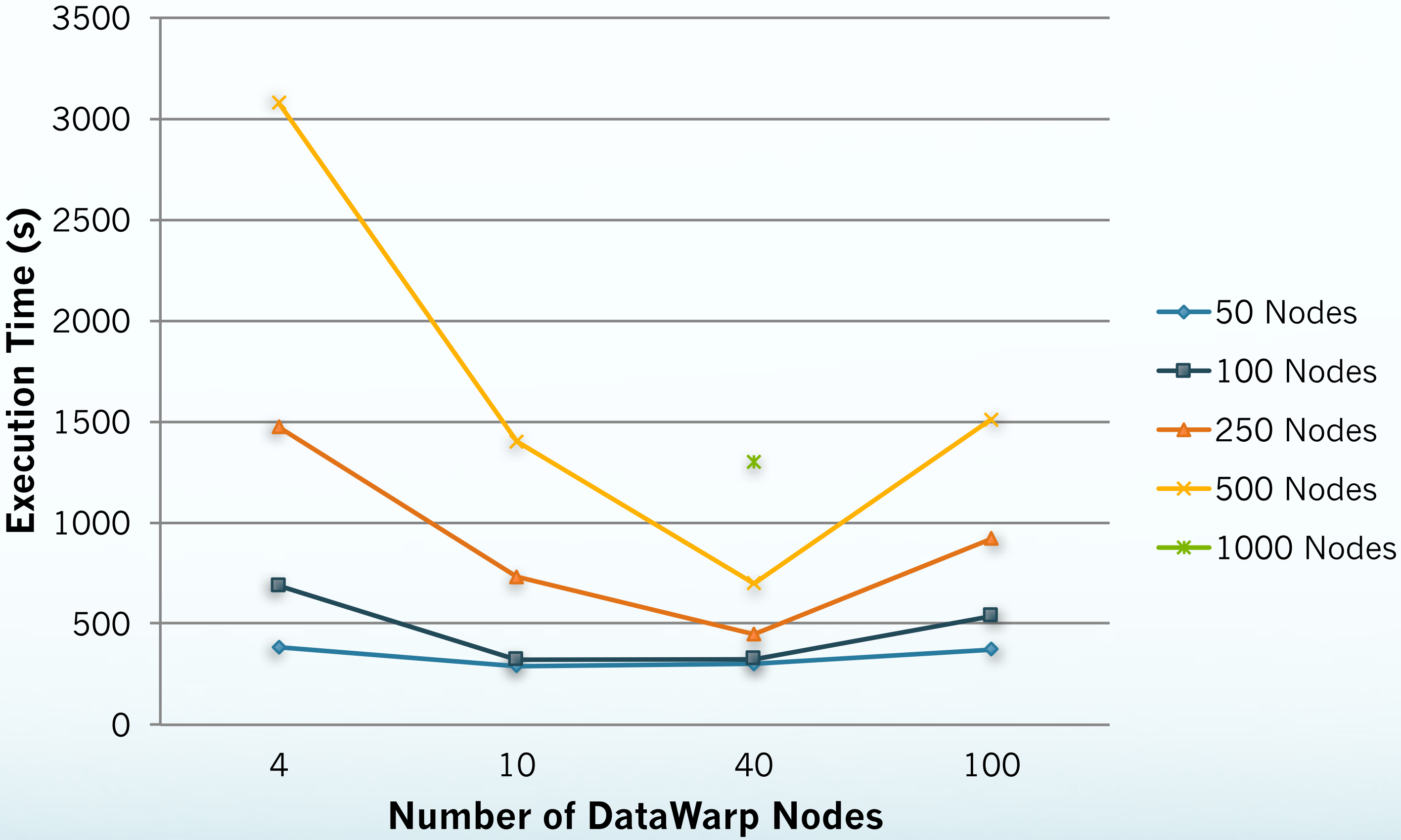
Most Common Access Sizes

access size	count
17310596	28058209
4935	58267
21188	26485
3268	5297

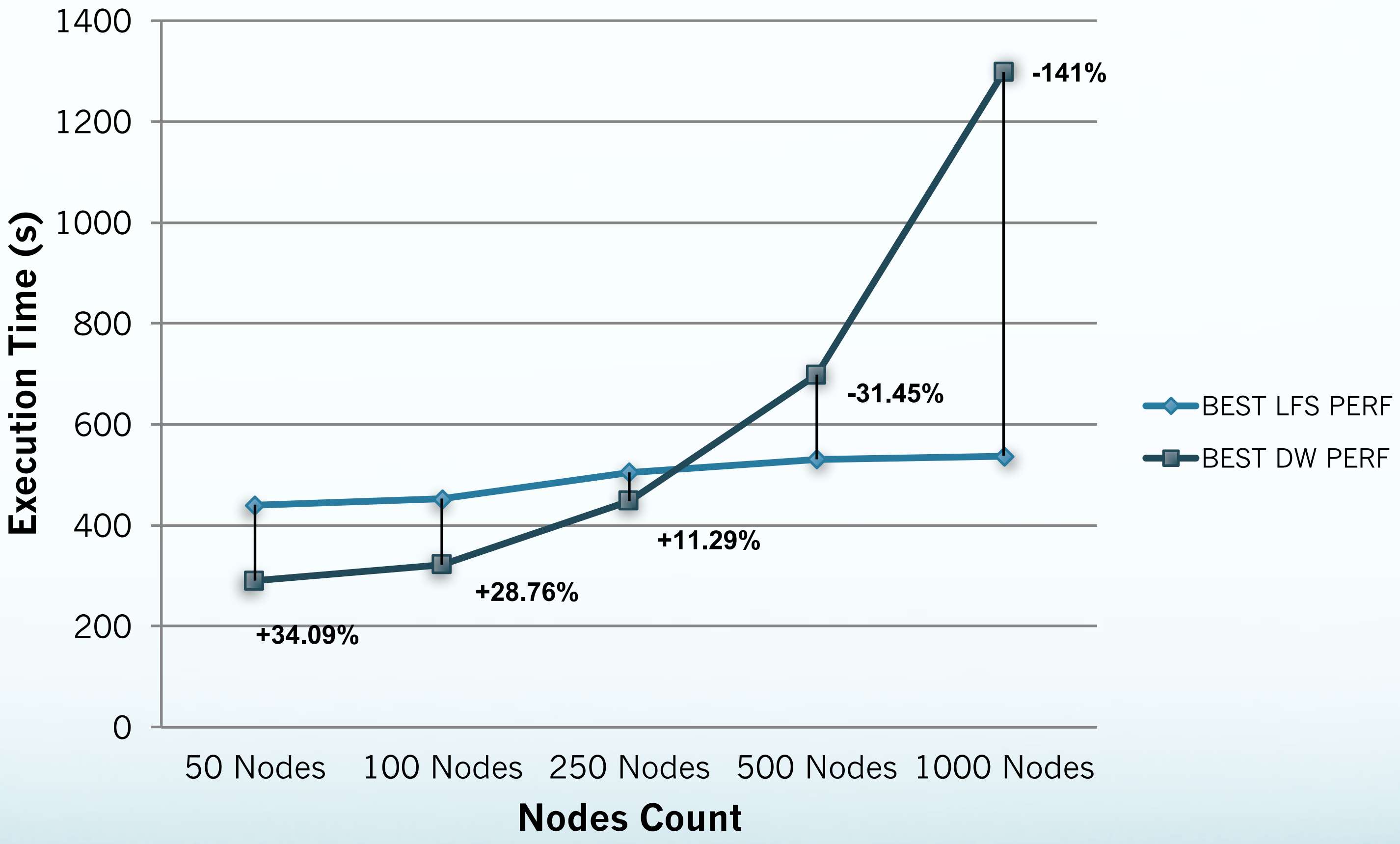
File Count Summary

type	number of files	avg. size	max size
total opened	5305	17M	86G
read-only files	6	15G	86G
write-only files	1	108M	108M
read/write files	0	0	0
created files	1	108M	108M

Tuning DataWarp Nodes Count for Natural Migration



Lustre Filesystem vs DataWarp



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

Summary and Future Work on Seismic Natural Migration Algorithm

- Tuning Lustre stripe count significantly improved the seismic natural migration code, especially at larger scale.
- Natural migration code benefited from DataWarp burst buffer up to a certain scale with up to 34% improvement.
- Next Steps: study the performance of algorithmic changes to minimize I/O and use MPI communications instead.

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

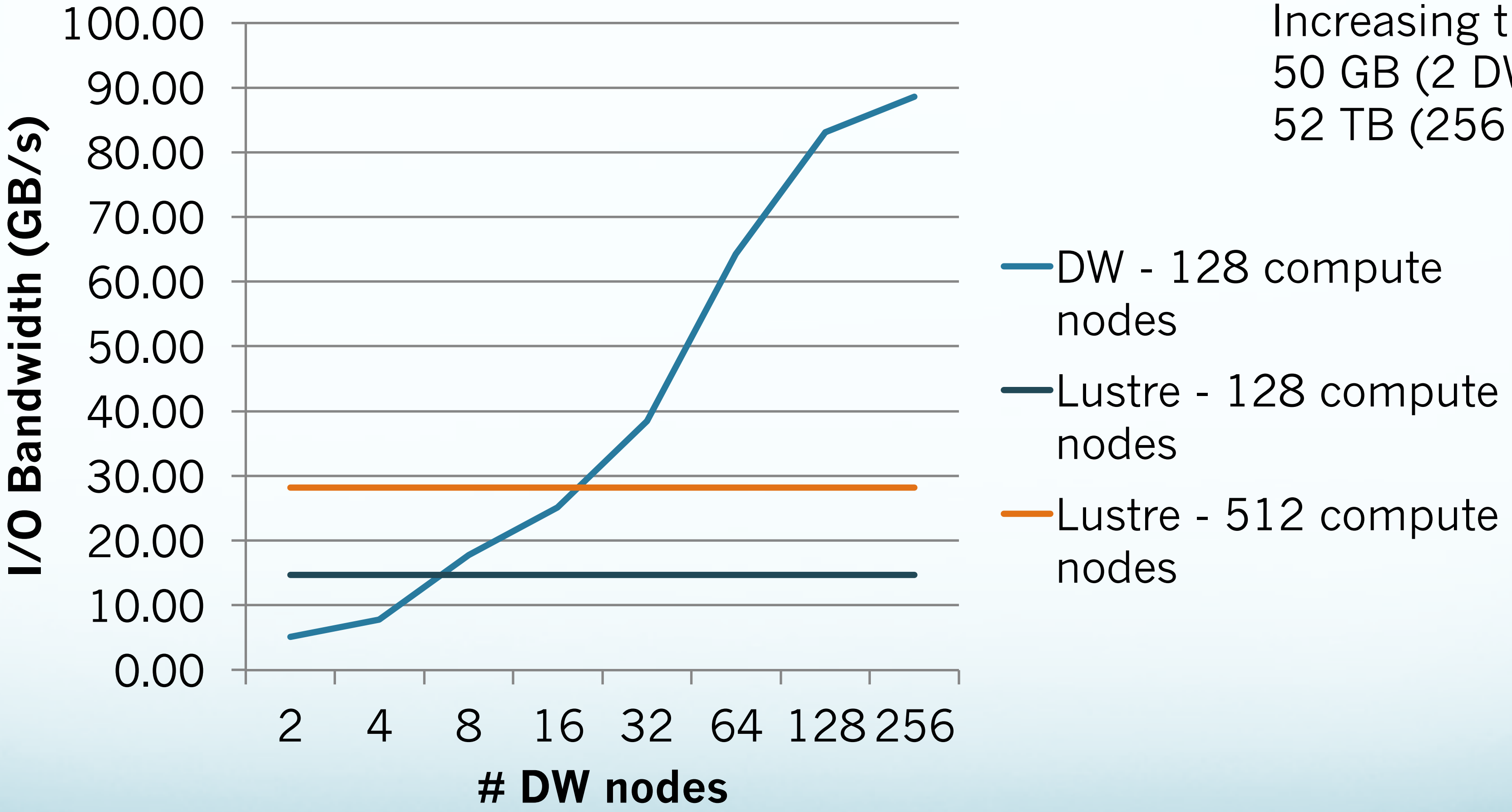
Study-case NAS BTIO

Applications

NAS BTIO

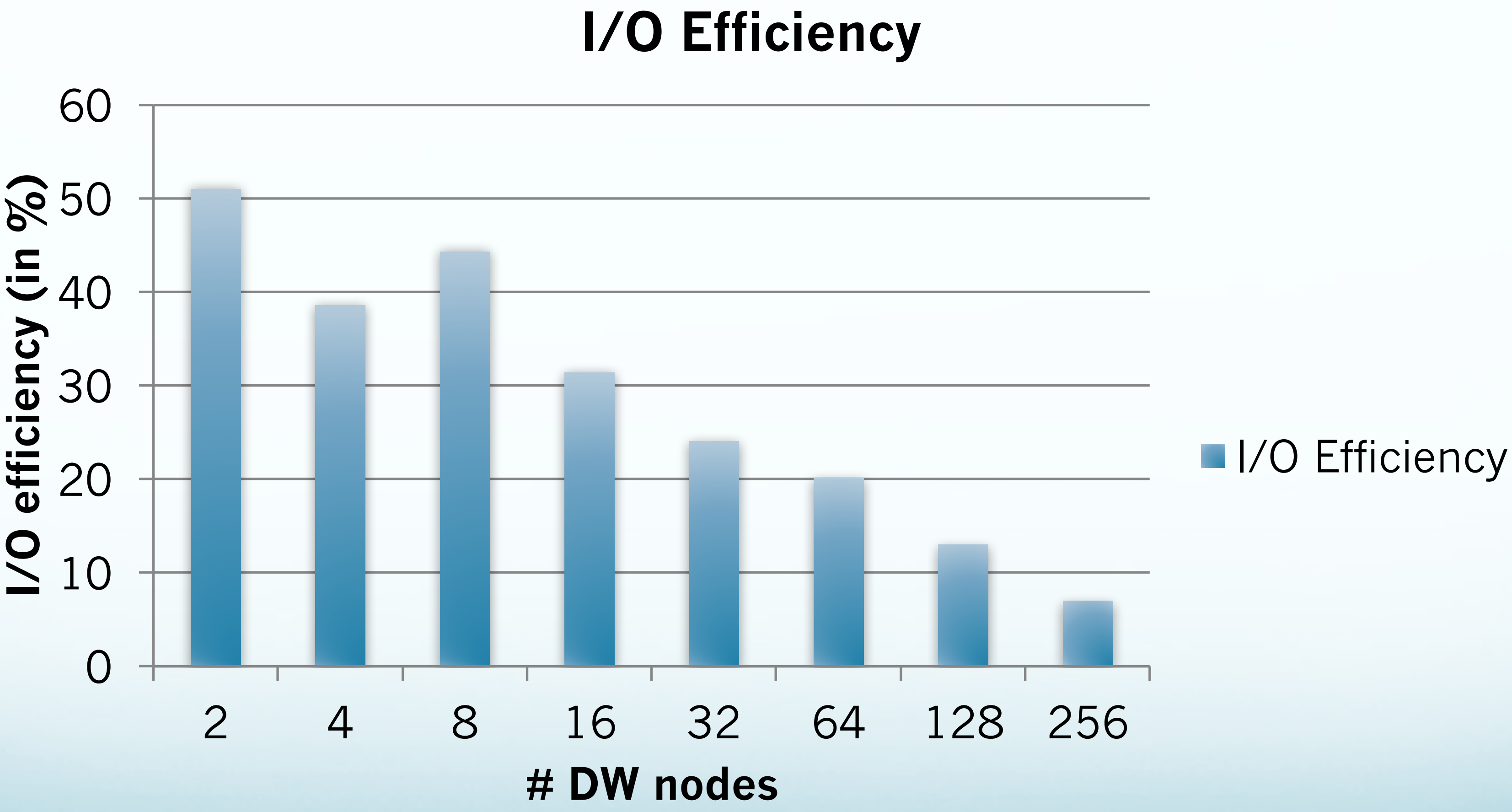
“As part of the NAS parallel benchmark set an IO benchmark has been developed which is based on one of the computational kernels. The BT benchmark is based on a CFD code that uses an implicit algorithm to solve the 3D compressible Navier-Stokes equations.”

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

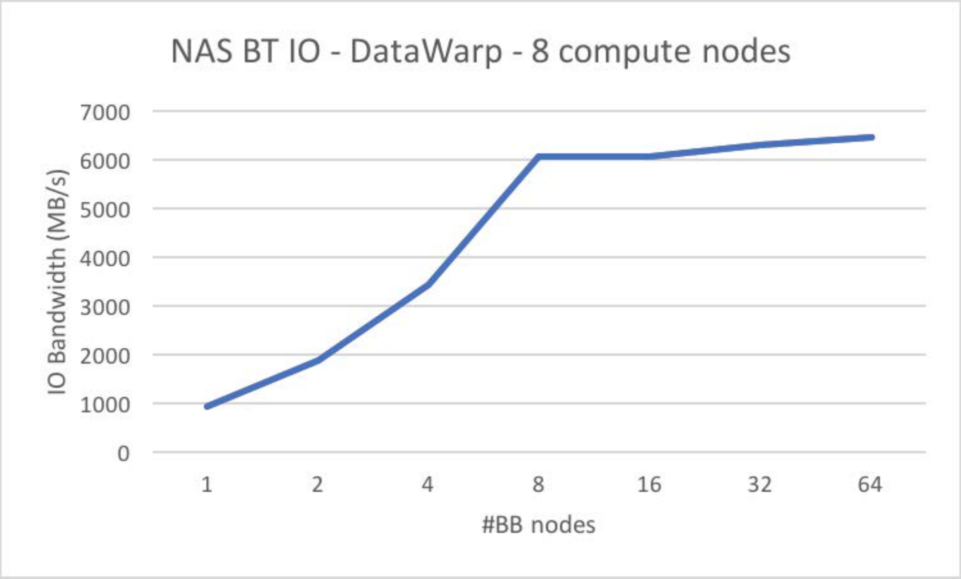


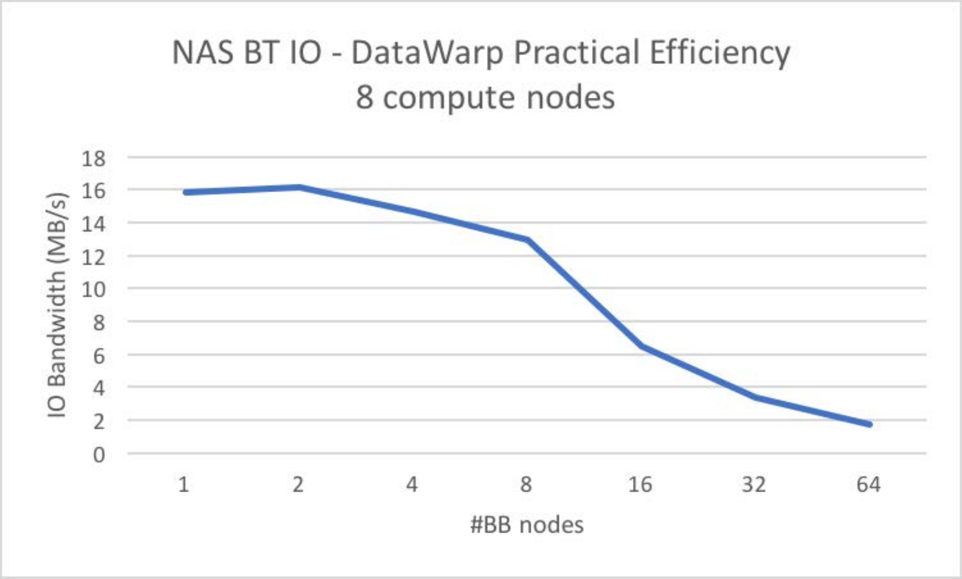
Applications

- 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

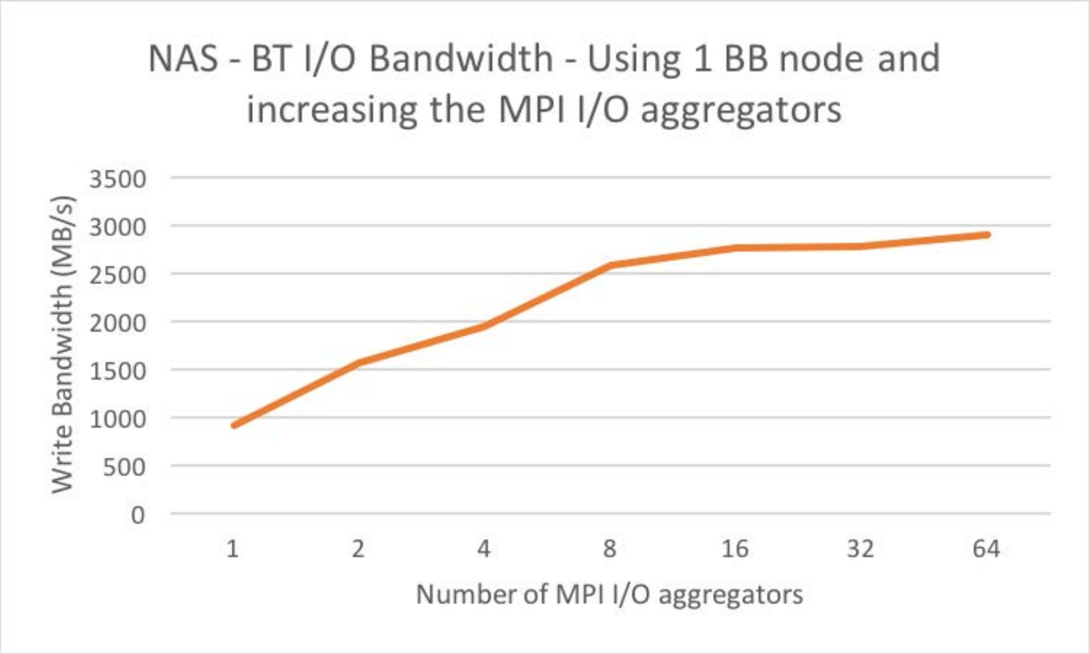
- A user tries to scale his application on Burst Buffer by just increasing the BB nodes and this does not always provide the best results.
- Increasing the BB nodes by 64 times, provide less than 8 times better performance and the practical efficiency is less than 2%!





Collective Buffering – MPI I/O aggregators II

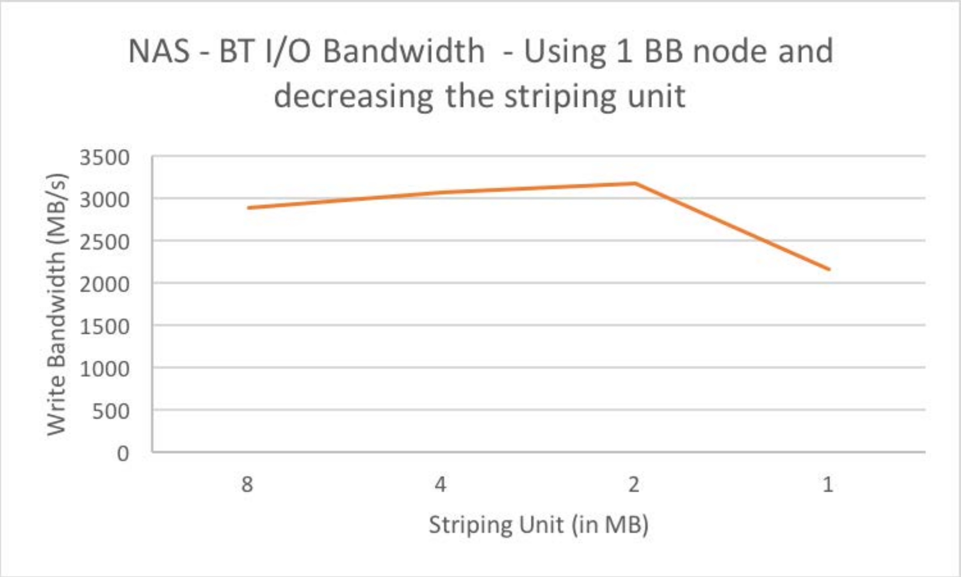
- Using optimized MPI I/O aggregators improved the performance up to 3,11 times on just one BB node.
- We achieved best performance with 64 MPI I/O aggregators



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"**

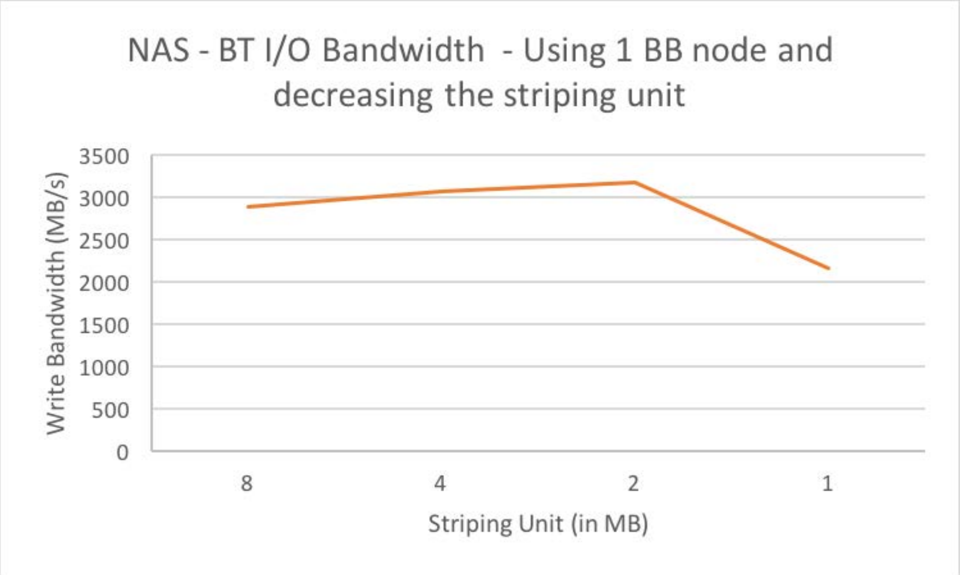
NAS BT I/O - Understanding striping unit

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
```



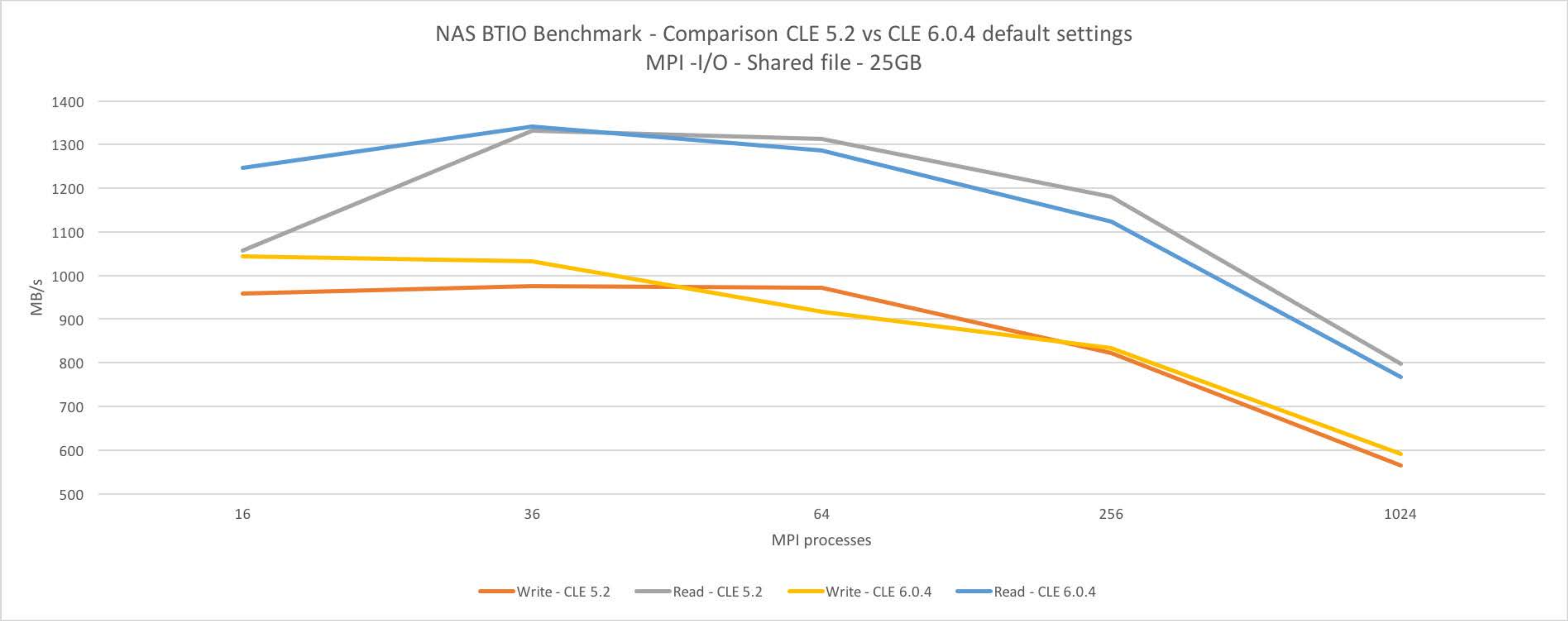
By **decreasing** the **striping unit** by 8 times, the **system writes** were increased by 8 times.

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.

CLE comparison

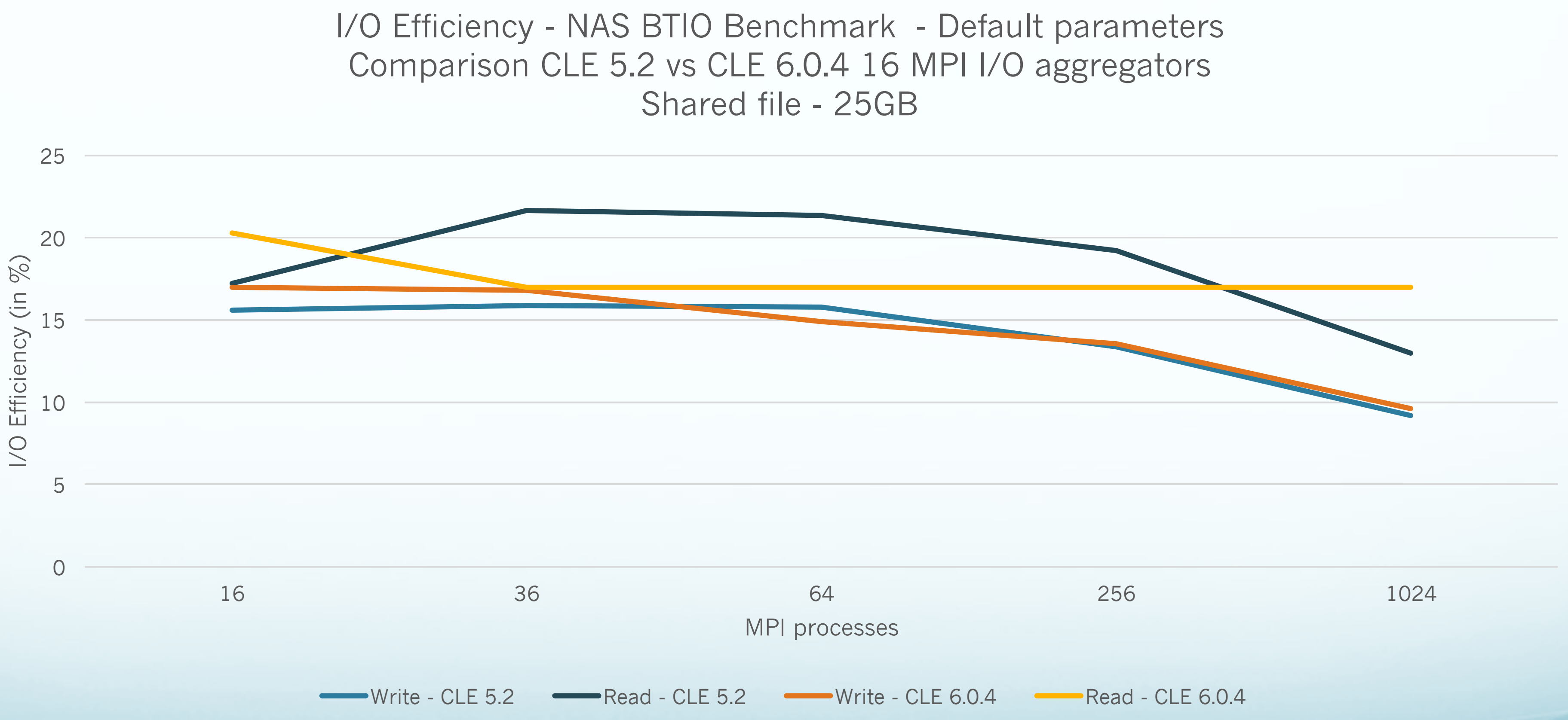
- Cray provides the CLE 6 with new functionalities and performance improvements.
- In the next slides we compare the CLE 5.2 vs 6.0.4
- We use NAS BTIO, with a domain which leads to a shared output file of 25GB.
- We use 1 BB node

CLE 5.2 vs 6.0.4 – default settings

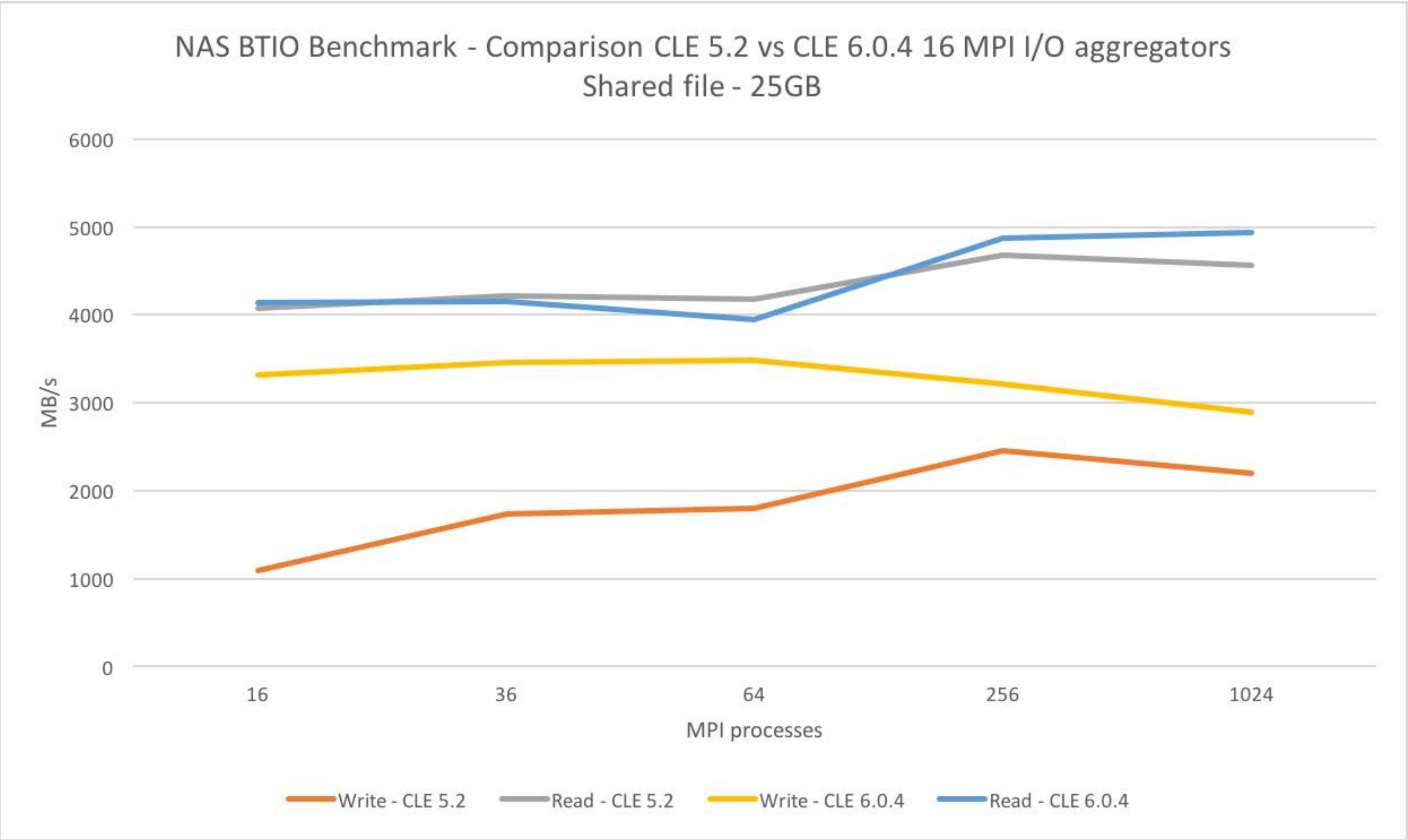


With default settings, there is no significant performance difference between the CLE 5.2 and 6.0.4 in this specific case.

I/O Efficiency – Default parameters

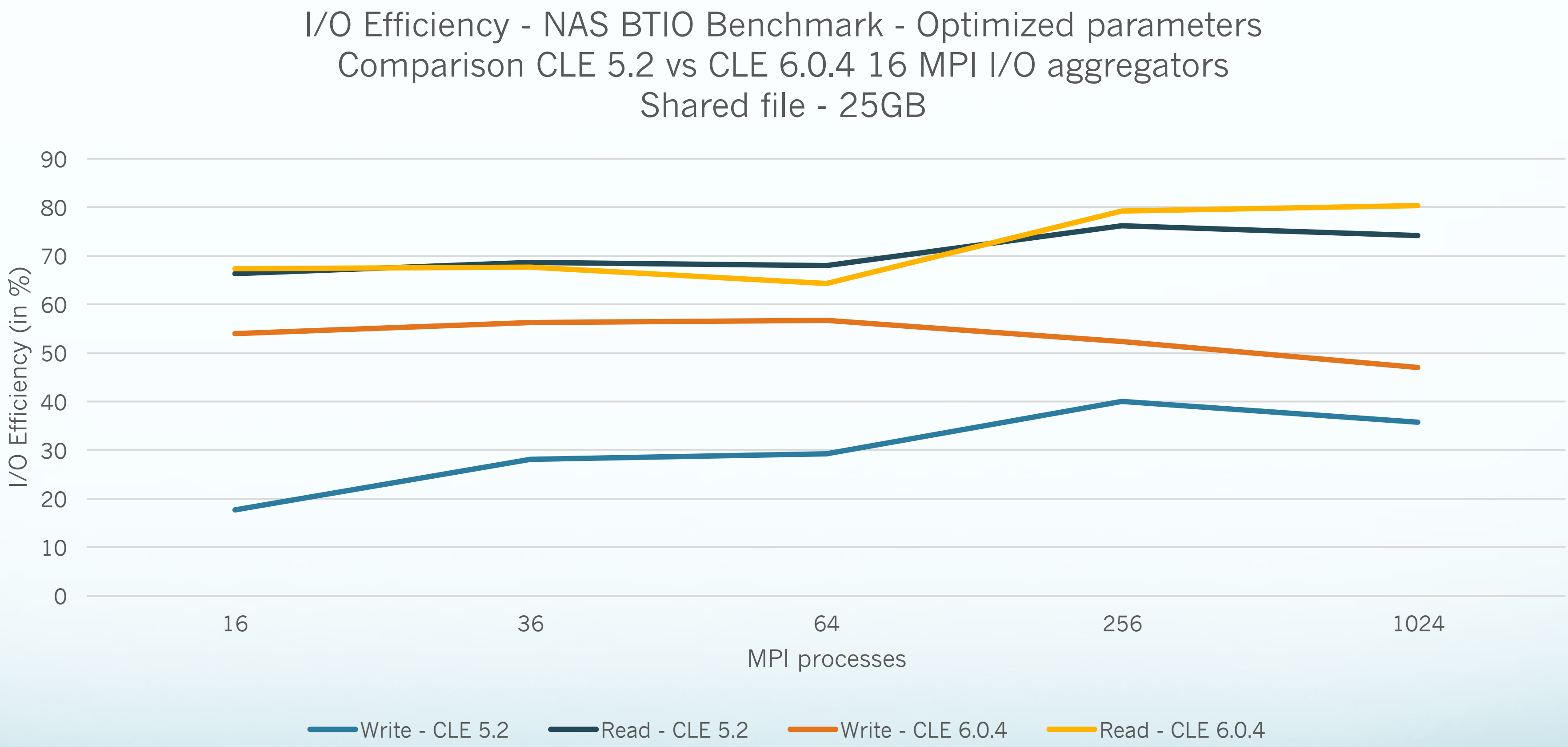


CLE 5.2 vs 6.0.4 – optimized parameters



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



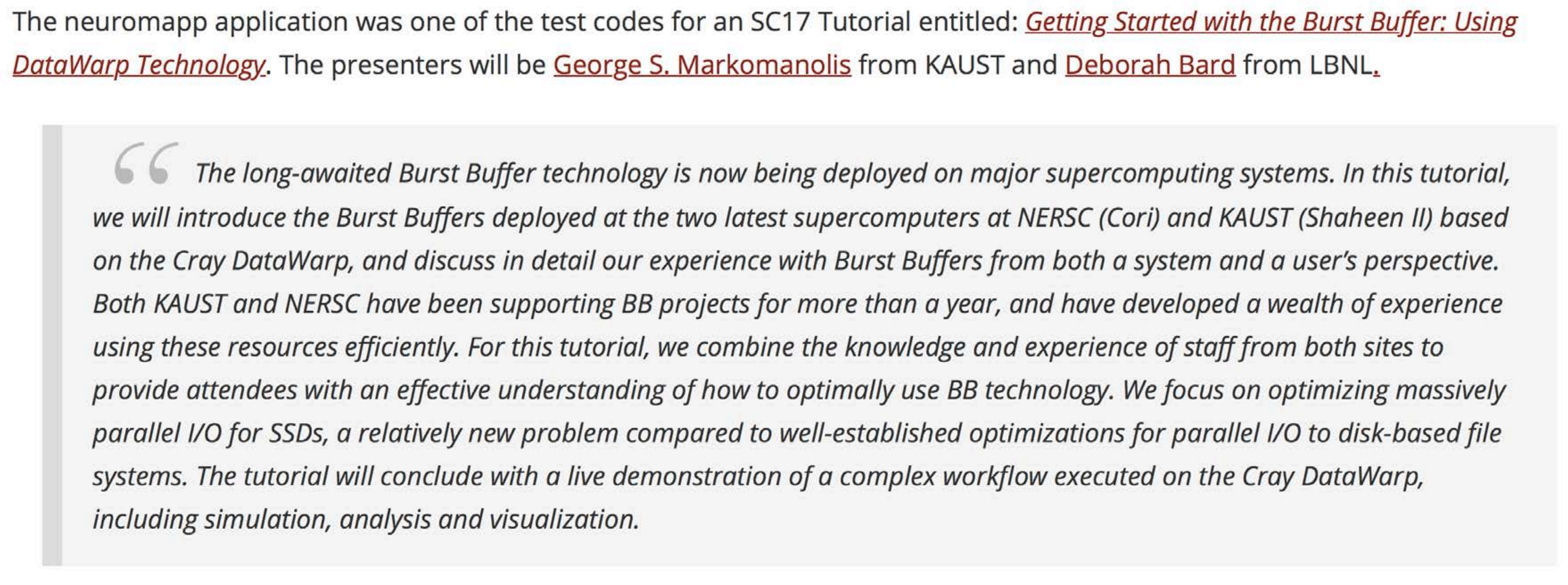
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Study-case Neuromap
Application provided for the SC17 tutorial

Neuromap - Replib

- The Neuronm(ini)app(lication) library reproduces the algorithms of the main software of the Blue Brain 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

International Outreach



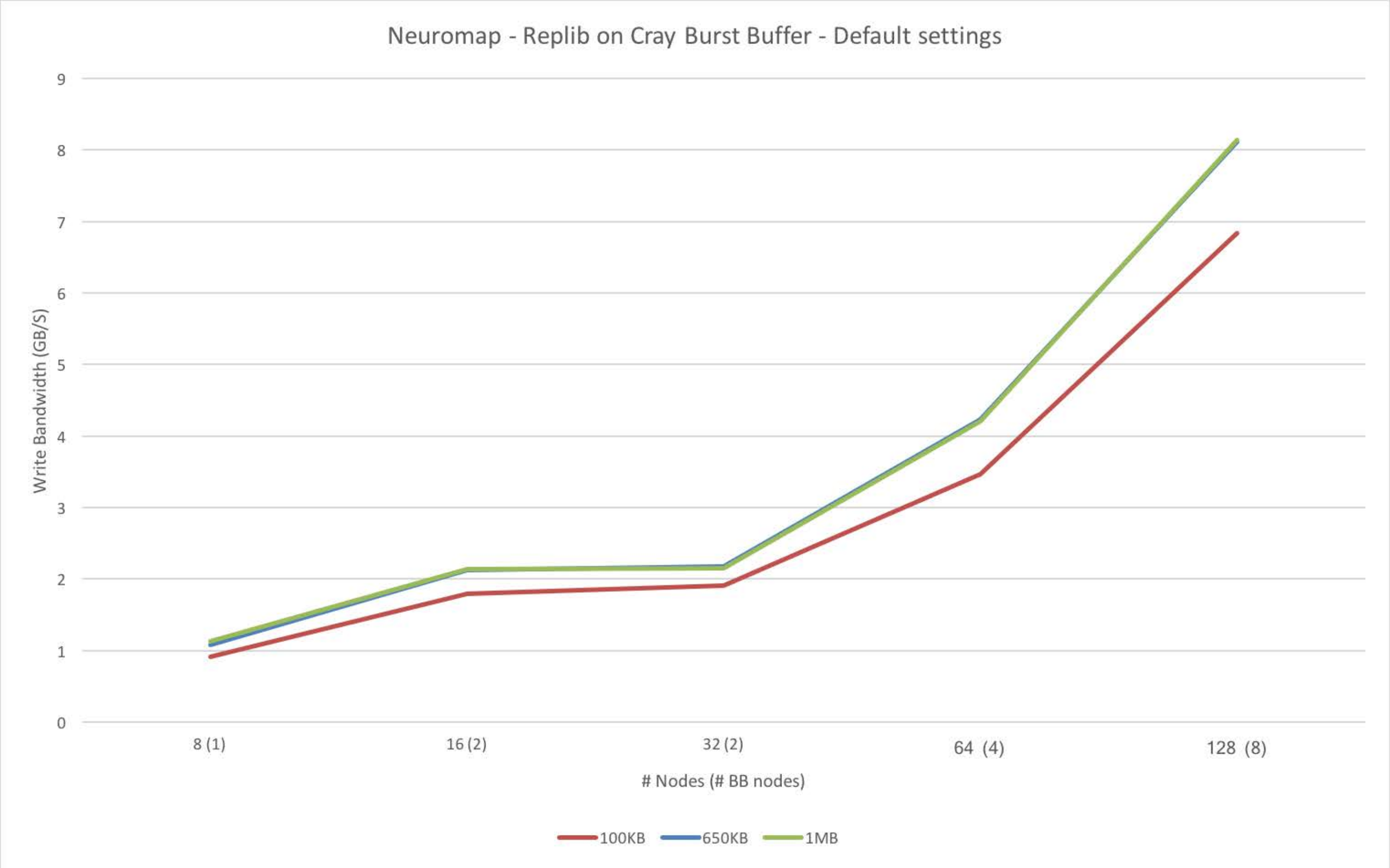
The neuromapp application was one of the test codes for an SC17 Tutorial entitled: *Getting Started with the Burst Buffer: Using DataWarp Technology*. The presenters will be George S. Markomanolis from KAUST and Deborah Bard from LBNL.

> *The long-awaited Burst Buffer technology is now being deployed on major supercomputing systems. In this tutorial, we will introduce the Burst Buffers deployed at the two latest supercomputers at NERSC (Cori) and KAUST (Shaheen II) based on the Cray DataWarp, and discuss in detail our experience with Burst Buffers from both a system and a user's perspective. Both KAUST and NERSC have been supporting BB projects for more than a year, and have developed a wealth of experience using these resources efficiently. For this tutorial, we combine the knowledge and experience of staff from both sites to provide attendees with an effective understanding of how to optimally use BB technology. We focus on optimizing massively parallel I/O for SSDs, a relatively new problem compared to well-established optimizations for parallel I/O to disk-based file systems. The tutorial will conclude with a live demonstration of a complex workflow executed on the Cray DataWarp, including simulation, analysis and visualization.*

https://insidehpc.com/2017/08/video-io-challenges-brain-tissue-simulation/

Neuromap – Replib on Cray Burst Buffer

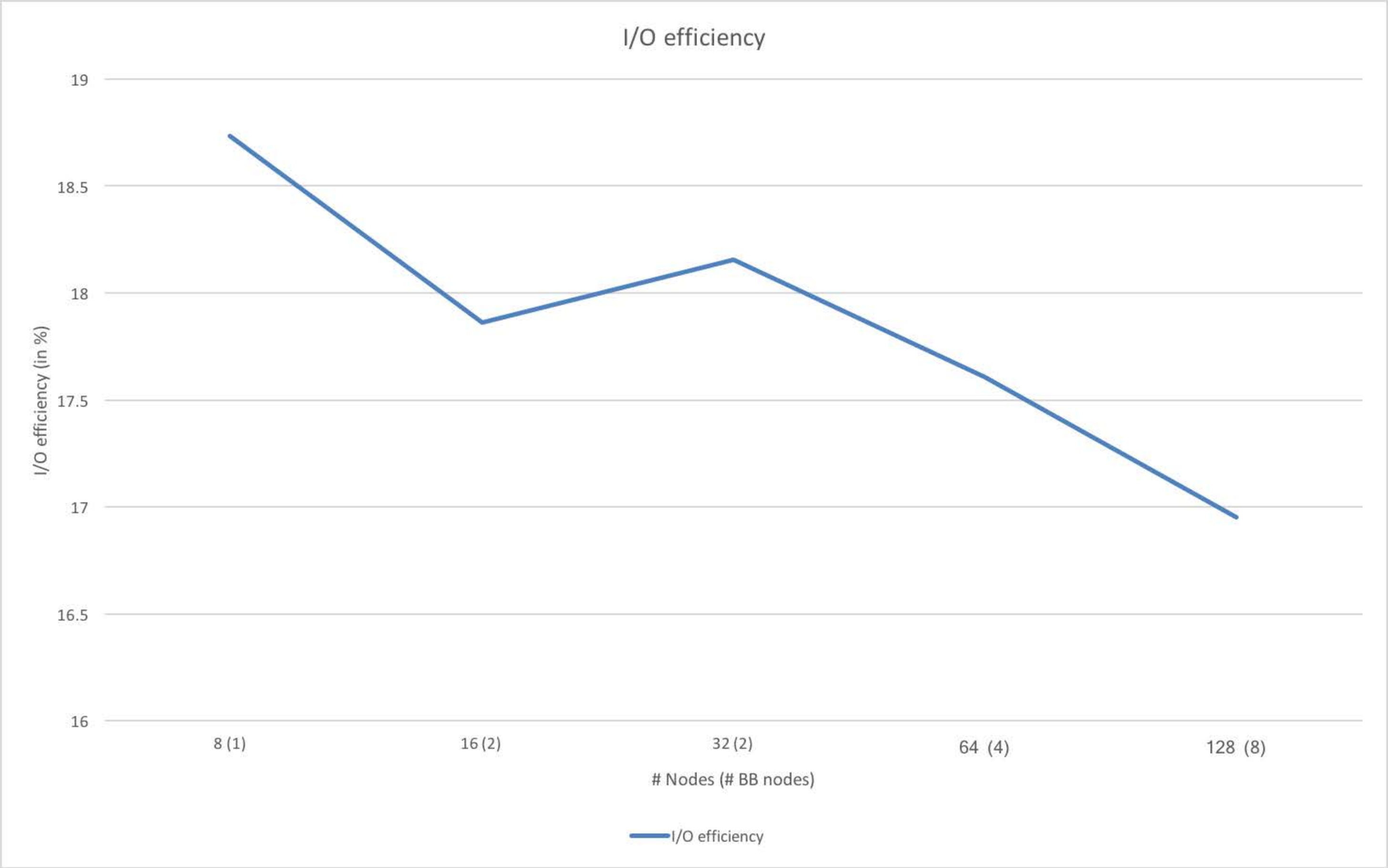
Default parameters



- 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

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.

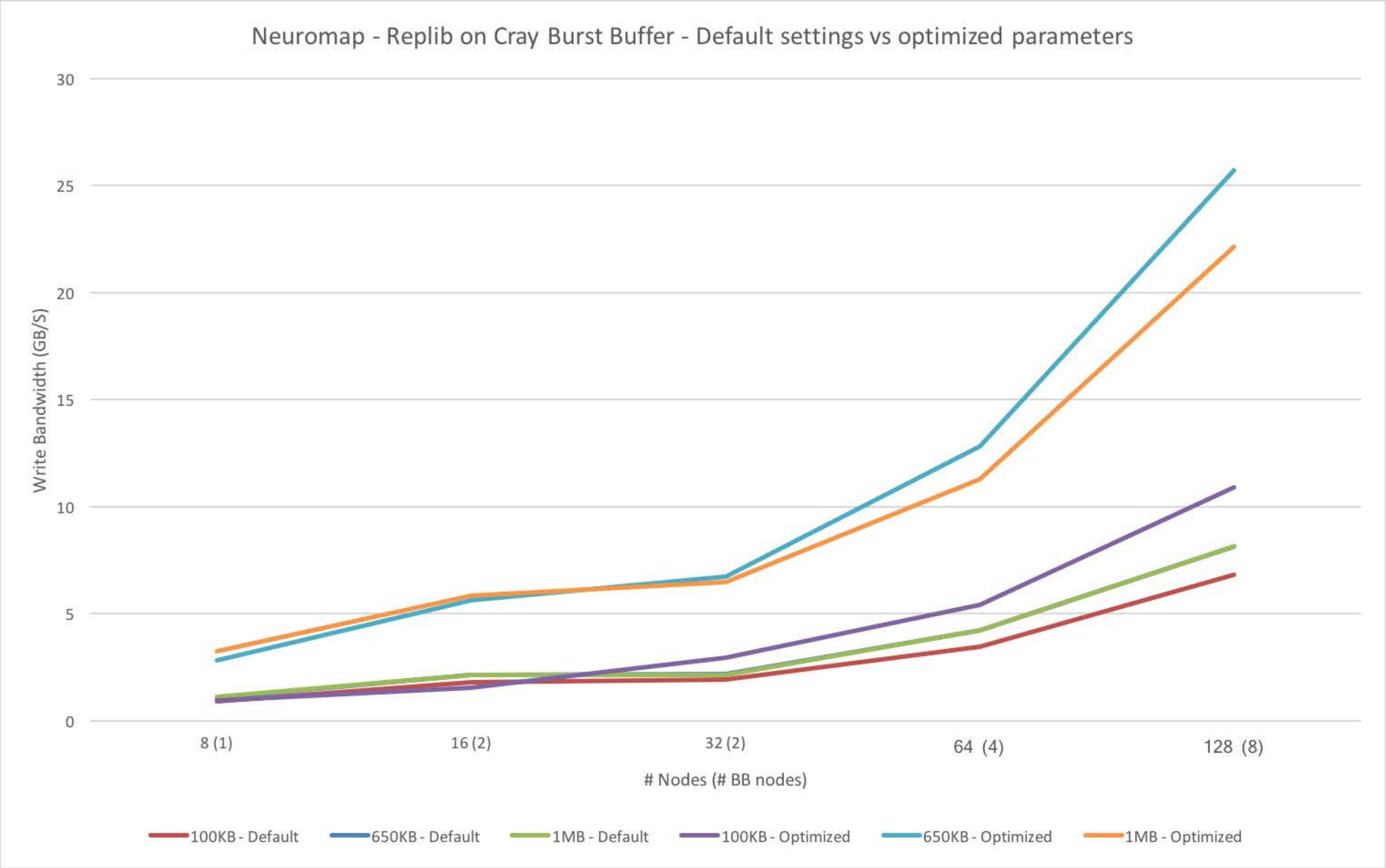
MPI I/O Statistics – Default parameters

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 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.

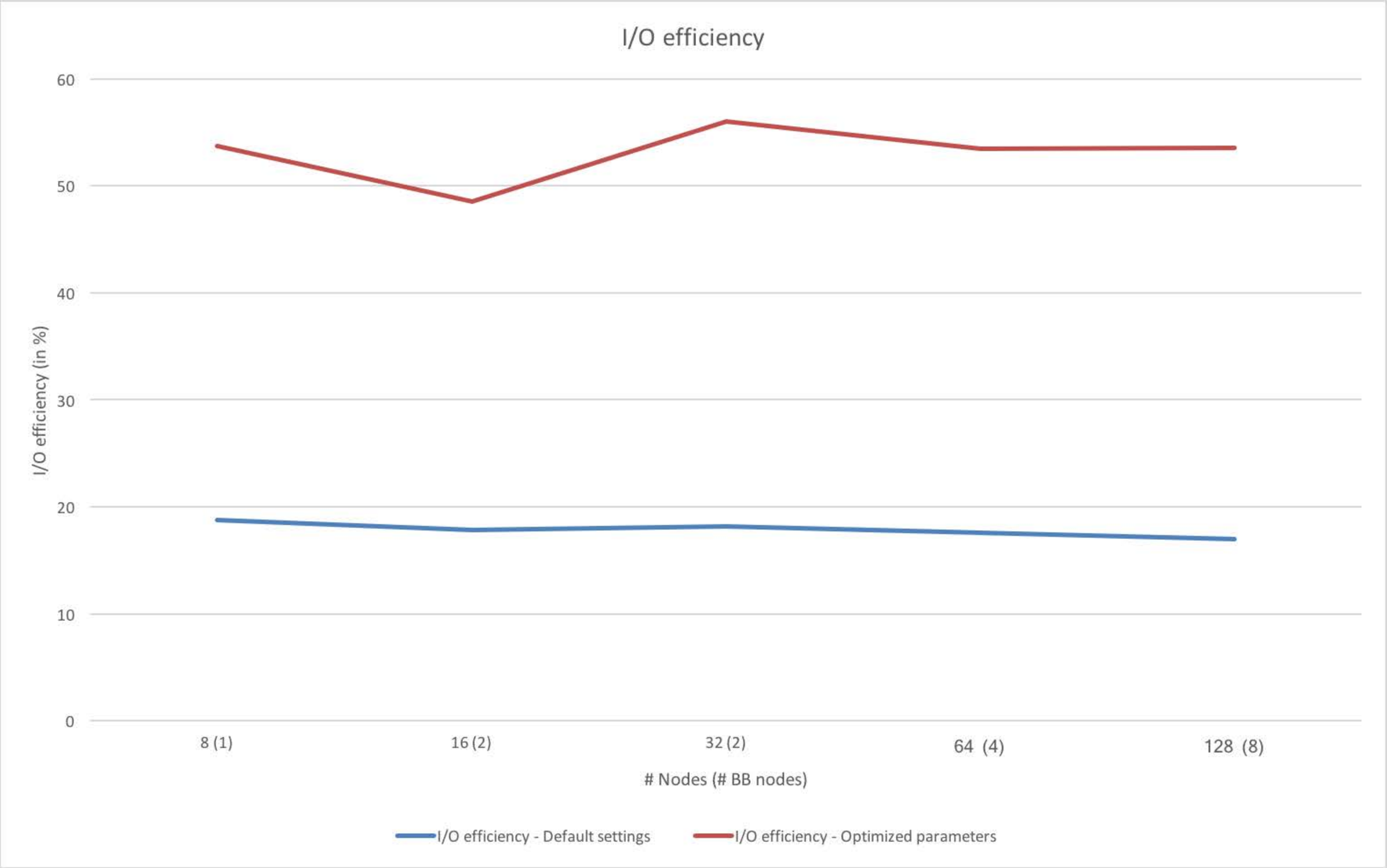
MPI I/O Statistics – Optimized parameters

```
+-----------------------------------------------------------------------+
| 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.

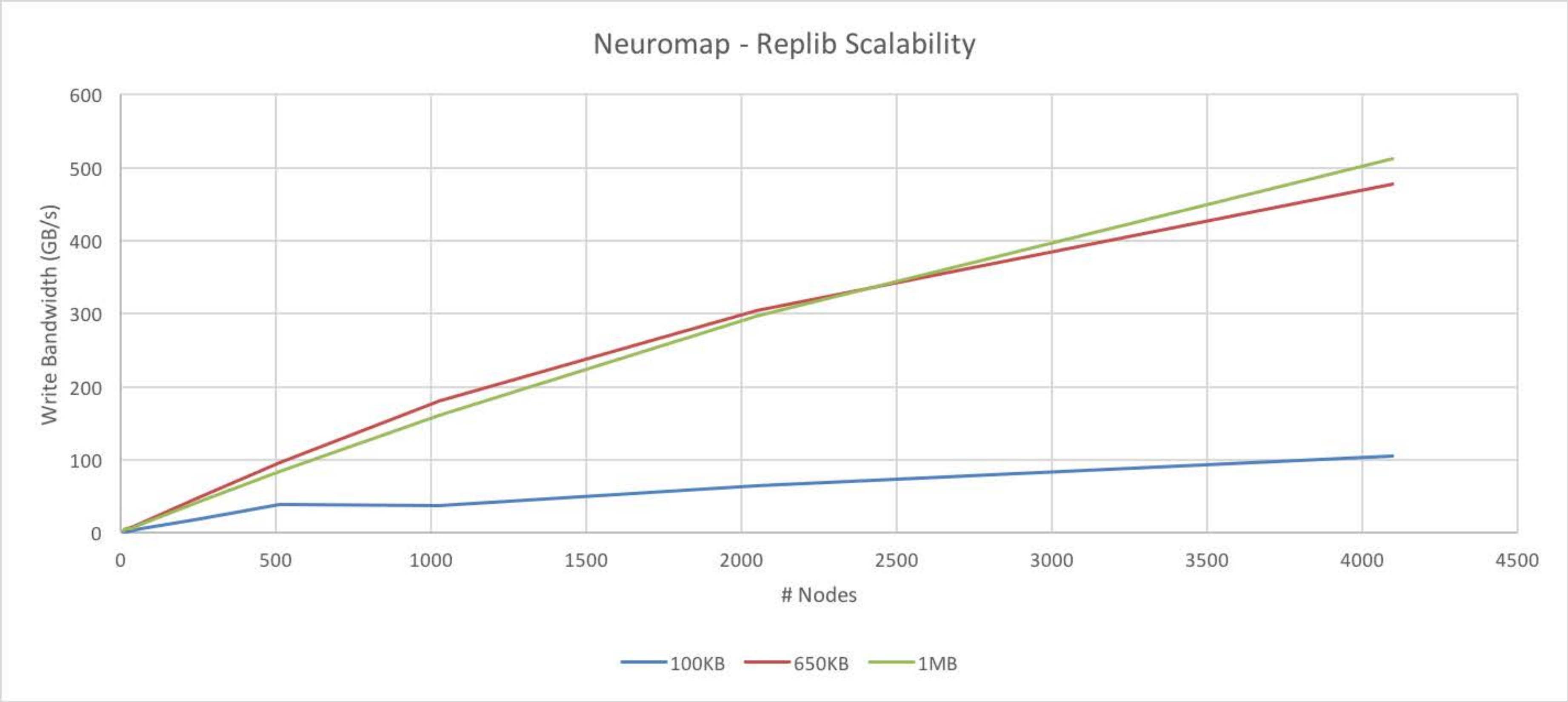
Neuromap – Replib on Cray Burst Buffer

I/O efficiency comparison with optimized parameters



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

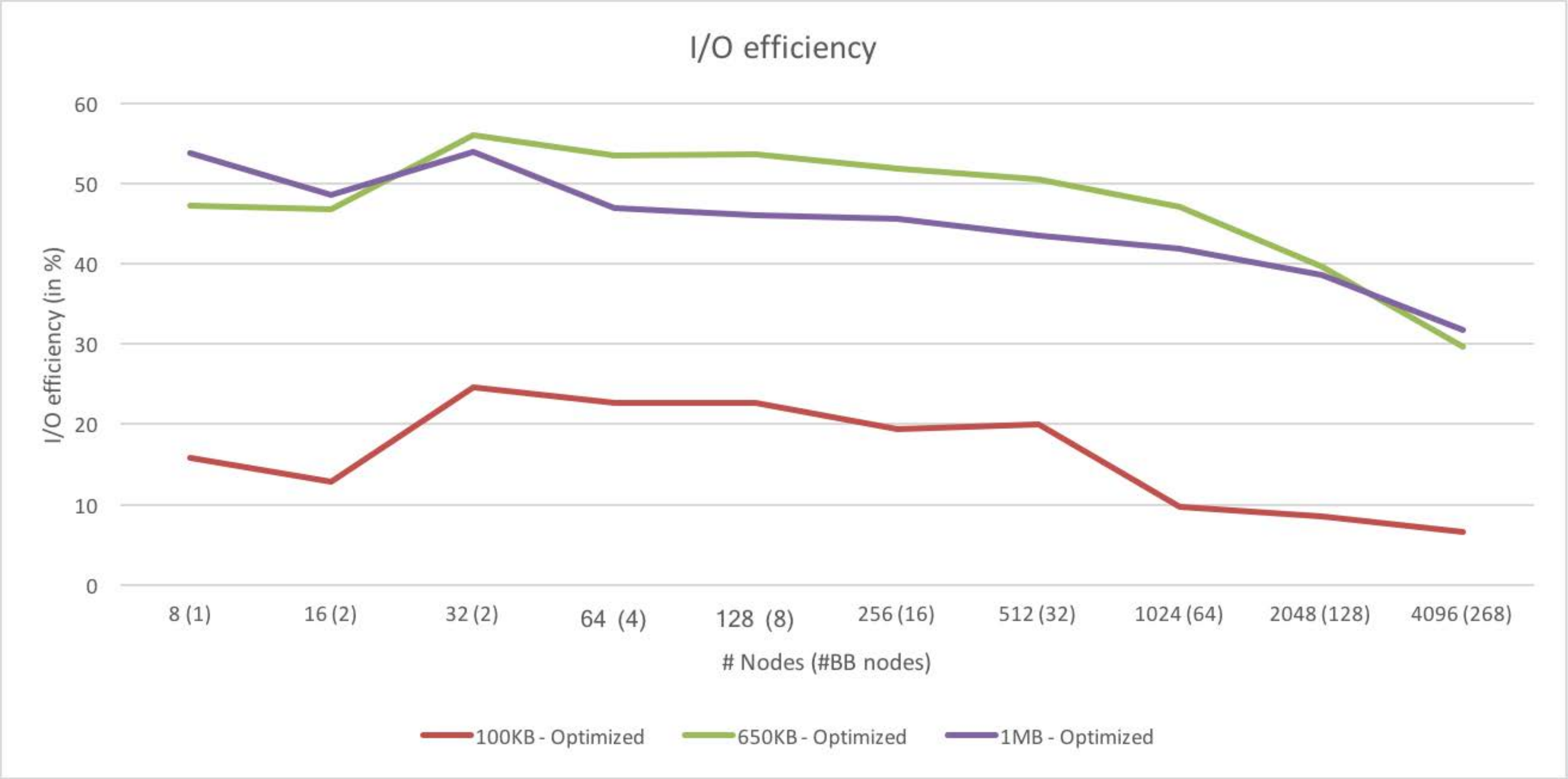
Neuromap – Replib on Cray Burst Buffer



- 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



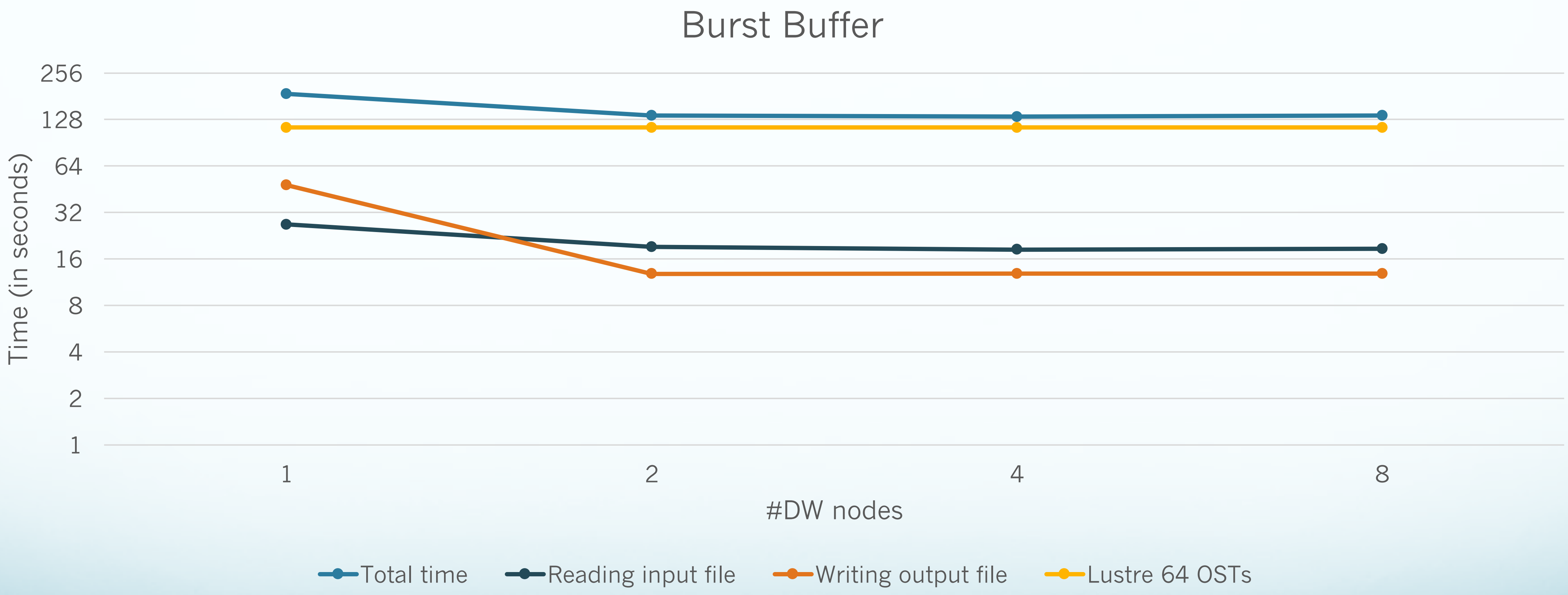
- For the case of chunks of 100KB, the I/O efficiency is between 6,54% and 24,6%
- However, for the cases of 650KB and 1MB, the I/O efficiency varies from 29,7% till 56%

Study-case WRF-CHEM
(on Cori)

WRF-CHEM on Burst Buffer

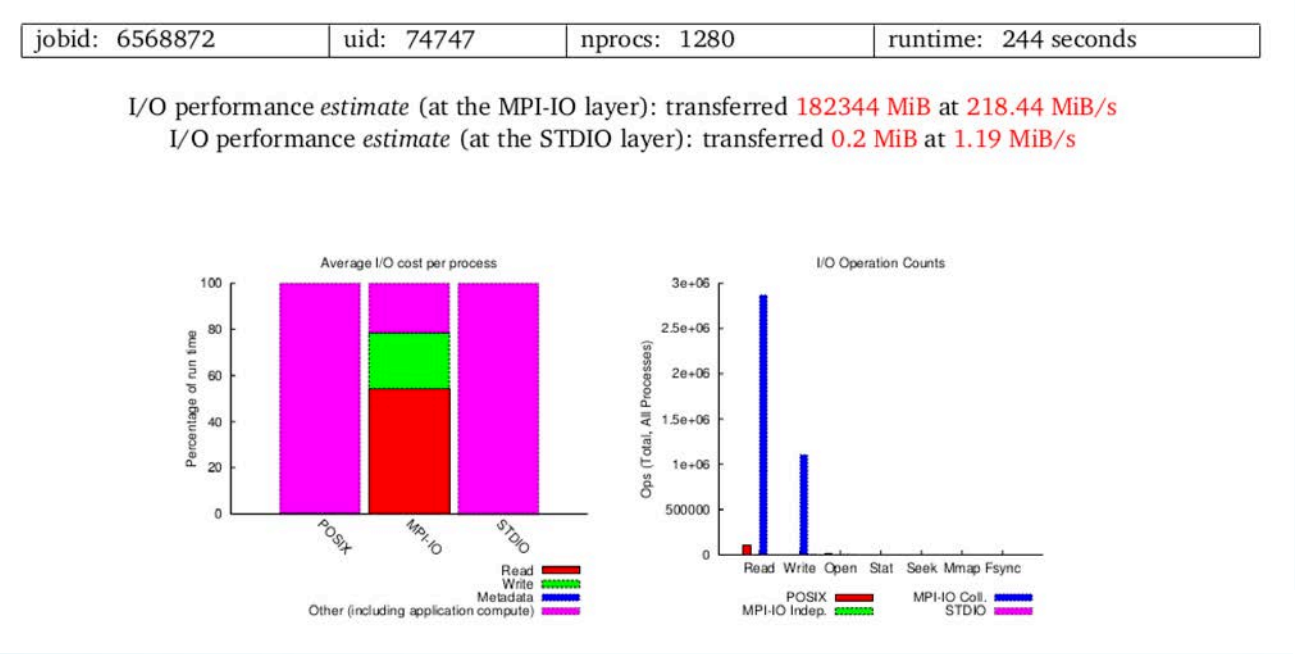
- Weather Research and Forecasting Model coupling with Chemistry
- Small domain: 330 x 275
- Size of input file: 804 MB
- Size of output file: 2.9GB, it is saved every one hour of simulation
- Output file quite small
- For all the WRF-CHEM experiments we use 1280 MPI processes (40 nodes), as this is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

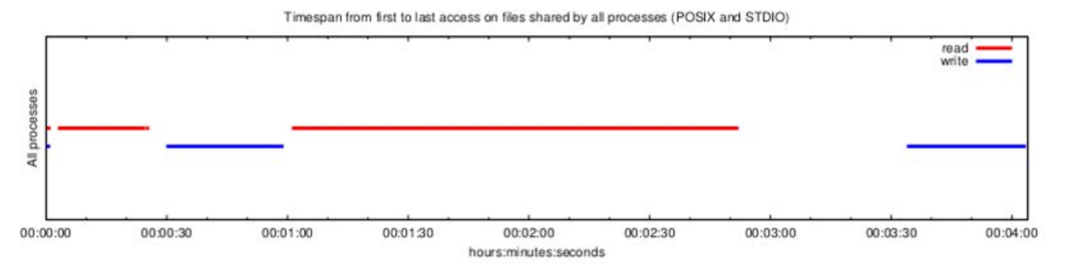
Darshan – WRFChem
1 BB node – default parameters

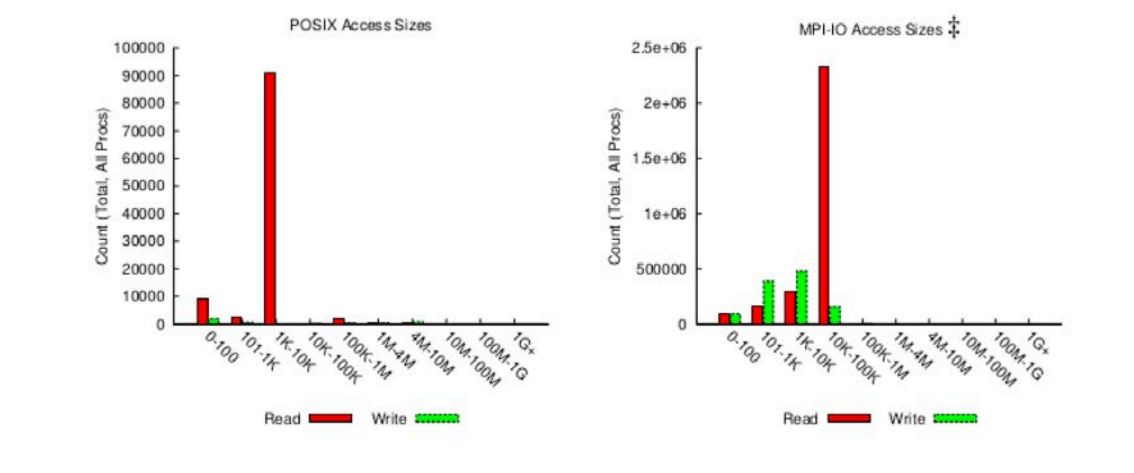


jobid: 6568872	uid: 74747	nprocs: 1280	runtime: 244 seconds

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
```

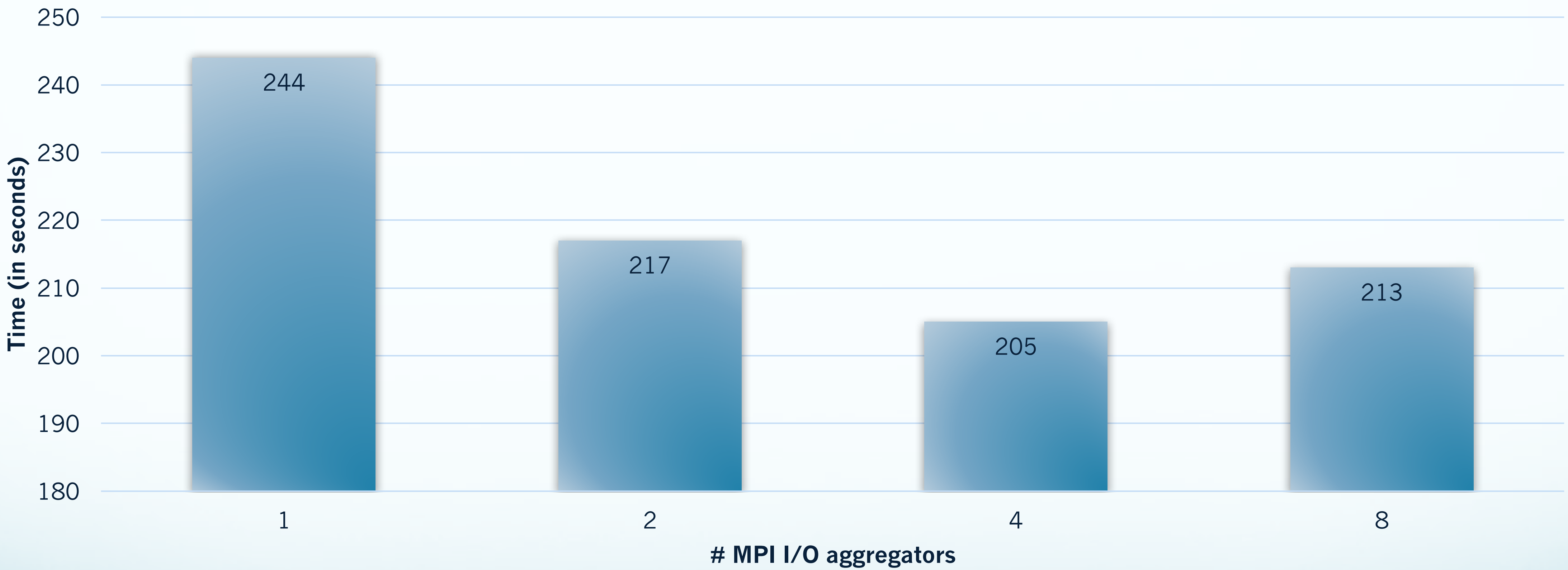
MPI I/O phases Statistics (MPICH_MPIIO_TIMERS=1) II

```
+----------------------------------------------------------------------+
| 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

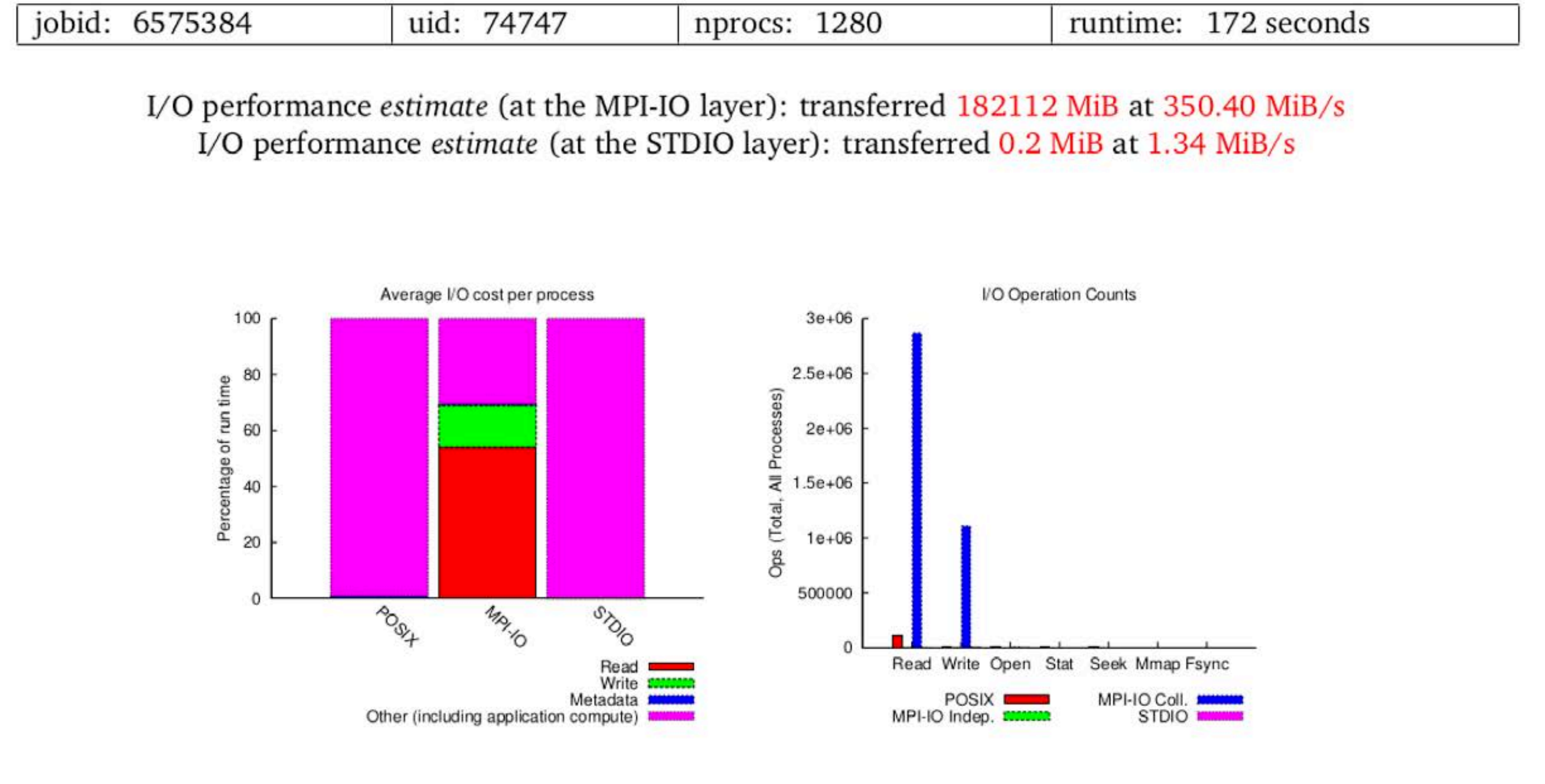
```
+--------------------------------------------------------+
| MPIIO read access patterns for wrfbdy_d01
|   independent reads       = 2
|   collective reads        = 2338560
|   independent readers     = 1
|   aggregators             = 2
|   stripe count            = 1
|   stripe size             = 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.
+--------------------------------------------------------+
```

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)

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

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

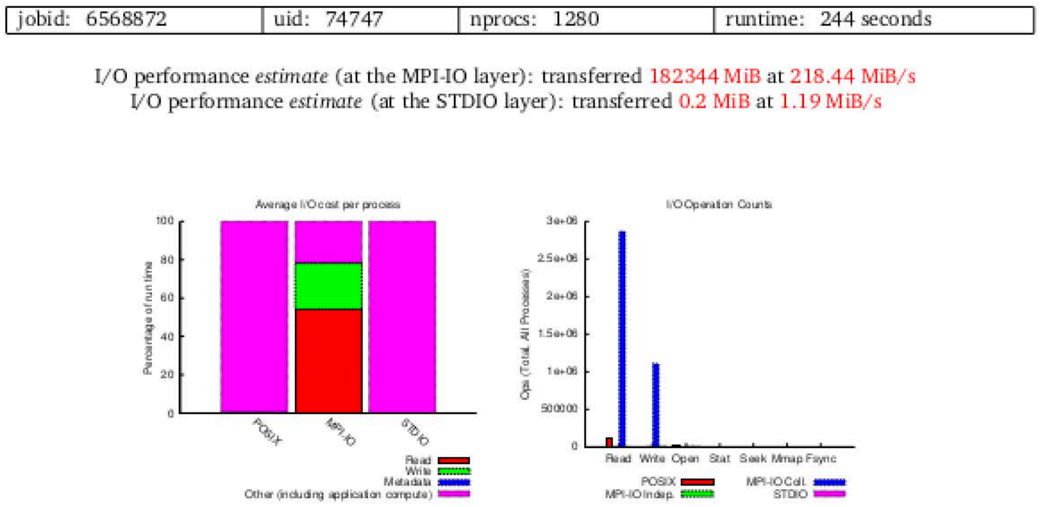
```
+--------------------------------------------------------+
| MPIIO read access patterns for wrfbdy_d01
|   independent reads       = 2
|   collective reads        = 2338560
|   independent readers     = 1
|   aggregators             = 4
|   stripe count            = 1
|   stripe size             = 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.
+--------------------------------------------------------+

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

3% of the reads are striped

Looking for the optimum parameters

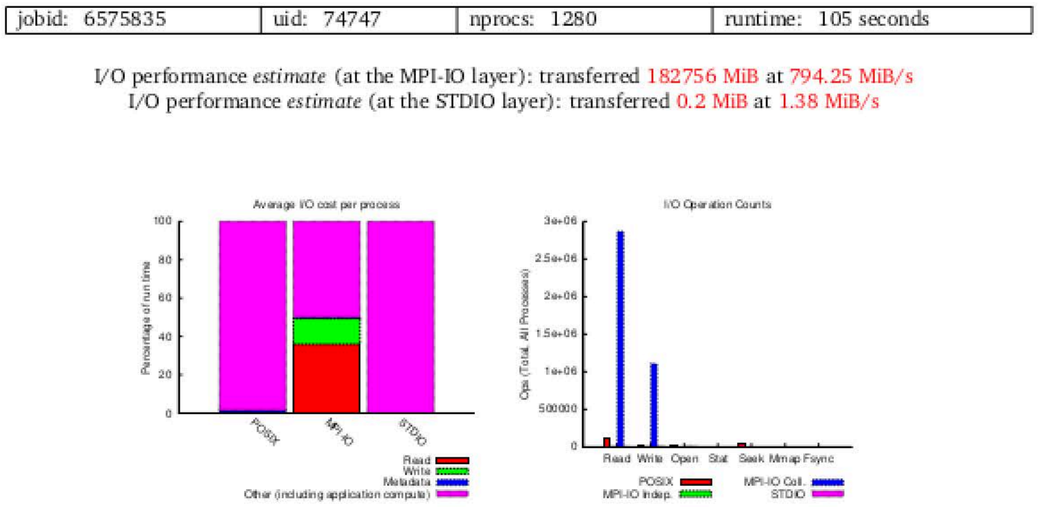
- We executed more experiments and tested various parameters according to the MPI IO statistics data.
- If the performance does not increase while we decrease the value of the striping unit, increase the number of the MPI I/O aggregators.
- While we decrease the value of the striping unit, the number 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 one BB node!

Optimum parameters

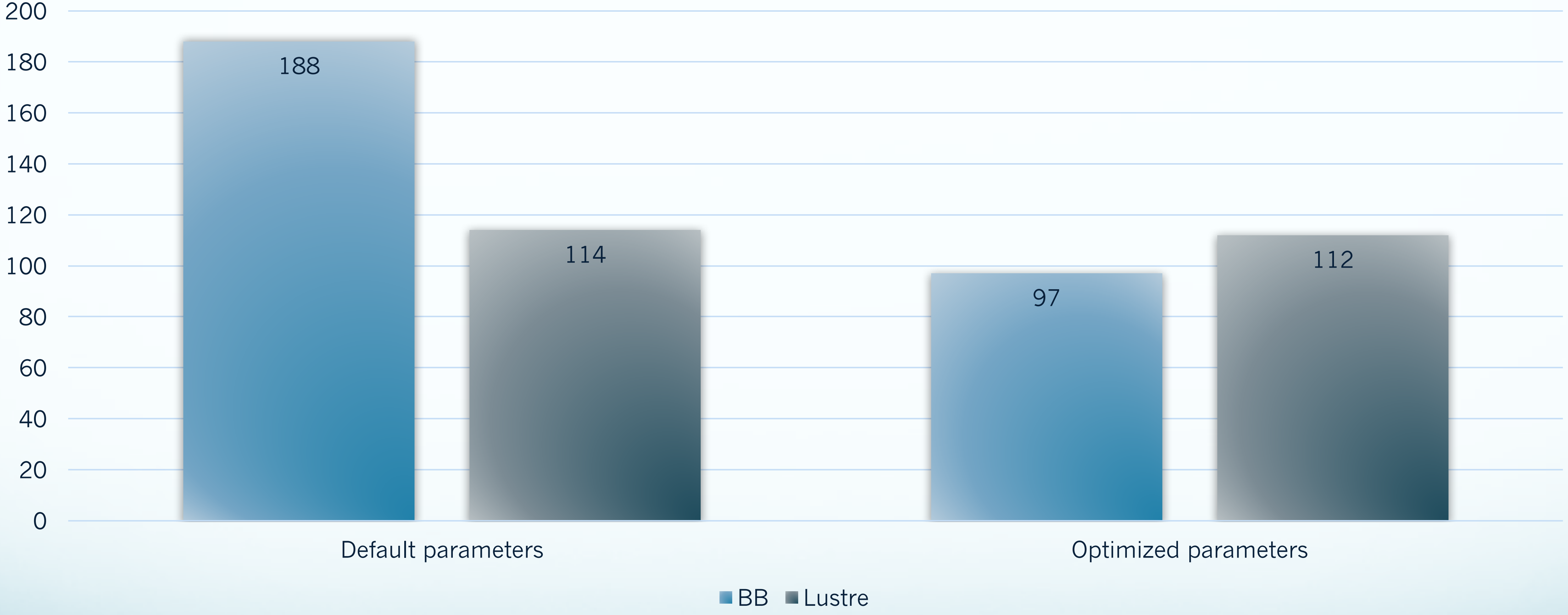
```
MPICH_MPIIO_HINTS="wrfinput*:cb_nodes=16:striping_unit=262144,\
wrfout*:cb_nodes=16:striping_unit=262144,\
wrfb*:cb_nodes=16:striping_unit=50482"
```

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

Comparison between BB and Lustre on Shaheen

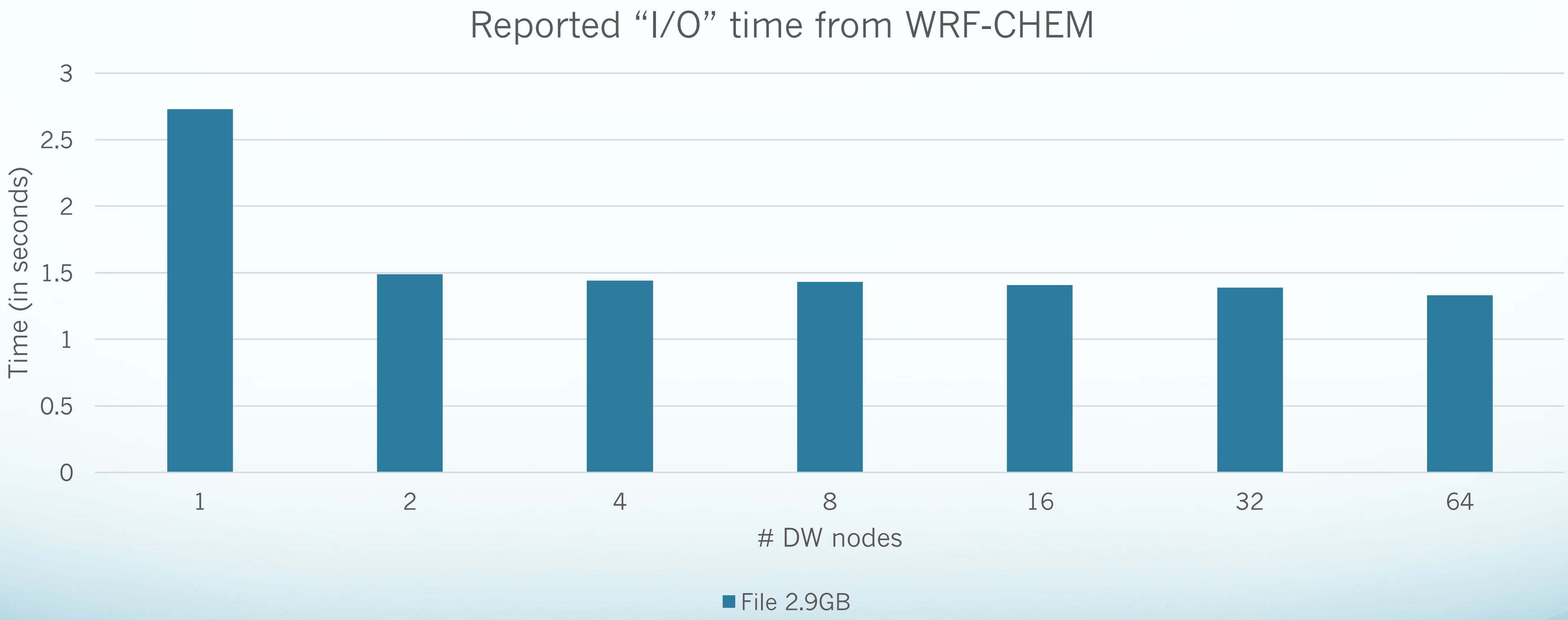


The total execution time on BB is 13.4% faster than Lustre for one hour of simulation of WRF-CHEM. For 24 hours of simulation the execution time on BB is faster than Lustre by 14.8%

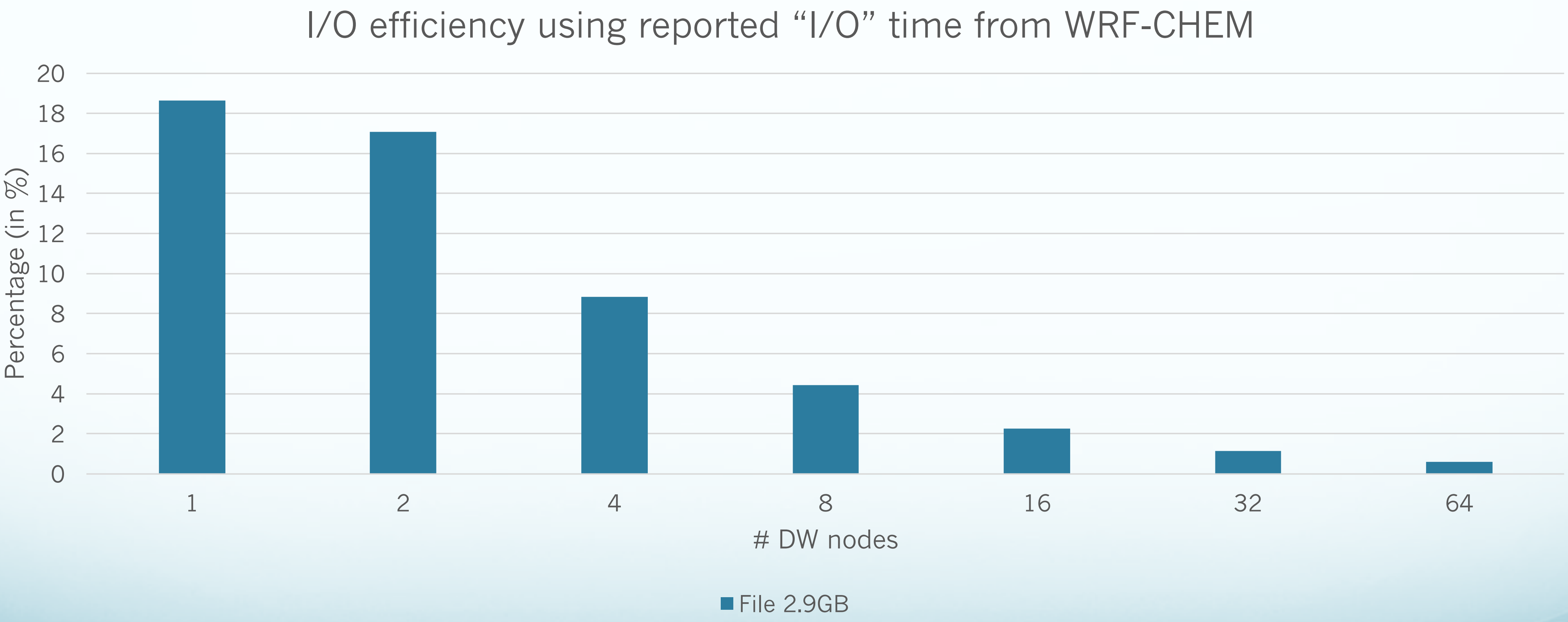
We achieved better performance with BB by using one single BB node in comparison to 64 OSTs of Lustre

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

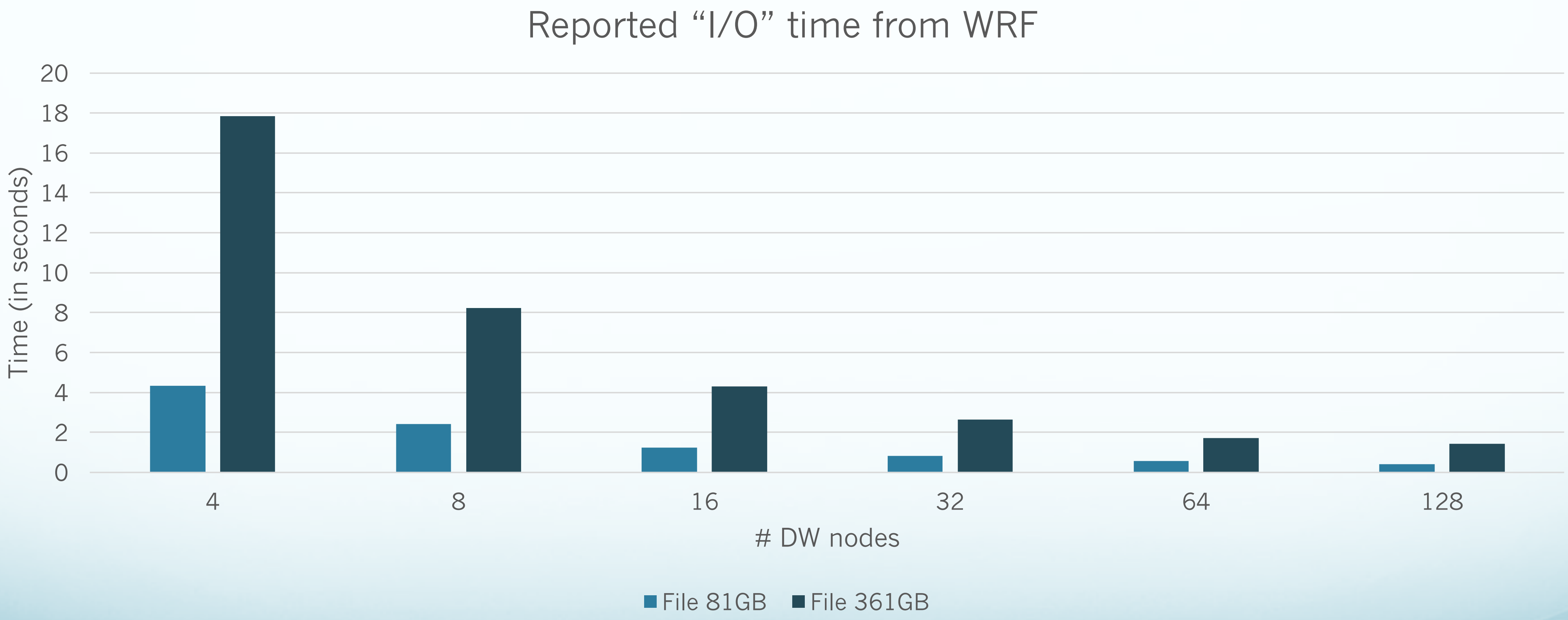
WRF-CHEM – Split output to one file per process



WRF-CHEM – Split output to one file per process II



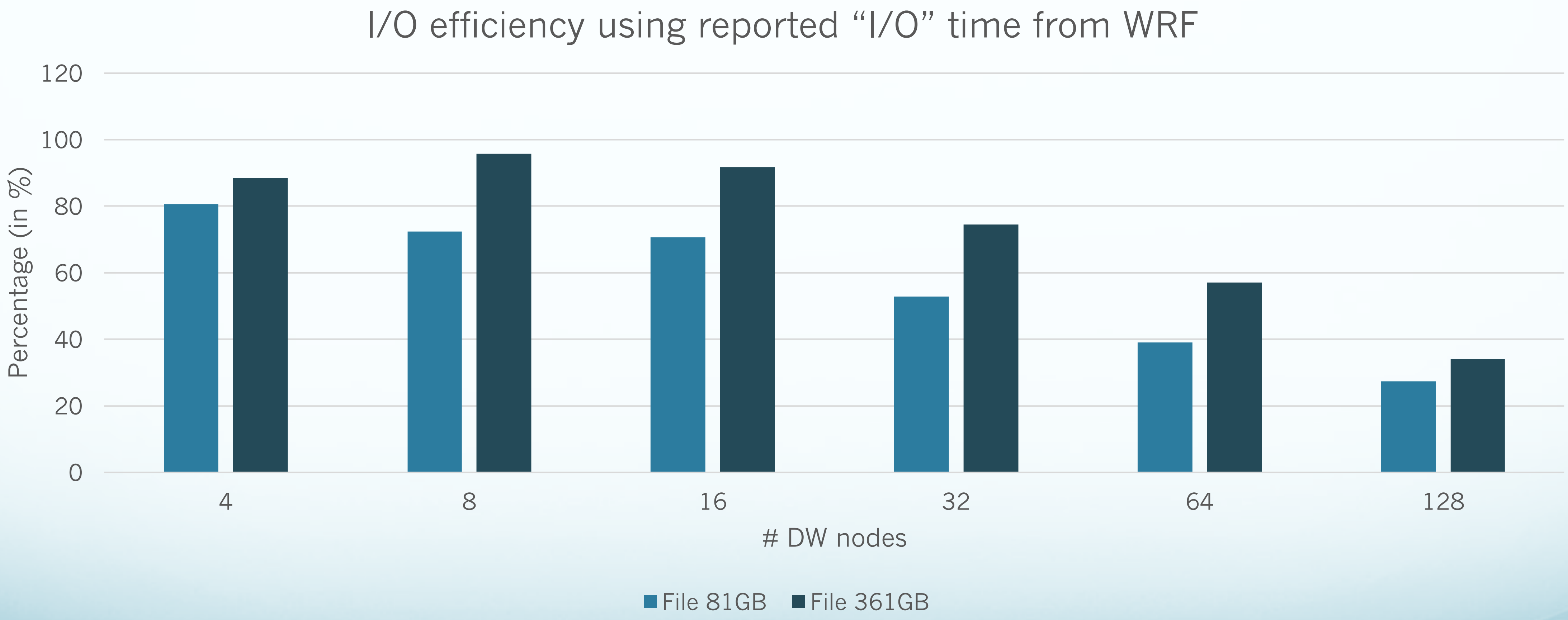
WRF – Split output to one file per process – Large cases



WRF – Alaska domain 1km 6075 x 6075 x 28, 256 compute nodes

King Abdullah University of Science and Technology

WRF – Split output to one file per process II



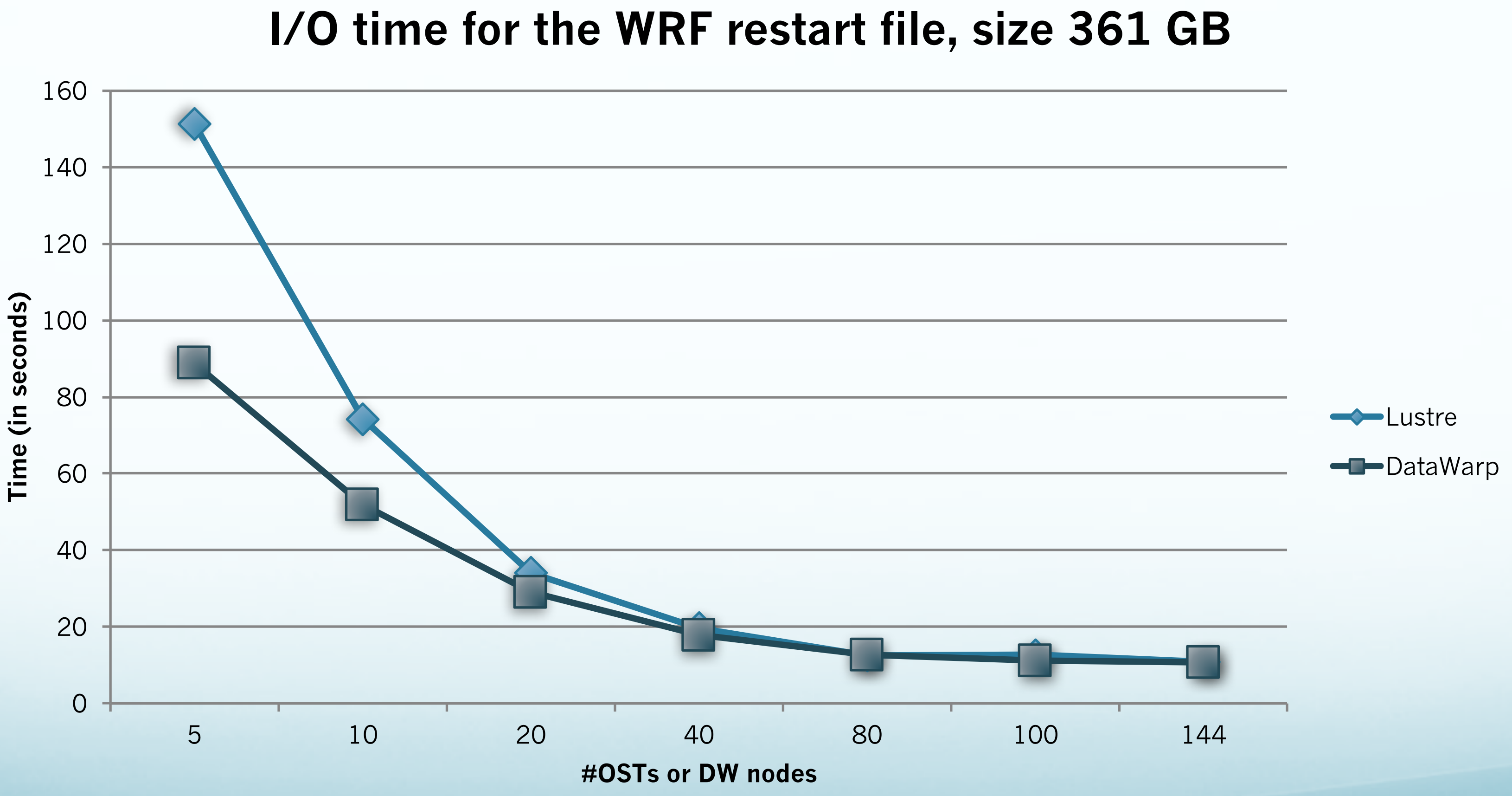
Multiple runs

- Submitting 3 jobs of 20 compute nodes and requesting 64 DW nodes each one
  - used_bb_nodes.sh
    192 BB nodes are used with at least one BB job
    0 BB nodes are used from more than one BB job
  - Variation 2-3%

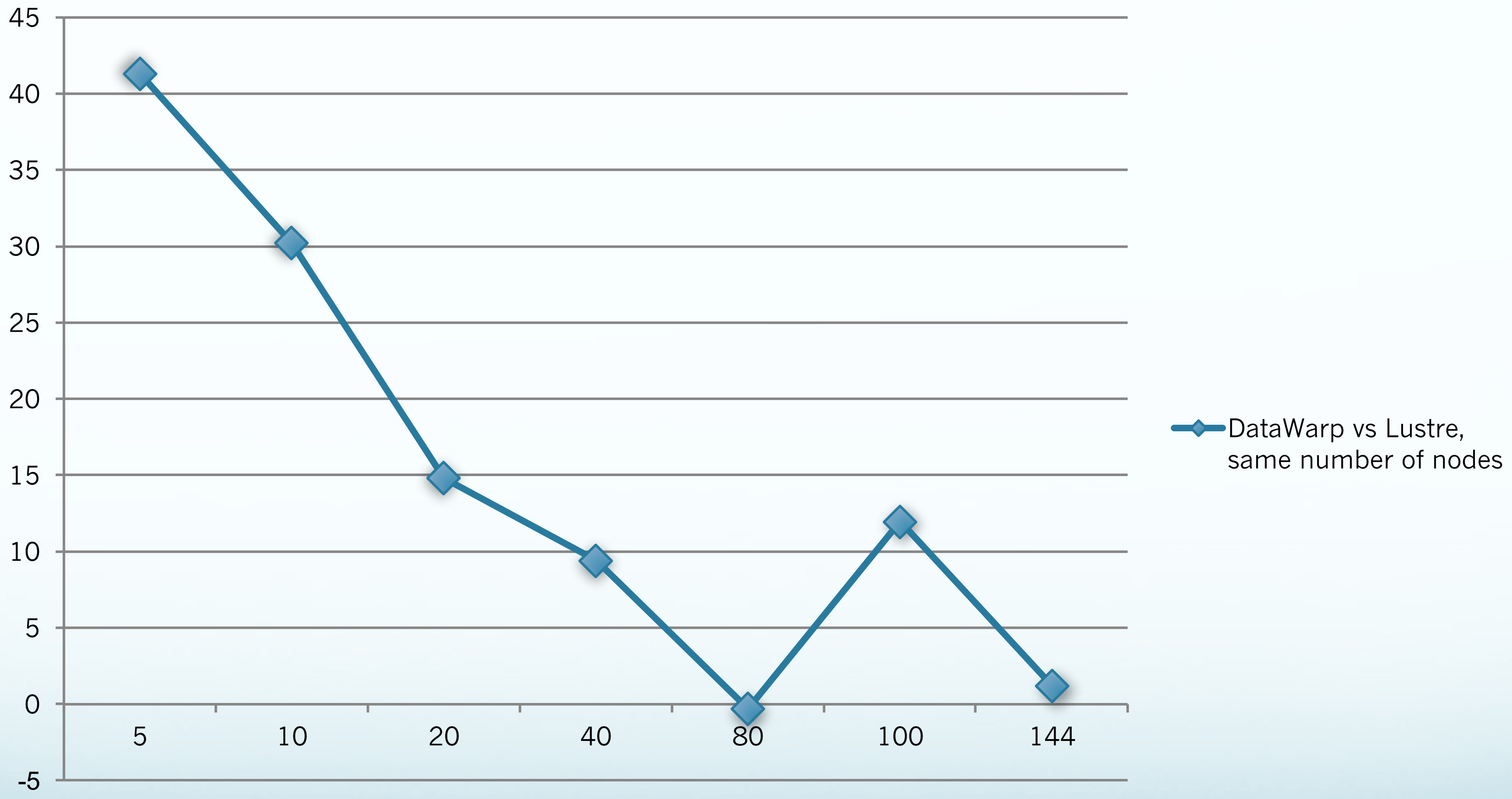
- Variation can be significant when the system is mpre than 60-70% used

DataWarp vs Lustre for same number of nodes (OSTs)

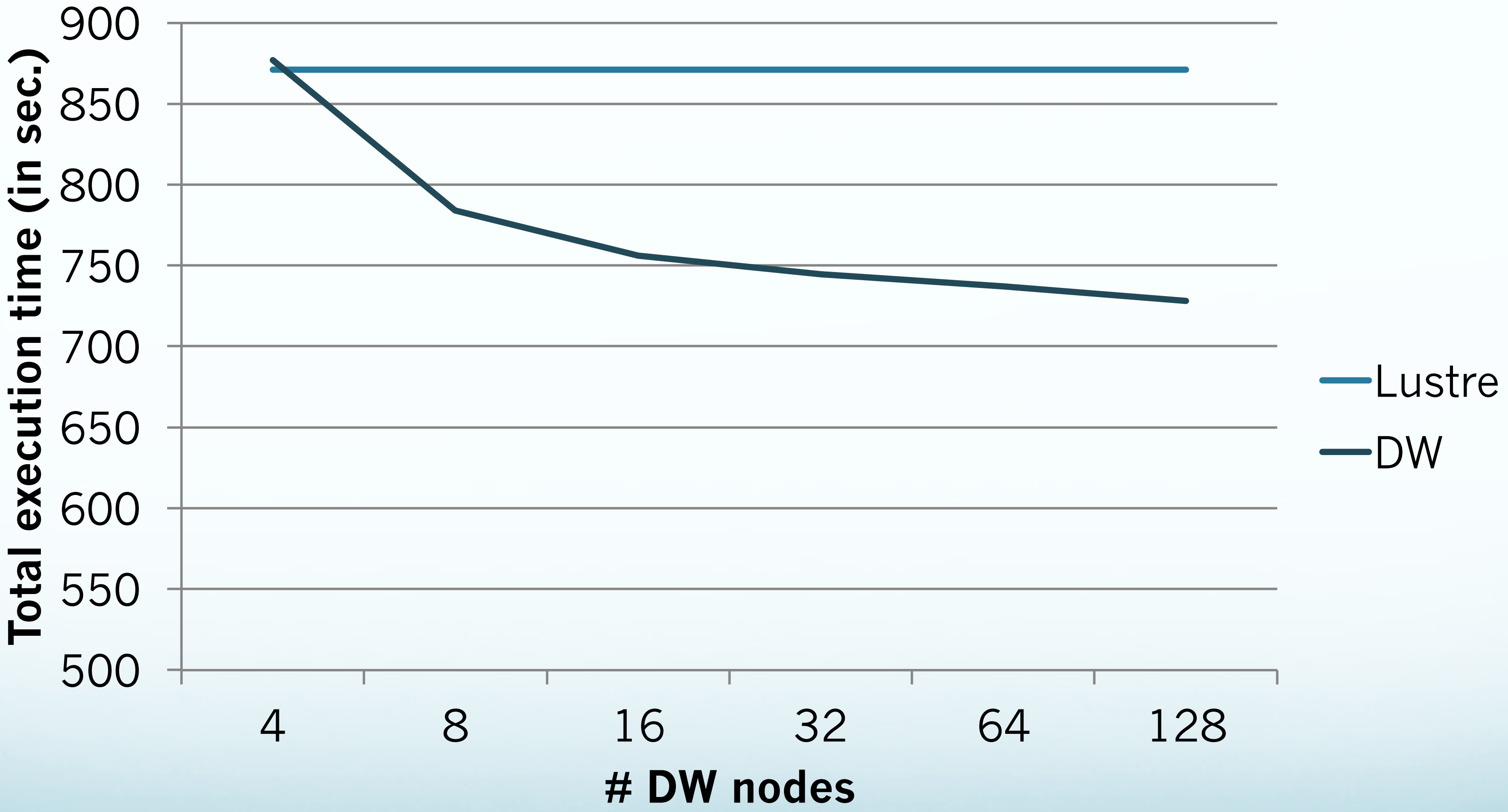
WRF reports I/O time but it includes other functionalities which is beyond I/O



DataWarp vs Lustre, percentage of performance difference



WRF – Lustre vs DW



Study-case Seissol

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.

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
```

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

DataWarp API

- Libatawarp
- **dw_get_stripe_configuration**
- **dw_query_directory_stage**
- **dw_query_file_stage**
- **dw_set_stage_concurrency**
- **dw_stage_file_out**
- **dw_wait_directory_stage**

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

ExpBB: An auto-tuning framework to explore the Performance of Burst Buffer (Cray DataWarp)

Motivation

- Burst Buffer (BB) does not provide the expected performance... or we do not know how to use it?
- A user should be familiar with some technical details and most of them are science-focus researchers.
- We need a tool that a user can execute and extract the optimized parameters for his application and the used domain.

Framework preparation I

- **Fill in the required information in the beginning of the ExPBB script**
- export executable="btio"
- #Declare option for the executable (leave empty if no arguments)

  export arguments="inputbt1.data"
- #Declare the minimum requested Burst Buffer size in GB

  export min_bb_size=1

Framework preparation II

- #Declare stage-in folder, full path

  export stage_in="/project/k01/markomg/development/expbb"

- #Declare stage-out folder, full path

  export stage_out="/project/k01/markomg/back2"

- **The executable is required to have been compiled with the Darshan profiling tool**
- **The framework works for parallel I/O on shared file**

Important 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 or 2
  - Statistics on the actual read/write operations after collective buffering
- export MPICH_MPIIO_HINTS="…"
  - Declare I/O hints

Execution of ExPBB

- If your submission script is called btio.sh, then execute:

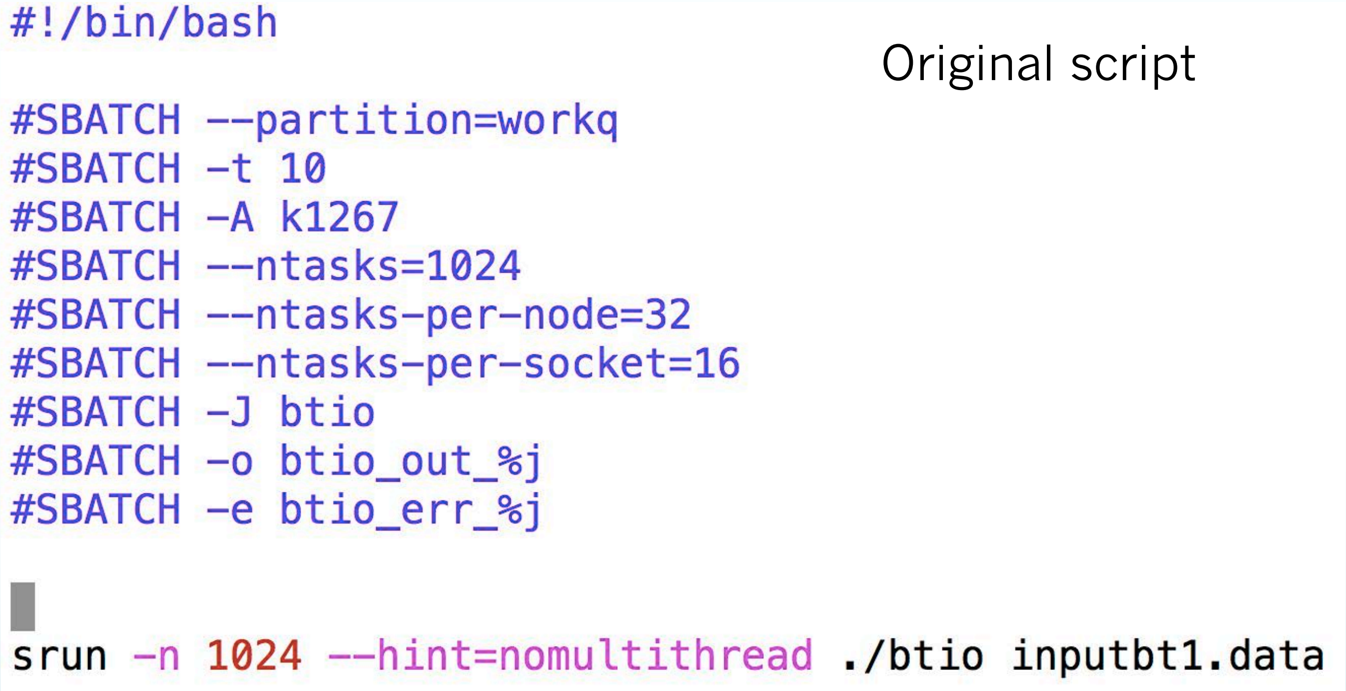
  ./expbb btio.sh

- Then the following will happen:
  - A parser will extract the compute resources from the original script and it will add the corresponding #DW commands in a copy of the original script. From the requested GBs the number of minimum BB nodes will be calculated.
  - The previous important MPI environment variables are added to all the new generated submission scripts
  - Two executions will take place, one on Lustre and one on BB. This happens for two reasons, first to extract the basic execution time for comparison reasons, and second to extract the default striping unit and buffer for each case.

Execution of ExPBB II

- Then the tool will create a new submissions script depending on the number of the BB nodes, for example on Shaheen II we have 268 BB nodes, if we need 4 BB nodes minimum, then there will be scripts for 4, 8, 16, 32, 64, 128, and 256 BB nodes.
- Each of the script includes extra code before and after the *srun* command, where loops change the values of the parameters, where their range depends on the default values extracted on the first BB execution.
- After the srun command a parser is called, where it reads the Darshan performance data and acts accordingly
- The first script will be submitted with the minimum requested nodes and it will start investigating the results.
- All the results will be written in txt files that are easily accessible

ExPBB example – Original script

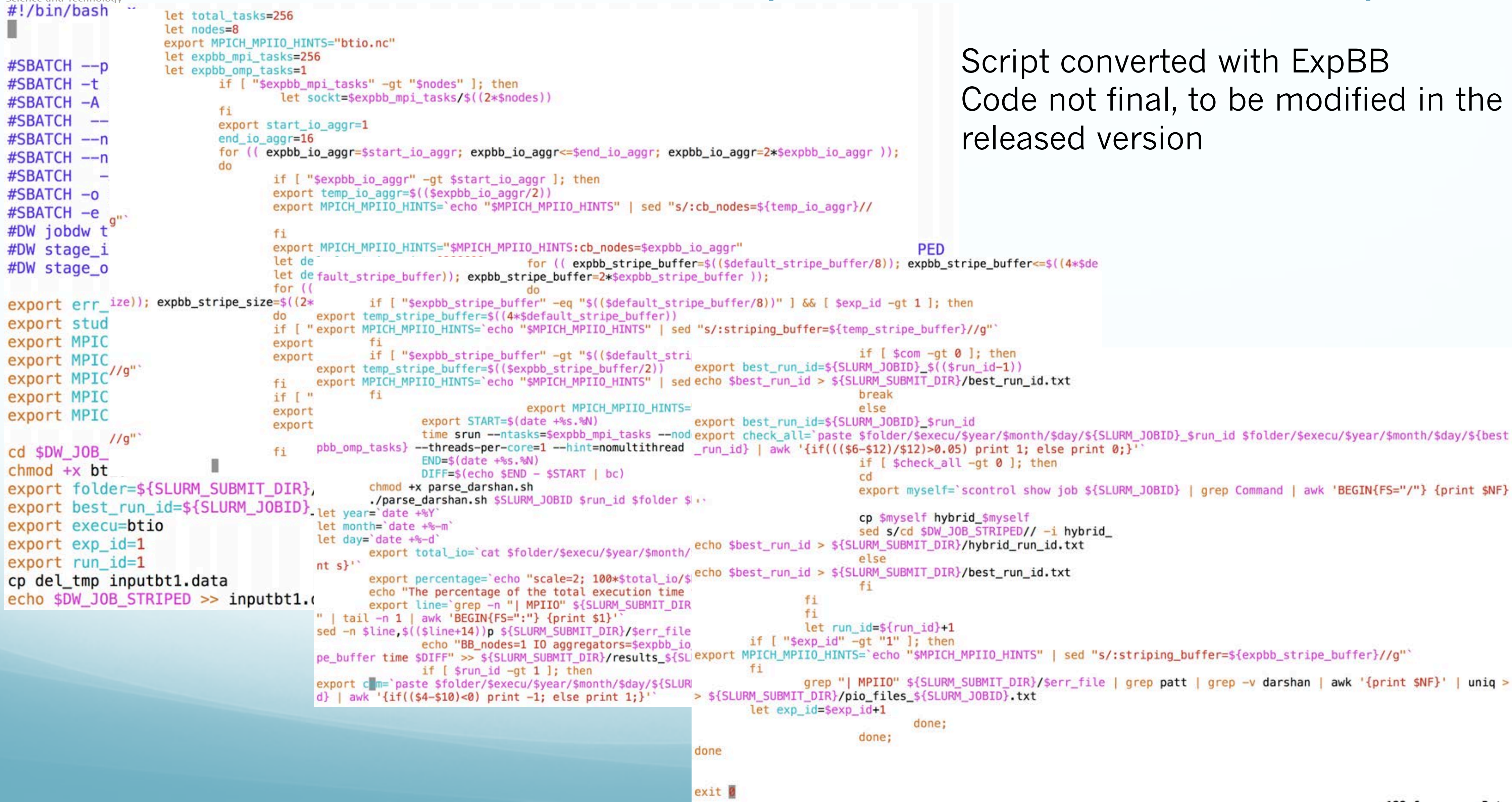


Original script

```
#!/bin/bash

#SBATCH --partition=workq
#SBATCH -t 10
#SBATCH -A k1267
#SBATCH --ntasks=1024
#SBATCH --ntasks-per-node=32
#SBATCH --ntasks-per-socket=16
#SBATCH -J btio
#SBATCH -o btio_out_%j
#SBATCH -e btio_err_%j


srun -n 1024 --hint=nomultithread ./btio inputbt1.data
```

ExPBB example – Converted script

Script converted with ExpBB
Code not final, to be modified in the released version



```
#!/bin/bash
▮

#SBATCH --p
#SBATCH -t
#SBATCH -A
#SBATCH  --
#SBATCH --n
#SBATCH --n
#SBATCH  -
#SBATCH -o
#SBATCH -e
#DW jobdw t
#DW stage_i
#DW stage_o

export err_
export stud
export MPIC
export MPIC
export MPIC
export MPIC
export MPIC

cd $DW_JOB_
chmod +x bt
export folder=${SLURM_SUBMIT_DIR}/
export best_run_id=${SLURM_JOBID}_
export execu=btio
export exp_id=1
export run_id=1
cp del_tmp inputbt1.data
echo $DW_JOB_STRIPED >> inputbt1.
```

```
g"`
ize)); expbb_stripe_size=$((2*
//g"`
//g"`
```

```
let total_tasks=256
let nodes=8
export MPICH_MPIIO_HINTS="btio.nc"
let expbb_mpi_tasks=256
let expbb_omp_tasks=1
        if [ "$expbb_mpi_tasks" -gt "$nodes" ]; then
                let sockt=$expbb_mpi_tasks/$((2*$nodes))
        fi
        export start_io_aggr=1
        end_io_aggr=16
        for (( expbb_io_aggr=$start_io_aggr; expbb_io_aggr<=$end_io_aggr; expbb_io_aggr=2*$expbb_io_aggr ));
        do
                if [ "$expbb_io_aggr" -gt $start_io_aggr ]; then
                export temp_io_aggr=$(($expbb_io_aggr/2))
                export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HINTS" | sed "s/:cb_nodes=${temp_io_aggr}//

                fi
                export MPICH_MPIIO_HINTS="$MPICH_MPIIO_HINTS:cb_nodes=$expbb_io_aggr"
                let de
                let de
                for ((
                do
                if [ "
                export
                export

                fi
                if [ "
                export
                export

                fi
PED
```

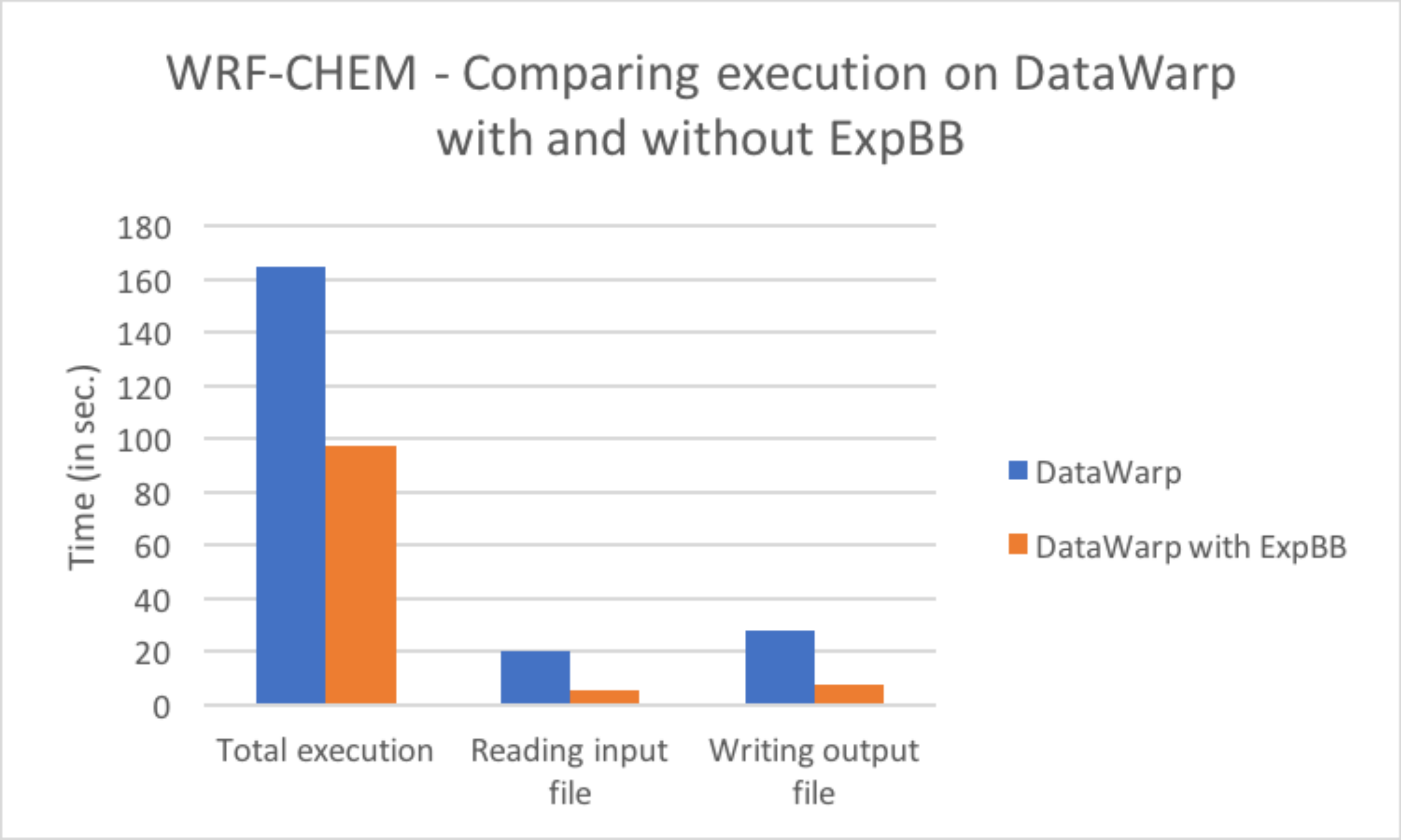
```
                          for (( expbb_stripe_buffer=$(($default_stripe_buffer/8)); expbb_stripe_buffer<=$((4*$de
fault_stripe_buffer)); expbb_stripe_buffer=2*$expbb_stripe_buffer ));
                          do
        if [ "$expbb_stripe_buffer" -eq "$(($default_stripe_buffer/8))" ] && [ $exp_id -gt 1 ]; then
export temp_stripe_buffer=$((4*$default_stripe_buffer))
export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HINTS" | sed "s/:striping_buffer=${temp_stripe_buffer}//g"`
        fi
        if [ "$expbb_stripe_buffer" -gt "$(($default_stri
export temp_stripe_buffer=$(($expbb_stripe_buffer/2))
export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HINTS" | sed
        fi
                          export MPICH_MPIIO_HINTS=
                export START=$(date +%s.%N)
                time srun --ntasks=$expbb_mpi_tasks --nod
pbb_omp_tasks} --threads-per-core=1 --hint=nomultithread
                END=$(date +%s.%N)
                DIFF=$(echo $END - $START | bc)
        chmod +x parse_darshan.sh
        ./parse_darshan.sh $SLURM_JOBID $run_id $folder $
let year=`date +%Y`
let month=`date +%-m`
let day=`date +%-d`
        export total_io=`cat $folder/$execu/$year/$month/
nt s}'`
        export percentage=`echo "scale=2; 100*$total_io/$
        echo "The percentage of the total execution time
        export line=`grep -n "| MPIIO" ${SLURM_SUBMIT_DIR
" | tail -n 1 | awk 'BEGIN{FS=":"} {print $1}'`
sed -n $line,$(($line+14))p ${SLURM_SUBMIT_DIR}/$err_file
                echo "BB_nodes=1 IO aggregators=$expbb_io
pe_buffer time $DIFF" >> ${SLURM_SUBMIT_DIR}/results_${SL
                if [ $run_id -gt 1 ]; then
export c▮m=`paste $folder/$execu/$year/$month/$day/${SLUR
d} | awk '{if(($4-$10)<0) print -1; else print 1;}'`
```

```
                        if [ $com -gt 0 ]; then
export best_run_id=${SLURM_JOBID}_$(($run_id-1))
echo $best_run_id > ${SLURM_SUBMIT_DIR}/best_run_id.txt
                        break
                        else
export best_run_id=${SLURM_JOBID}_$run_id
export check_all=`paste $folder/$execu/$year/$month/$day/${SLURM_JOBID}_$run_id $folder/$execu/$year/$month/$day/${best
_run_id} | awk '{if((($6-$12)/$12)>0.05) print 1; else print 0;}'`
                        if [ $check_all -gt 0 ]; then
                        cd
                        export myself=`scontrol show job ${SLURM_JOBID} | grep Command | awk 'BEGIN{FS="/"} {print $NF}

                        cp $myself hybrid_$myself
                        sed s/cd $DW_JOB_STRIPED// -i hybrid_
echo $best_run_id > ${SLURM_SUBMIT_DIR}/hybrid_run_id.txt
                        else
echo $best_run_id > ${SLURM_SUBMIT_DIR}/best_run_id.txt
                        fi
                fi
                fi
                let run_id=${run_id}+1
        if [ "$exp_id" -gt "1" ]; then
export MPICH_MPIIO_HINTS=`echo "$MPICH_MPIIO_HINTS" | sed "s/:striping_buffer=${expbb_stripe_buffer}//g"`
        fi
                grep "| MPIIO" ${SLURM_SUBMIT_DIR}/$err_file | grep patt | grep -v darshan | awk '{print $NF}' | uniq >
> ${SLURM_SUBMIT_DIR}/pio_files_${SLURM_JOBID}.txt
        let exp_id=$exp_id+1
                              done;
                        done;
done


exit ▮
```

Rules

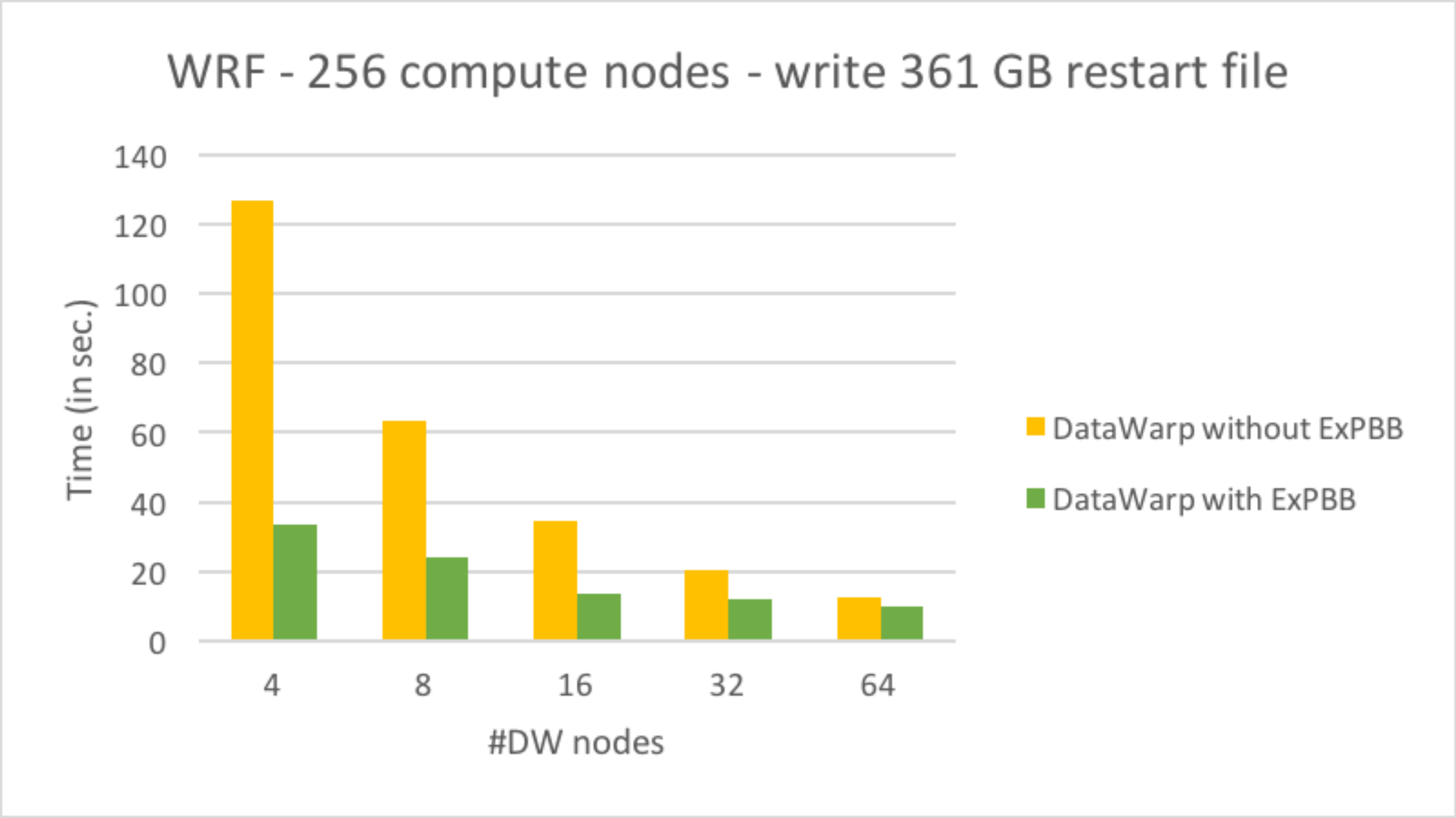
- If the performance becomes worse while we decrease the striping unit and the number of system write/reads is significant large, then increase the MPI I/O aggregators. If the I/O is slower again, then restart with the used number of MPI I/O aggregators but initial parameters values.
- When the exploration of specific number BB nodes finish, submit another job with double BB nodes and compare with the previous best performance result

Results I



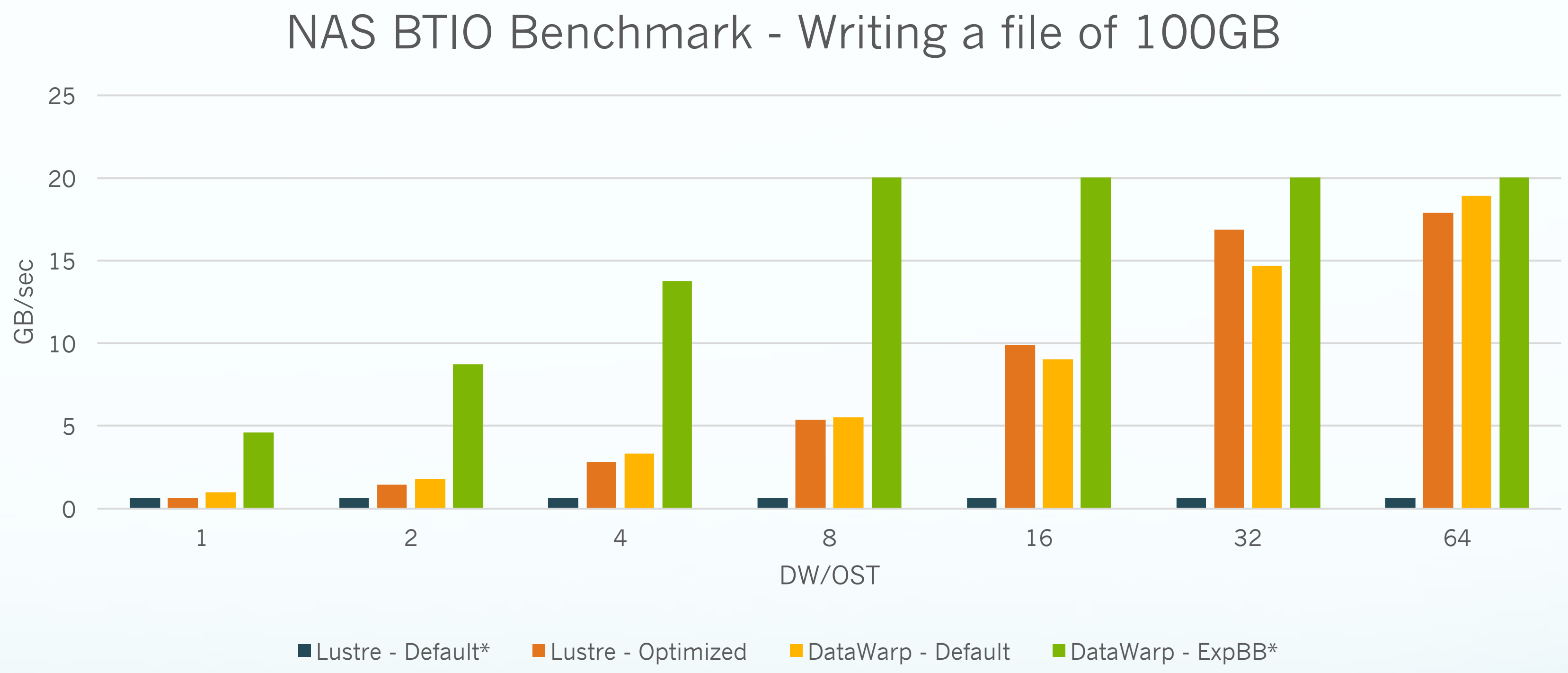
The total execution time is improved 1,7 times with ExPBB and the I/O is improved up to 3,8 times for 1 BB node. Finally, the total execution time is 13.4% faster than Lustre with 64 OSTs.

Results II



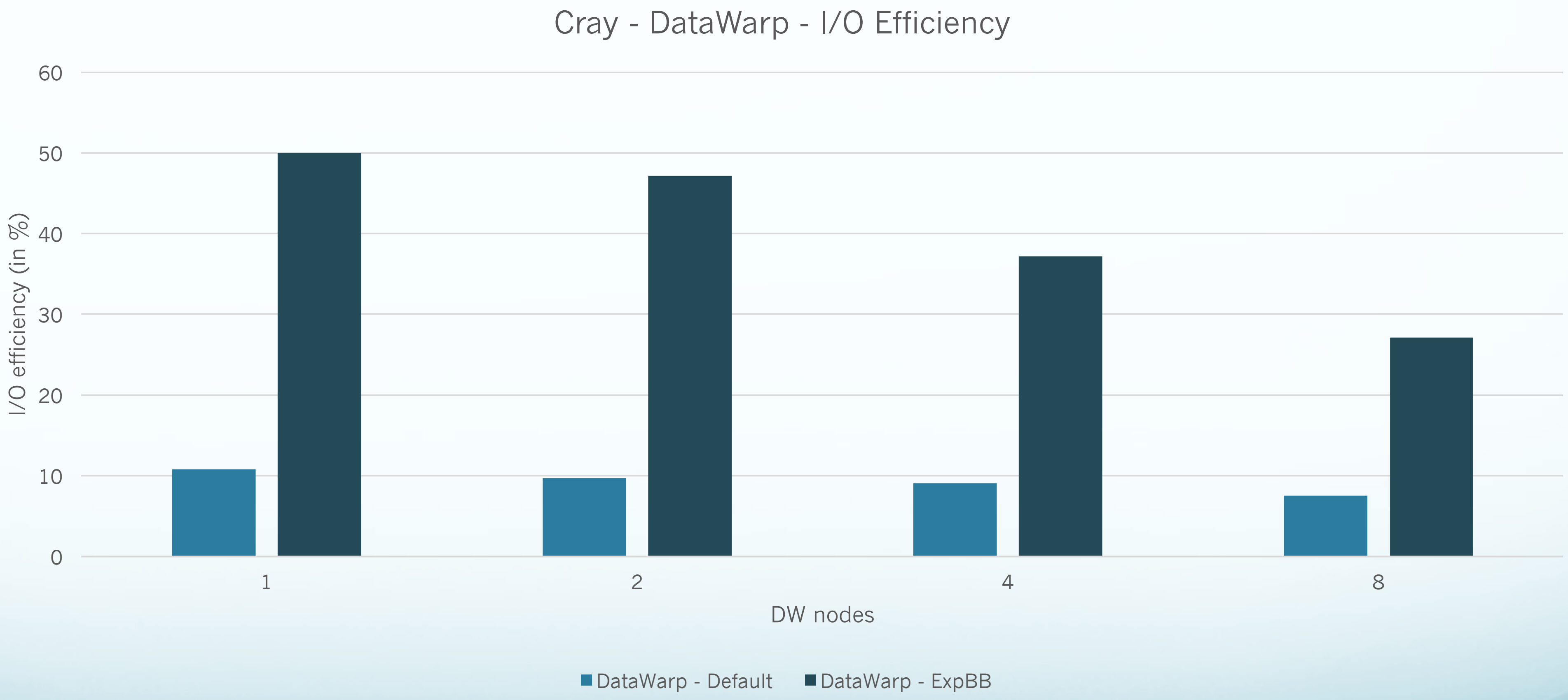
The I/O was improved with ExPBB between 1,28 till 3,8 times.
The execution on 16 BB nodes with ExPBB is faster than 64 BB nodes without ExpBB
MPICH_MPIIO_HINTS="wrfi*:cb_nodes=128:striping_unit=4194304, wrfo*:cb_nodes=256:striping_unit=4194304, wrfr*:cb_nodes=256:striping_unit=4194304"

Results III



We observe that for 8 BB nodes, with ExPBB framework, we have better performance than every other configuration. The maximum speedup compared to default BB execution, is 4,84. Moreover, 8 BB nodes have better performance than 64 OSTs. The ROI in this case is significant higher with Cray DataWarp.

I/O - Efficiency



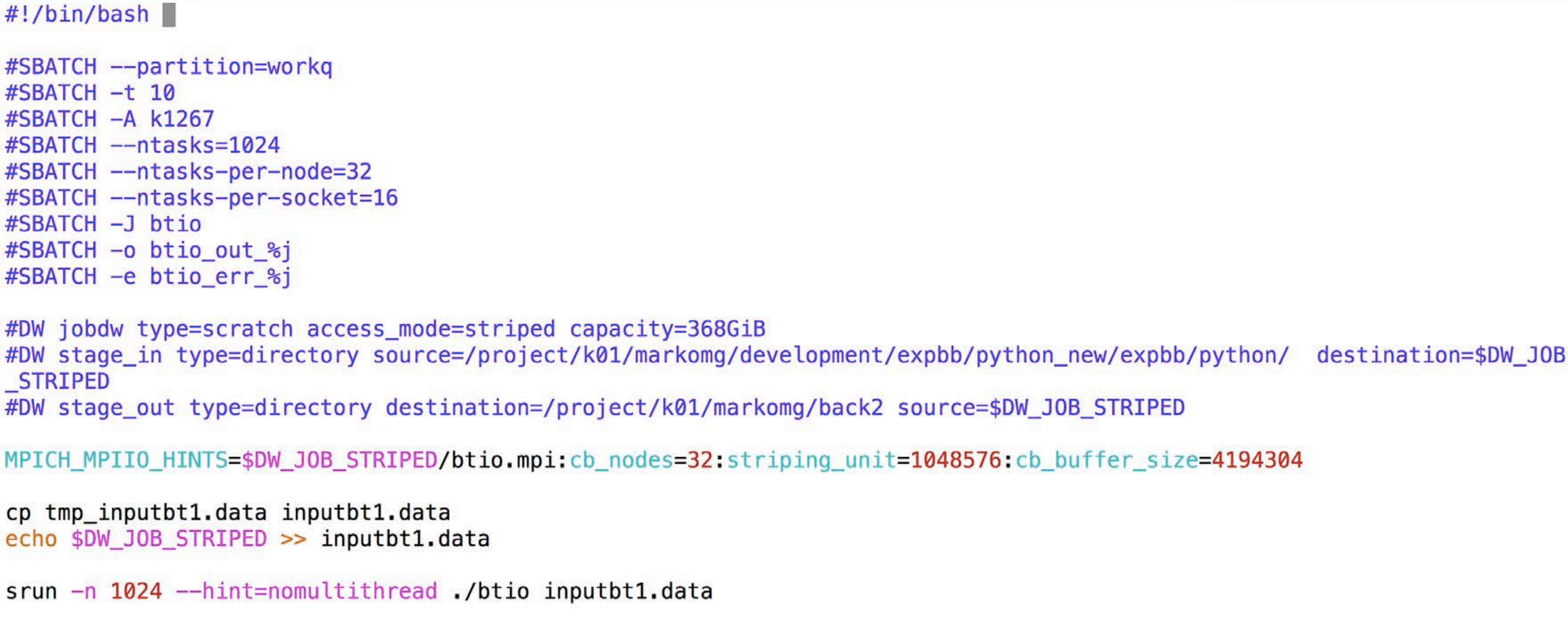
ExpBB – Output I

```
./expbb btio.sh

Preparing and executing default script on Lustre

I/O duration for the file btio.mpi on Lustre with 1 OSTs is 155.26 seconds

I/O duration for the file btio.mpi on Lustre with 2 OSTs is 69.87 seconds

I/O duration for the file btio.mpi on Lustre with 4 OSTs is 35.27 seconds

I/O duration for the file btio.mpi on Lustre with 8 OSTs is 18.57 seconds

I/O duration for the file btio.mpi on Lustre with 16 OSTs is 10.12 seconds

I/O duration for the file btio.mpi on Lustre with 32 OSTs is 5.93 seconds

I/O duration for the file btio.mpi on Lustre with 64 OSTs is 5.59 seconds

I/O duration for the file btio.mpi on Lustre with 128 OSTs is 6.10 seconds
```

ExpBB – Output II

```
Preparing and executing default script on Burst Buffer
The I/O duration for the file btio.mpi on Burst Buffer with default parameters is 96.62 seconds
Starting auto-tuning execution on 1 Burst Buffer nodes
I/O duration for the file btio.mpi on 1 Burst Buffer with optimized parameters is 23.617 seconds
The new submission file with optimized parameters is named expbb_1_btio.sh
MPICH_MPIIO_HINTS=$DW_JOB_STRIPED/btio.mpi:cb_nodes=32:striping_unit=1048576:cb_buffer_size=4194304
…
Starting auto-tuning execution on 8 Burst Buffer nodes
I/O duration for the file btio.mpi on 8 Burst Buffer with optimized parameters is 4.99633 seconds
The new submission file with optimized parameters is named expbb_8_btio.sh
MPICH_MPIIO_HINTS=$DW_JOB_STRIPED/btio.mpi:cb_nodes=128:striping_unit=2097152:cb_buffer_size=8388608
```

New submission script for 1 BB node



```
#!/bin/bash

#SBATCH --partition=workq
#SBATCH -t 10
#SBATCH -A k1267
#SBATCH --ntasks=1024
#SBATCH --ntasks-per-node=32
#SBATCH --ntasks-per-socket=16
#SBATCH -J btio
#SBATCH -o btio_out_%j
#SBATCH -e btio_err_%j

#DW jobdw type=scratch access_mode=striped capacity=368GiB
#DW stage_in type=directory source=/project/k01/markomg/development/expbb/python_new/expbb/python/  destination=$DW_JOB
_STRIPED
#DW stage_out type=directory destination=/project/k01/markomg/back2 source=$DW_JOB_STRIPED

MPICH_MPIIO_HINTS=$DW_JOB_STRIPED/btio.mpi:cb_nodes=32:striping_unit=1048576:cb_buffer_size=4194304

cp tmp_inputbt1.data inputbt1.data
echo $DW_JOB_STRIPED >> inputbt1.data

srun -n 1024 --hint=nomultithread ./btio inputbt1.data
```

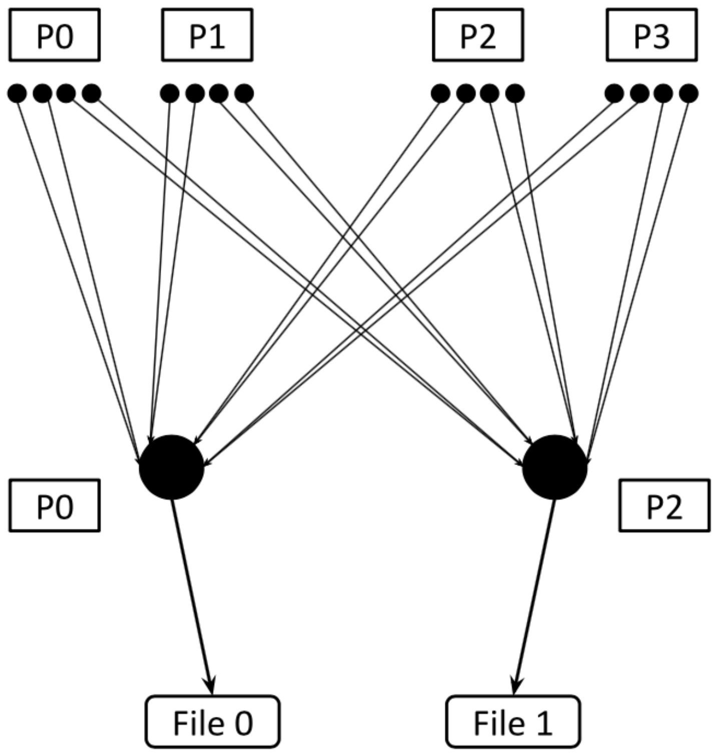
Study-case PIDX

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

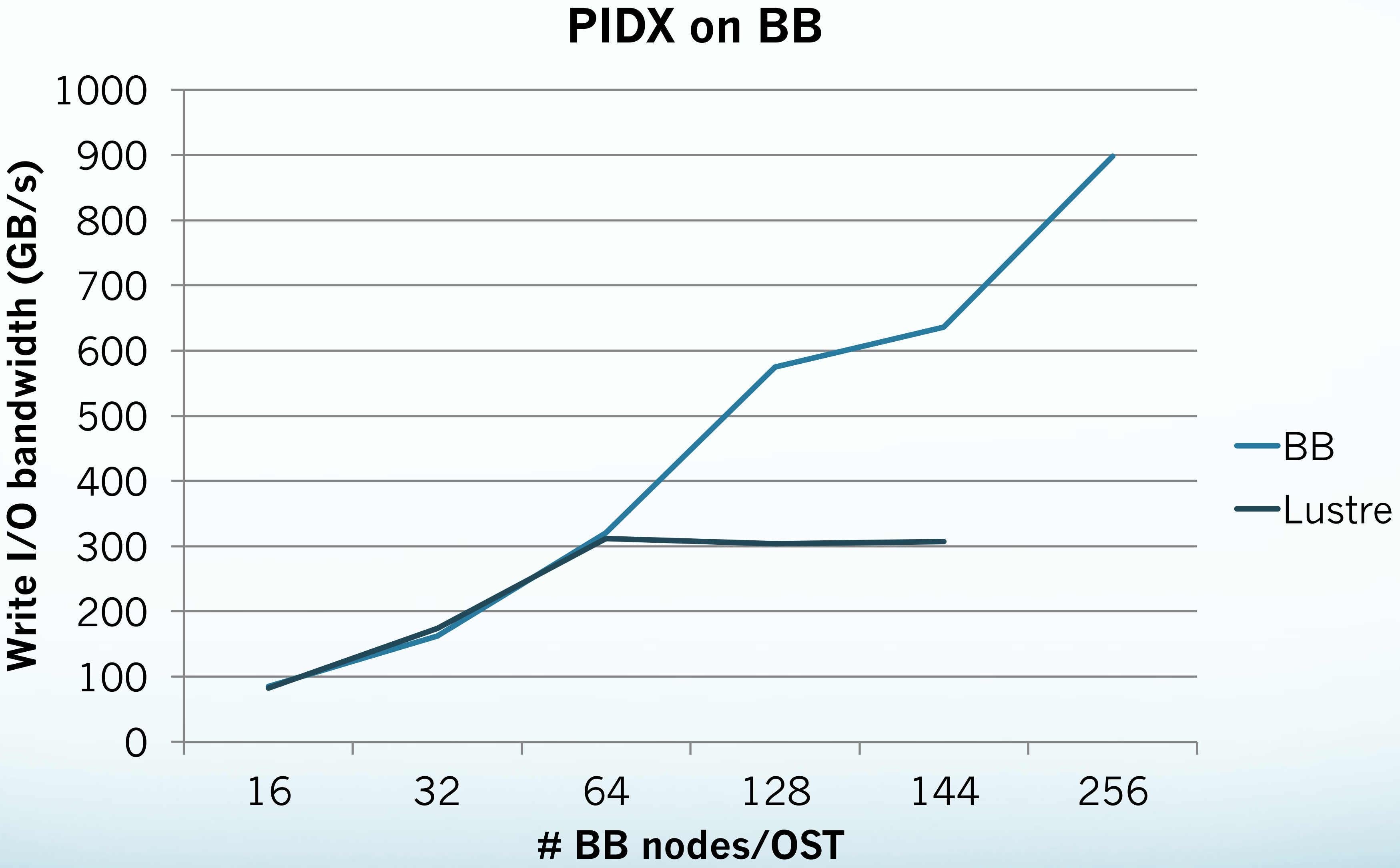


Synchronous reducing mode of communication using one-sided MPI

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

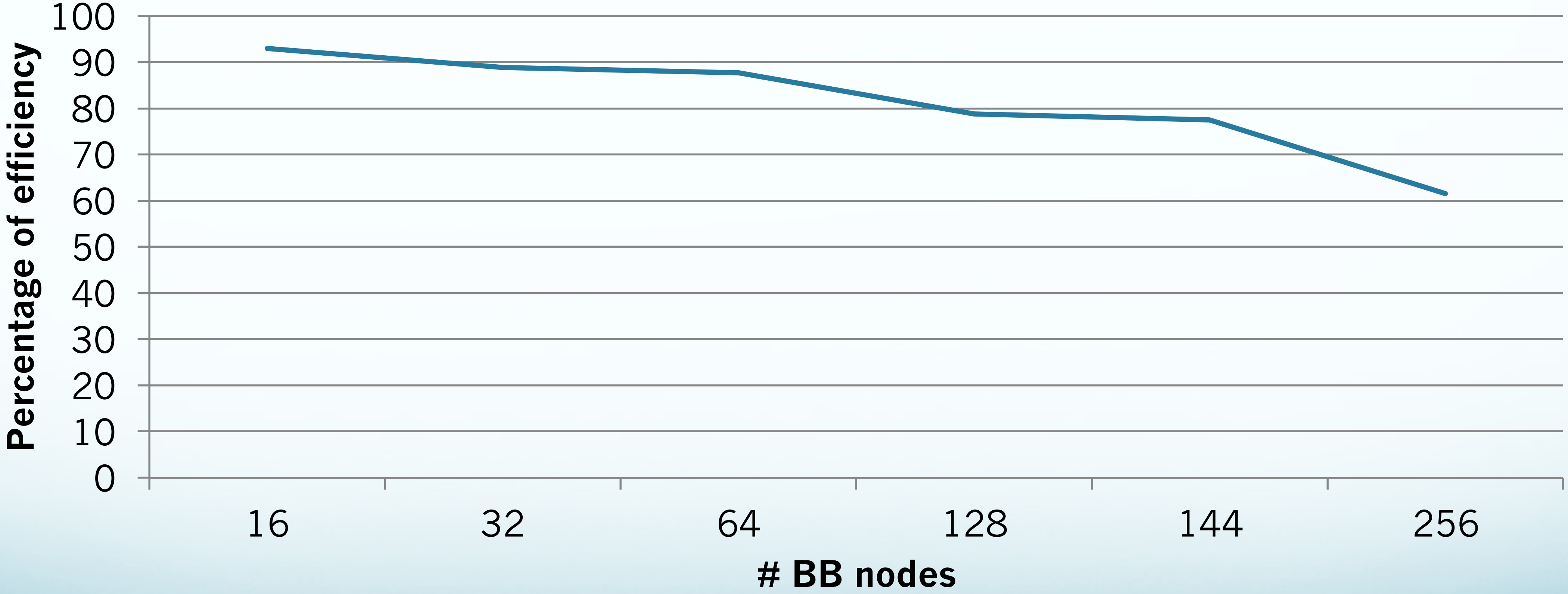
File locking contention is reduced by having an aggregator to write to only one file

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



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



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

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

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/gmarkomanolis/bb_ixpug18
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
```

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_PERSISTENT_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

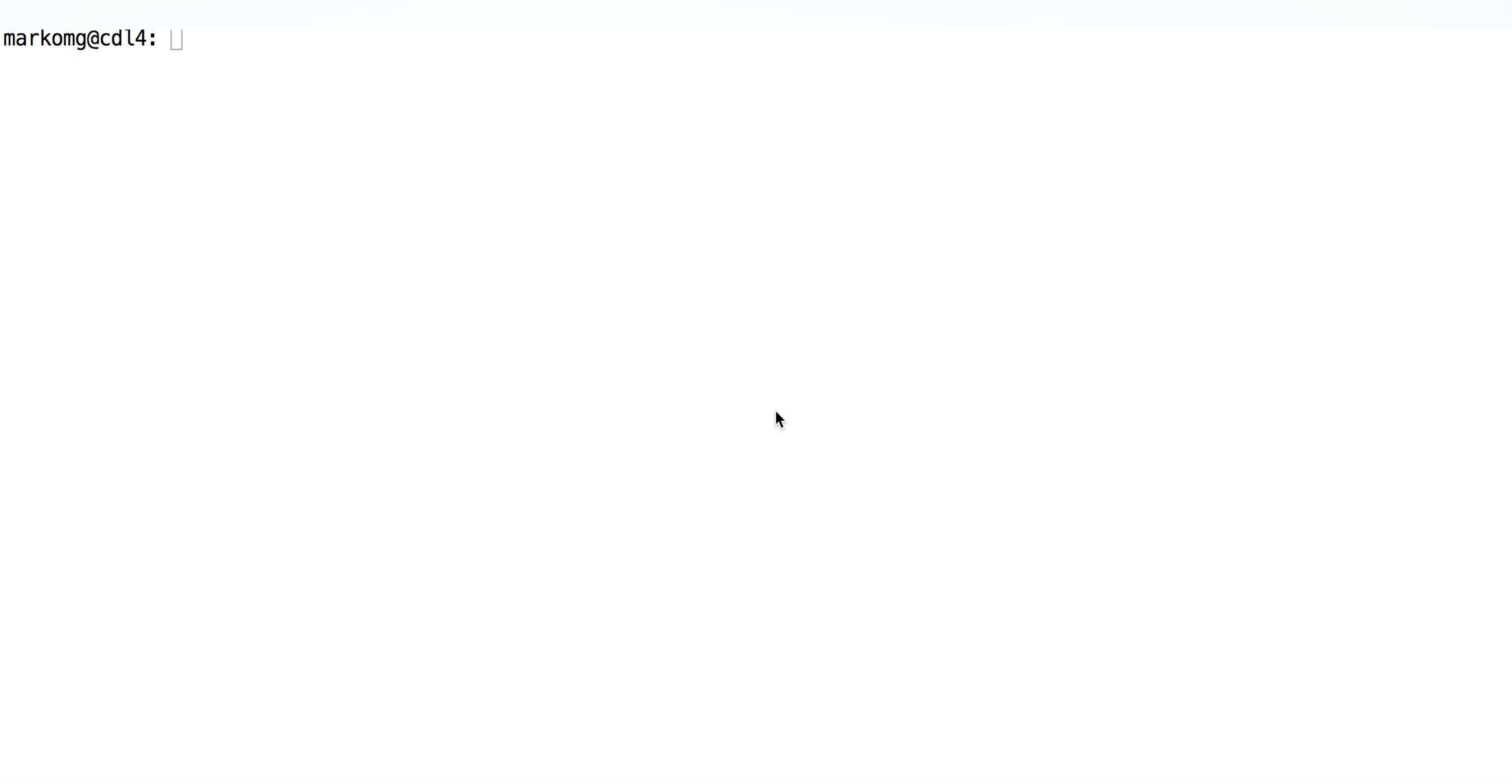
- File: **wait.sh**
- Execute: ./wait.sh number_of_images /path_to_Lustre_stage_out_folder/

```
#!/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
```



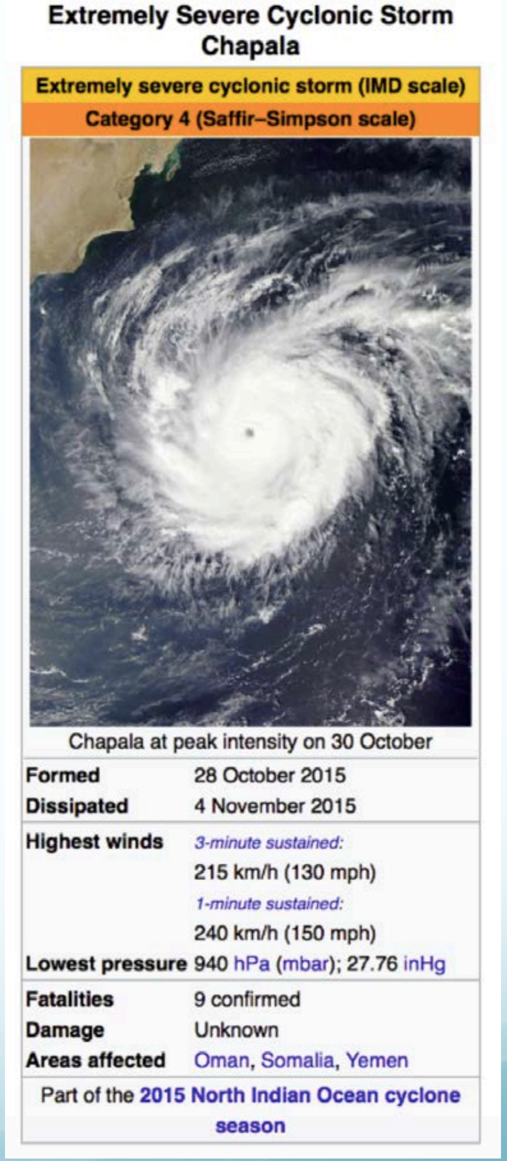
```
markomg@cdl4: 
```

Case 2: In situ processing and visualization

(collaboration with KVL)

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**

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

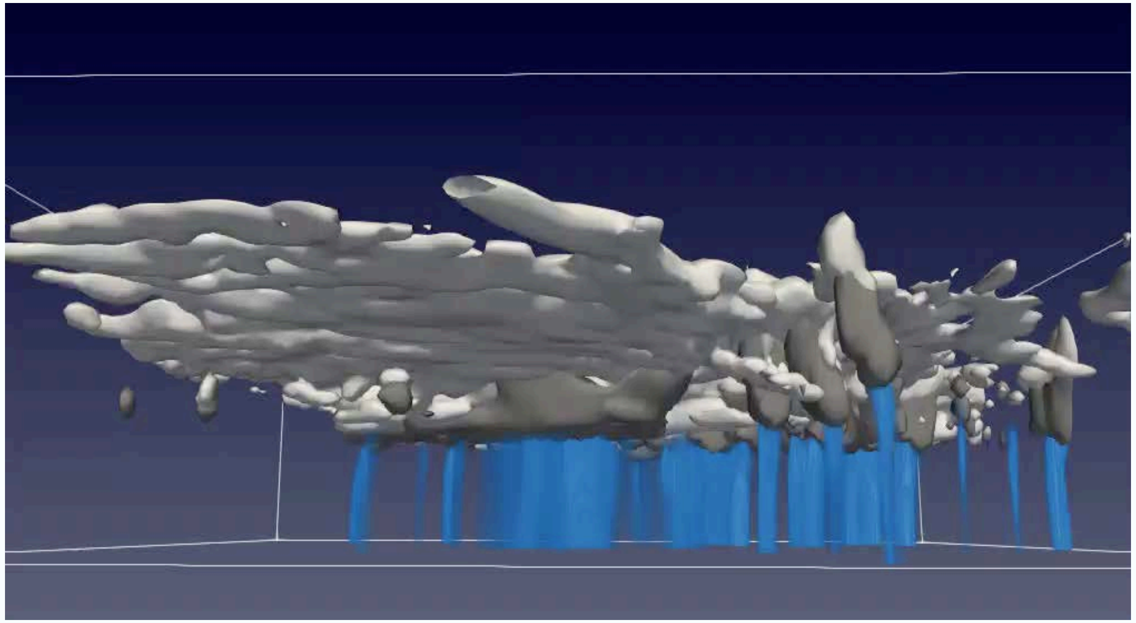
Results

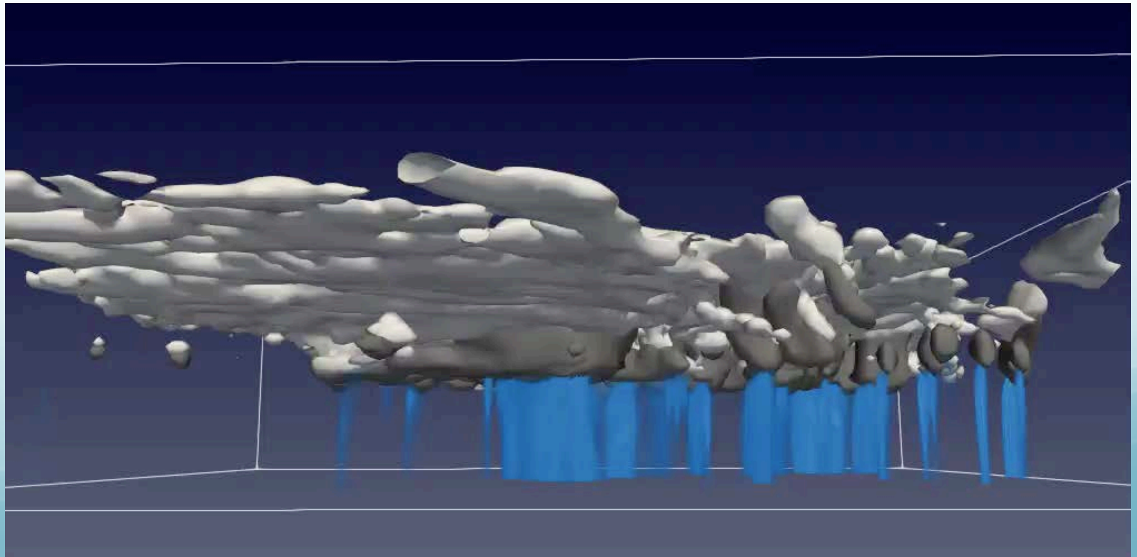
- We use two domains, one small (1100x1000x34) and one larger (3500x3000x34).
- In order to increase the details in the available data, we are testing two cases, saving data every one hour and every 10 minutes.
- A post-processing tool chops from the whole area only the cyclone region and saves this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

Videos here:
https://github.com/gmarkomanolis/bb_ixpug18
folder: complex_workflow/cyclone_vis

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.





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
- 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
- Think clever and innovative on how to implement your workflow

Thank you!
Questions?

georgios.markomanolis@kaust.edu.sa
saber.feki@kaust.edu.sa