Accelerate your IO with the NERSC Burst Buffer
HPC memory hierarchy

**Past**
- On Chip
  - CPU
    - Memory (DRAM)
      - Storage (HDD)
  - Off Chip

**Future**
- On Chip
  - CPU
    - Near Memory (HBM)
    - Far Memory (DRAM)
    - Near Storage (SSD)
- Off Chip
  - Far Storage (HDD)
HPC memory hierarchy

- Silicon and system integration
- Bring everything – storage, memory, interconnect – closer to the cores
- Raise center of gravity of memory pyramid, and make it fatter
  - Enable faster and more efficient data movement
  - Scientific Big Data: Addressing Volume, Velocity
Why an SSD Burst Buffer?

• **Motivation**: Handle spikes in I/O bandwidth requirements
  – Reduce overall application run time
  – Compute resources are idle during I/O bursts
Why an SSD Burst Buffer?

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

• **Some user applications have challenging I/O patterns**
  – High IOPs, random reads, different concurrency… perfect for SSDs

• **Cost rationale**: Disk-based PFS bandwidth is expensive
  – Disk capacity is relatively cheap
  – SSD *bandwidth* is relatively cheap
  
    => Separate bandwidth and spinning disk
    • Provide high BW without wasting PFS capacity
    • Leverage Cray Aries network speed
• DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or ‘persistent’).
• Users see a POSIX filesystem
• Filesystem can be striped across multiple BB nodes (depending on allocation size requested)
• DataWarp software (integrated with SLURM WLM) allocates portions of available storage to users per-job (or ‘persistent’).
• Users see a POSIX filesystem
• Filesystem can be striped across multiple BB nodes (depending on allocation size requested)
Burst Buffer Blade = 2xNodes

3.2 TB Intel P3608 SSD
PCle Gen3 8x
Xeon E5 v1

3.2 TB Intel P3608 SSD
PCle Gen3 8x
Xeon E5 v1

3.2 TB Intel P3608 SSD
PCle Gen3 8x

3.2 TB Intel P3608 SSD
PCle Gen3 8x

To HSN

Aries
Burst Buffer Blade = 2xNodes

- 3.2 TB Intel P3608 SSD
- PCIe Gen3 8x
- Xeon E5 v1

- ~1.8PiB of SSDs over 288 nodes
- Accessible from both HSW and KNL nodes
DataWarp: Under the hood

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

http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/
Two kinds of DataWarp Instances

• “Instance”: an allocation on the BB
• Can it be shared? What is its lifetime?
  – Per-Job Instance
    • Can only be used by job that creates it
    • Lifetime is the same as the creating job
    • Use cases: PFS staging, application scratch, checkpoints
  – Persistent Instance
    • Can be used by any job (subject to UNIX file permissions)
    • Lifetime is controlled by creator
    • Use cases: Shared data, PFS staging, Coupled job workflow
    • NOT for long-term storage of data!
Two DataWarp Access Modes

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

• Private
  – Files are assigned to one DataWarp node
  – File are visible to only the compute node that created them
  – Each DataWarp node is an MDS for one or more compute nodes
How to use DataWarp

**Principal user access: SLURM Job script directives: #DW**

- Allocate job or persistent DataWarp space
- Stage files or directories in from PFS to DW; out DW to PFS
- Access BB mount point via $DW_JOB_STRIPED, $DW_JOB_PRIVATE, $DW_PERSISTENT_STRIPED_name

**User library API – libdatawarp**

- Allows direct control of staging files asynchronously
- C library interface
  - [https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-8](https://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-8)
  - [https://github.com/NERSC/BB-unit-tests/tree/master/datawarpAPI](https://github.com/NERSC/BB-unit-tests/tree/master/datawarpAPI)
Integration with SLURM

• ‘type=scratch’ – duration just for compute job (i.e. not ‘persistent’)

• ‘access_mode=striped’ – visible to all compute nodes (i.e. not ‘private’) and striped across multiple BB nodes
  – Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 5 BB nodes)

• Data ‘stage_in’ before job start and ‘stage_out’ after

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

• ‘type=scratch’ – duration just for compute job (i.e. not ‘persistent’)
• ‘access_mode=striped’ – visible to all compute nodes (i.e. not ‘private’) and striped across multiple BB nodes
  — Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 5 BB nodes)
• Data ‘stage_in’ before job start and ‘stage_out’ after

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

- `type=scratch` – duration just for compute job (i.e. not ‘persistent’)
- `access_mode=striped` – visible to all compute nodes (i.e. not ‘private’) and striped across multiple BB nodes
  - Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 5 BB nodes)
- Data ‘stage_in’ before job start and ‘stage_out’ after

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

- ‘type=scratch’ – duration just for compute job (i.e. not ‘persistent’)
- ‘access_mode=striped’ – visible to all compute nodes (i.e. not ‘private’) and striped across multiple BB nodes
  — Actual distribution across BB Nodes is in units of (configurable) granularity (currently 200 GB at NERSC in wlm_pool, so 1000 GB would normally be placed on 5 BB nodes)
- Data ‘stage_in’ before job start and ‘stage_out’ after

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

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

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

Delete

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

Use in another job

```bash
#BB destroy_persistent name=myBBname
```

```bash
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```
Integration with SLURM

• Using a *persistent* DataWarp instance
  – Lifetime different from the batch job
  – Usable by any batch job
  – `name=xyz:` Name of persistent instance to use

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

Delete

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

Use in another job

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

• Using a *persistent* DataWarp instance
  – Lifetime different from the batch job
  – Usable by any batch job
  – `name=xyz`: Name of persistent instance to use

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

Delete

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

Use in another job

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

```bash
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
sicrun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```
Integration with SLURM

• Using a persistent DataWarp instance
  – Lifetime different from the batch job
  – Usable by any batch job
  – name=xyz: Name of persistent instance to use

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

Delete

Use in another job

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

```bash
mkdir $DW_PERSISTENT_STRIPED_myBBname/test1
srun a.out INSERT_YOUR_CODE_OPTIONS_HERE
```
Resources

• NERSC Burst Buffer Web Pages
  http://www.nersc.gov/users/computational-systems/cori/burst-buffer/

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

• Burst Buffer Early User Program Paper
  http://www.nersc.gov/assets/Uploads/Nersc-BB-EUP-CUG.pdf
Burst Buffer is doing very well against benchmark performance targets

- Out-performs Lustre significantly
- Fastest IO system in the world!

<table>
<thead>
<tr>
<th></th>
<th>IOR Posix FPP</th>
<th>IOR MPIO Shared File</th>
<th>IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>1.7 TB/s</td>
<td>1.3 TB/s</td>
<td>28M</td>
</tr>
<tr>
<td>Write</td>
<td>1.6 TB/s</td>
<td>1.4 TB/s</td>
<td>13M</td>
</tr>
</tbody>
</table>

*Best Measured (287 Burst Buffer Nodes : 11120 Compute Nodes; 4 ranks/node)*

*Bandwidth tests: 8 GB block-size 1MB transfers  IOPS tests: 1M blocks 4k transfer*
Striping, granularity and pools

- DataWarp nodes are configured to have “granularity”
  - Minimum amount of data that will land on one node

- Two “pools” of DataWarp nodes, with different granularity
  - *wlm_pool* (default): 200GiB
    - #DW jobdw capacity=1000GB access_mode=striped type=scratch pool=wlm_pool
  - *sm_pool*: 20.14 GiB
    - #DW jobdw capacity=1000GB access_mode=striped type=scratch pool=sm_pool

- For example, 1.2TiB will be striped over 6 BB nodes in *wlm_pool*, but over 60 BB nodes in *sm_pool*
  - No guarantee that allocation will be spread evenly over BB nodes; may see >1 “grain” on a single node
Performance tips

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

• NERSC has the first Burst Buffer for open science in the USA
  – And the first in the world that is being tested for real use cases!
• Users are able to take advantage of SSD performance
  – Some tuning may be required to maximise performance
• Many bugs now worked through
  – But care is needed when using this new technology!
• User experience today is generally good
  – Let us know if you’re interested in using the Burst Buffer…
More information...

https://www.nersc.gov/users/computational-systems/cori/burst-buffer/
Extra slides
SSD write protection

• SSDs support a set amount of write activity before they wear out

• Runaway application processes may write an excessive amount of data, and therefore, “destroy” the SSDs

• Three write protection policies
  – Maximum number of bytes written in a period of time
  – Maximum size of a file in a namespace
  – Maximum number of files allowed to be created in a namespace

• Log, error, log and error
  -- EROFS  (write window exceeded)
  -- EMFILE  (maximum files created exceeded)
  (maximum file size exceeded)