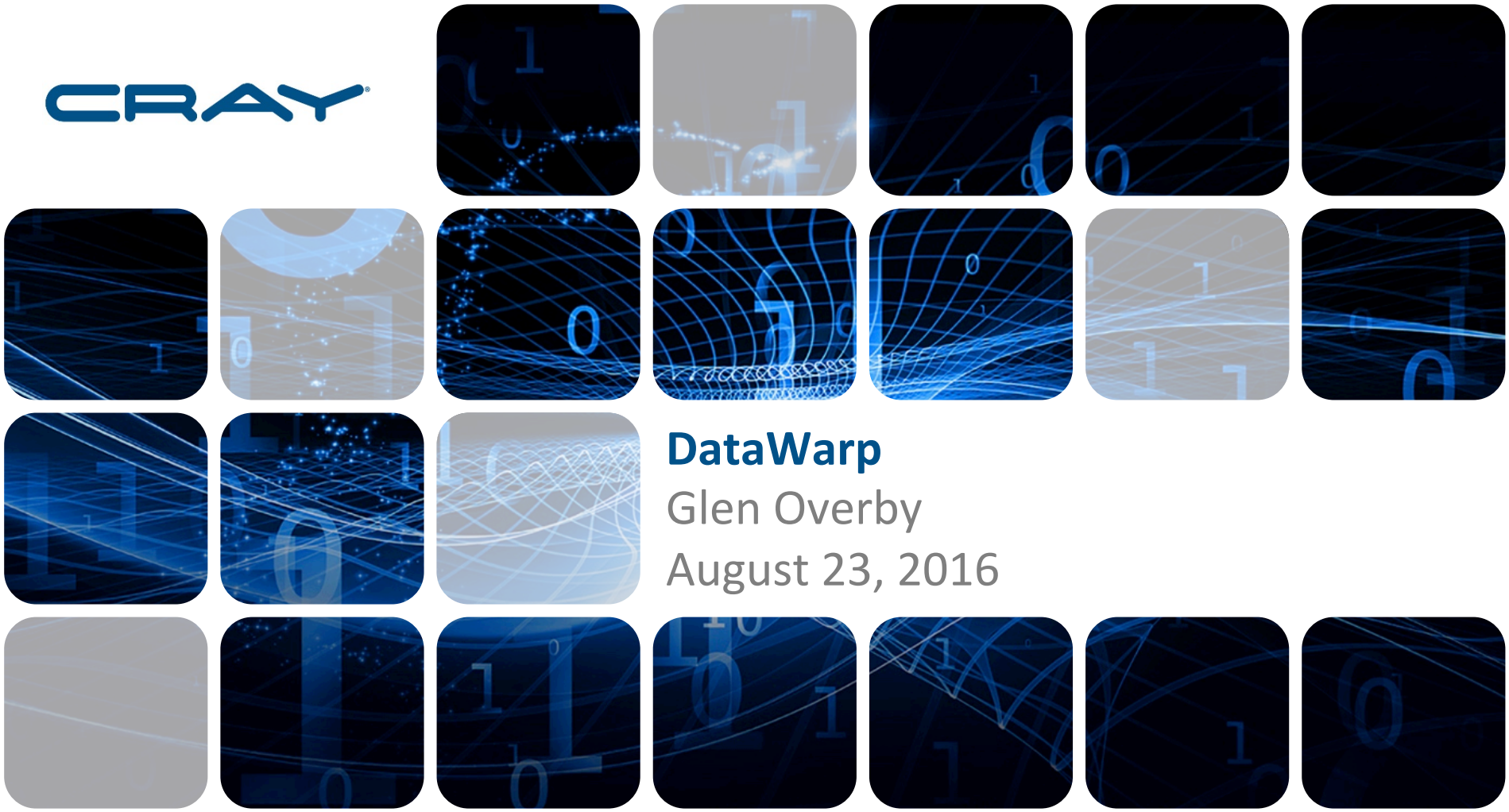
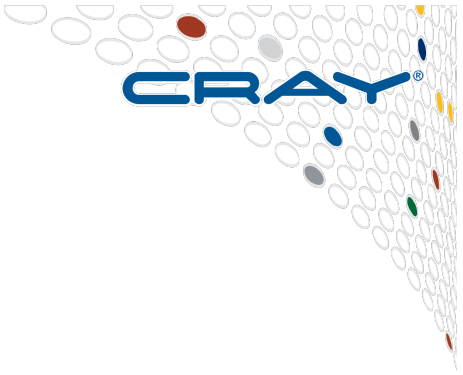


DataWarp

Glen Overby

August 23, 2016

Topics



- Introduction
- Overview
- Concepts
- DataWarp Usage in WLMs
- Using dwstat
- Future Enhancements

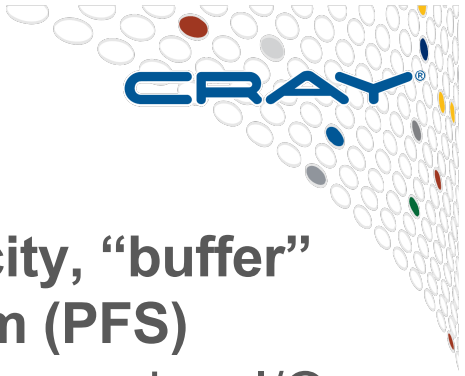
Overview - What is DataWarp?

- **DataWarp is Cray's implementation of the Burst Buffer concept, plus more**
- **Has both Hardware & Software components**
  - **Hardware**
    - XC40 Service node, directly connected to Aries network
    - PCIe SSD Cards installed on the node
  - **Software**
    - DataWarp service daemons
    - DataWarp Filesystem (using DVS, LVM, XFS)
    - Integration with WorkLoad Managers (Slurm, M/T, PBSpro)

Observations & Trends

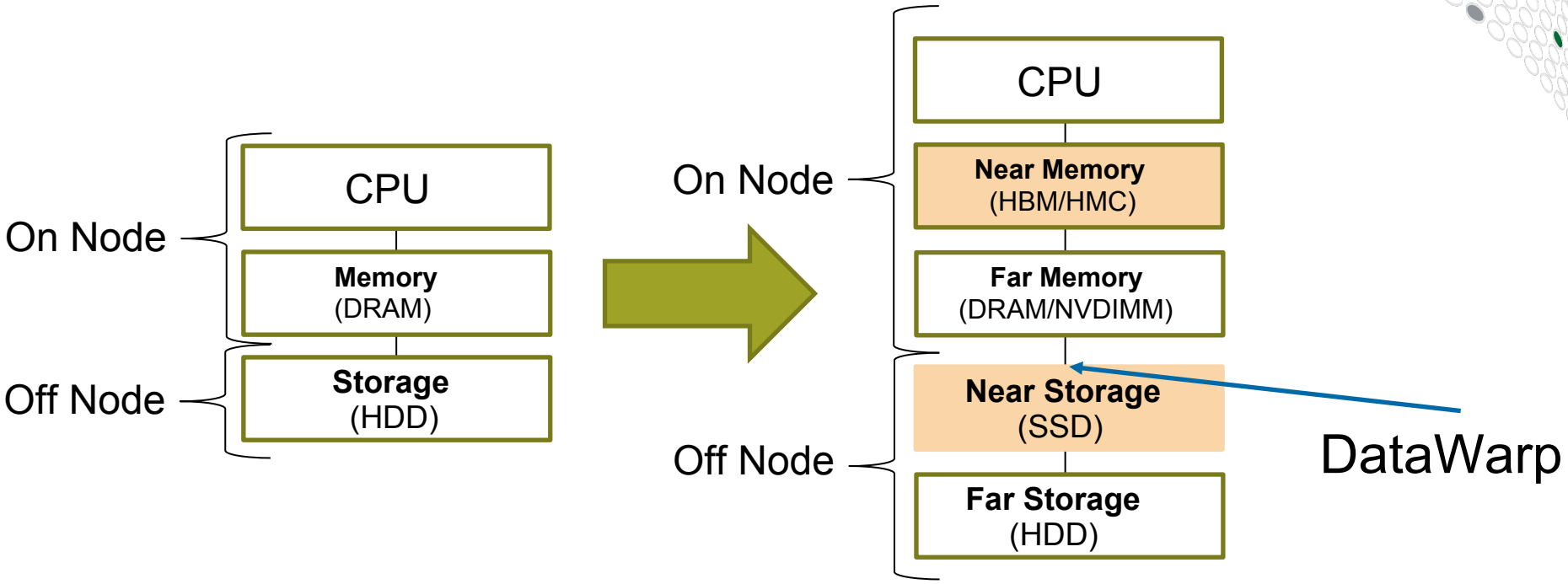
- **Many programs do I/O in bursts**
  - Read, Compute, Write, Compute, Write, Compute, etc.
- **Want to have high bandwidth when doing I/O**
  - Compute resources largely idle during I/O
- **Disk-based PFS bandwidth is expensive, capacity is cheap**
  - PFS do provide lots of capacity, reliability, permanence, etc.
- **SSD bandwidth is (relatively) inexpensive**
- **Large I/O load at beginning and end of job**
- **Cray Aries network is faster, lower latency, shorter distance than path to PFS**

Burst Buffer Concepts



- **Burst Buffer (BB) - A high bandwidth, lower capacity, "buffer" space, backed by a disk based Parallel File System (PFS)**
  - Increased BB bandwidth decreases time programs spend on I/O
- **BB can interact with PFS before, during, and after program use**
  - Stage data in to BB before computes allocated
  - Stage data back out to PFS after computes deallocated
  - Stage data in or out, using BB hardware, while program in computational phase

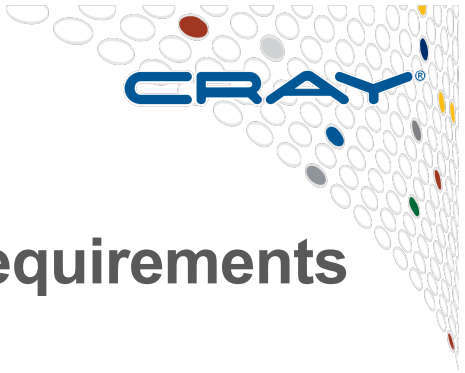
Exascale Computing Memory Trends



Past

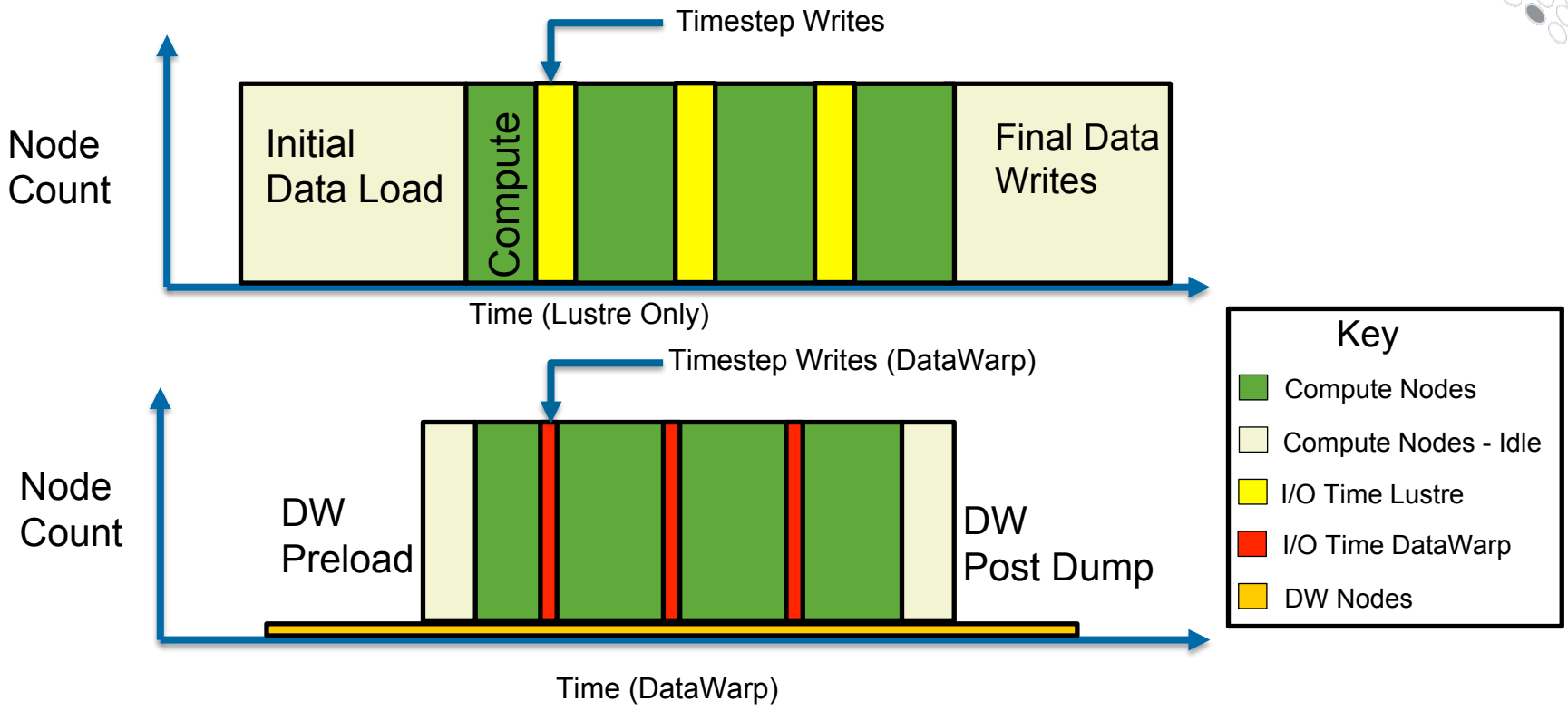
~~*Future*~~ *Now*

Cray DataWarp Motivation

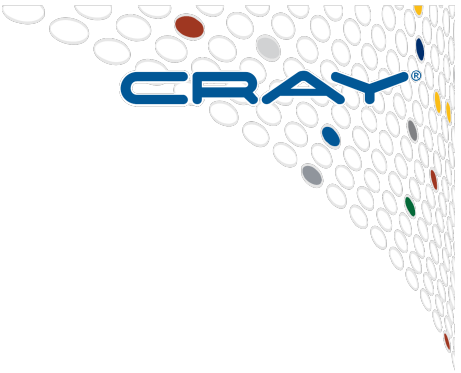


- **Handle peaks and spikes in I/O bandwidth requirements**
  - ...without increasing the size of the PFS
- **Reduce job wall clock time**
- **Provision capacity on rotating media separately from provisioning bandwidth**
  - Prevents excess capacity or insufficient bandwidth
  - Reduces cost

DataWarp - Minimize Compute Residence Time

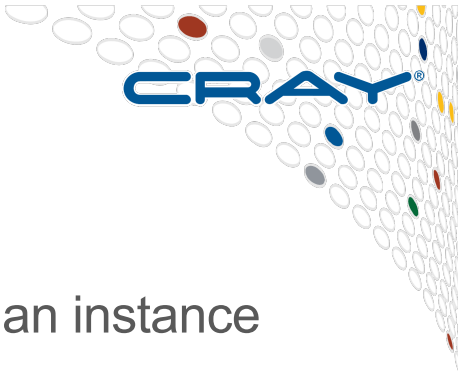


Two kinds of DataWarp Instances (Space)



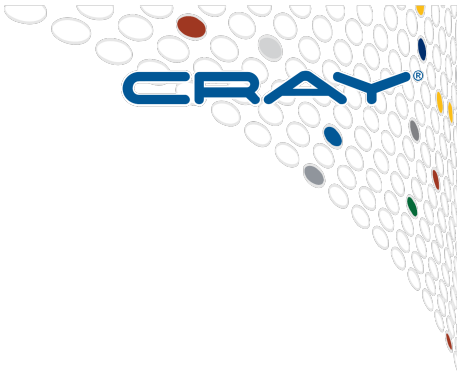
- **Can it be shared? What is its lifetime?**
- **Job Instance**
  - Can only be used by job that creates it
  - Use Lifetime is the same as the creating job
  - cases: PFS staging, application scratch, checkpoints
- **Persistent Instance**
  - Can be used by any job (subject to permissions)
  - Lifetime is controlled by creator
  - Use cases: Shared data, PFS staging, Job workflow, shared libraries, Python libraries, …..

DataWarp Access Modes



- **Striped ("Shared")**
  - Files are striped across all DataWarp nodes assigned to an instance
  - Files are visible to all compute nodes using the instance
  - Aggregates both capacity and bandwidth per file
  - One DataWarp node elected as the MDS
- **Private**
  - Files are striped across all DataWarp nodes assigned to an instance
  - File are visible to only the compute node that created them
  - Each DataWarp node is an MDS for one or more computes
  - Think of it like a big fast /tmp
- **Swap**
  - Per-compute node swap space

How to Utilize DataWarp



- **Job script directives - #DW ...**
  - Allocate job DataWarp space
  - Access persistent DataWarp space
  - Stage files or directories in from PFS to DW; out from DW to PFS
  - Supported by Slurm, Moab/TORQUE, and PBSPro
- **User library API - libdatawarp**
  - Allows direct control of staging files asynchronously
  - C library interface
- **Mount points**
  - Perform POSIX I/O through mount points on compute nodes
- **Command line**
  - “dwstat” command for users and admins to see status of their spaces
  - Other commands, like dwcli, mostly for system admins

Job Script commands: Without & With DataWarp

```
#!/bin/bash
#SBATCH -n 3200 -t 2000

export JOBDIR=/lustre/my_dir

srun –n 3200 a.out
```

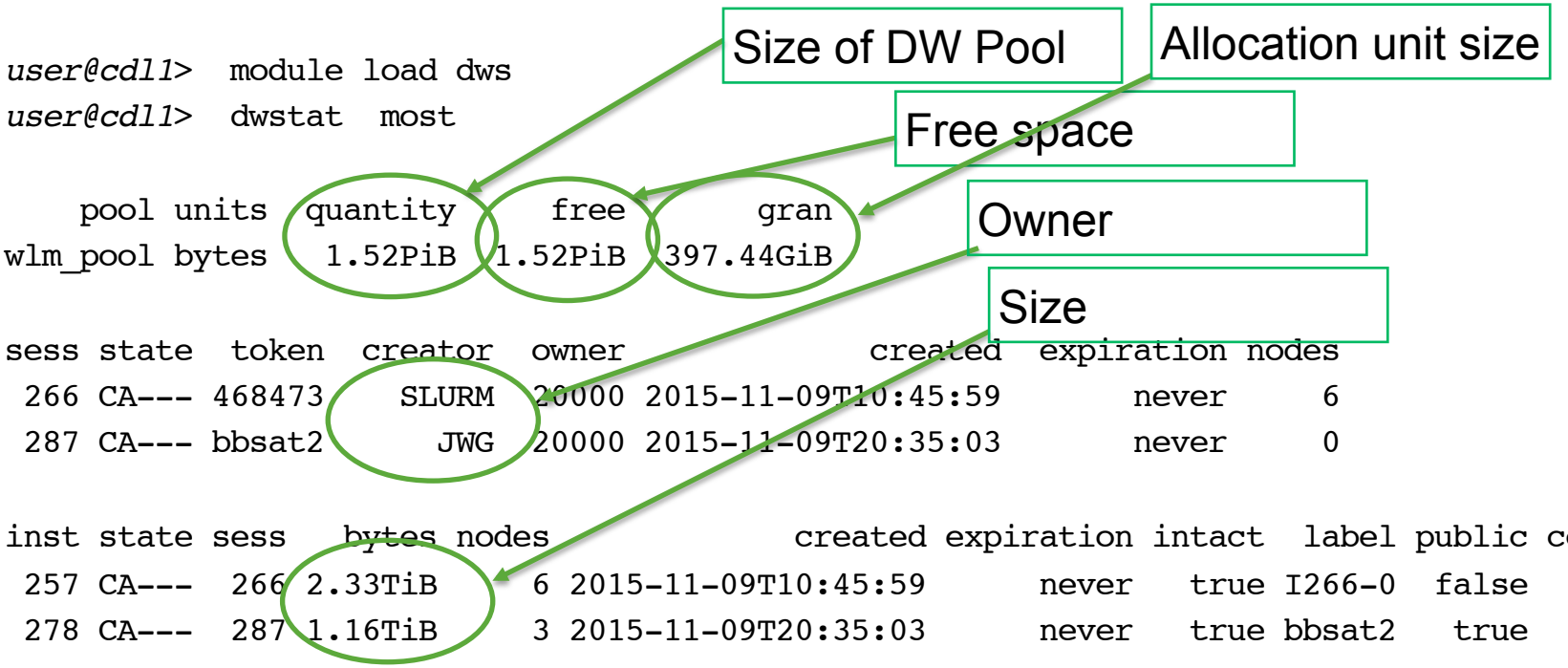
```
#!/bin/bash
#SBATCH -n 3200 -t 2000

#DW jobdw type=scratch access_mode=striped  capacity=1TiB
#DW stage_in type=directory source=/lustre/my_dir destination=$DW_JOB_STRIPED
#DW stage_out type=directory destination=/lustre/my_dir source=$DW_JOB_STRIPED

export JOBDIR=$DW_JOB_STRIPED

srun –n 3200 a.out
```

More detailed examples on the NERSC user web pages at:
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/

User library API example - libdatawarp

```
// module load datawarp   (to get access to the user library for building)
#include <datawarp.h>

// Get Info on DataWarp Configuration:
int r = dw_get_stripe_configuration(fd, &stripe_size, &stripe_width, &stripe_index);

// Use dw_stage_file_in function to move a file from PFS to DataWarp
int r = dw_stage_file_in(dw_file, pfs_file);

// Use dw_stage_file_out function to move a file from DataWarp to PFS
int r = dw_stage_file_out(dw_file, pfs_file, DW_STAGE_IMMEDIATE);

// Use dw_query_file_stage function to check stage in/out completion
int r = dw_query_file_stage(dw_file, &complete, &pending, &deferred, &failed);
```

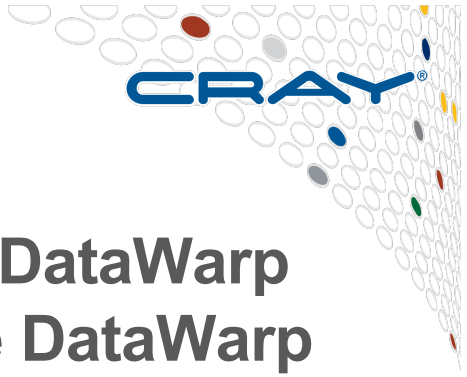
Command line example - dwstat



An example script for extracting job usage information from dwstat can be found at:
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-6

DataWarp Usage in WLMs

Overview



- WorkLoad Managers like Slurm understand DataWarp
- New #DW batch job script directives engage DataWarp Support
- Can request data be staged in to your burst buffer before your application processes run
- ...and staged out after your batch job exits
- Burst Buffer accessible via mount points on compute nodes, communicated via environment variables

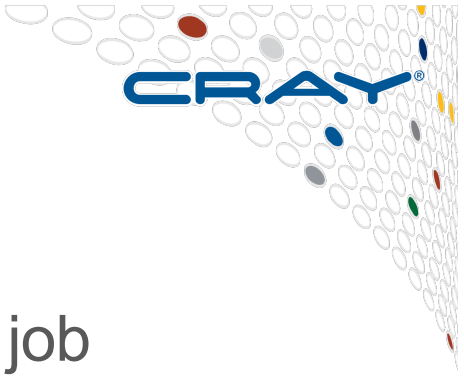
Job Script commands: Without & With DataWarp

```
#!/bin/bash
#SBATCH -n 3200 -t 2000

export JOBDIR=/lustre/my_dir

srun –n 3200 a.out
```

```
#!/bin/bash
#SBATCH -n 3200 -t 2000

#DW jobdw type=scratch access_mode=striped  capacity=1TiB
#DW stage_in type=directory source=/lustre/my_dir destination=$DW_JOB_STRIPED
#DW stage_out type=directory destination=/lustre/my_dir source=$DW_JOB_STRIPED

export JOBDIR=$DW_JOB_STRIPED

srun –n 3200 a.out
```

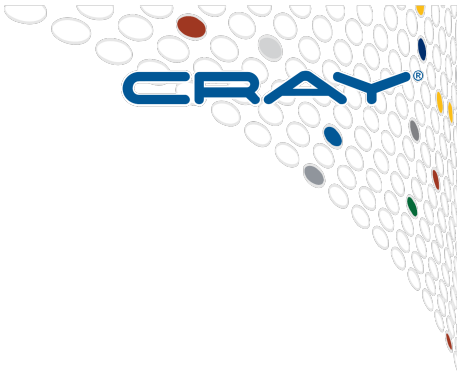
More detailed examples on the NERSC user web pages at:
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/

#DW jobdw ...



- **Requests a job DataWarp instance**
  - Lifetime of the instance is the same as batch job
  - Only usable by that batch job
- **capacity=<size>**
  - How much space do you need?
  - No control over server count. May need to request more space than you need.
- **type=scratch**
  - Selects use of DWFS filesystem

#DW jobdw ... (continued)

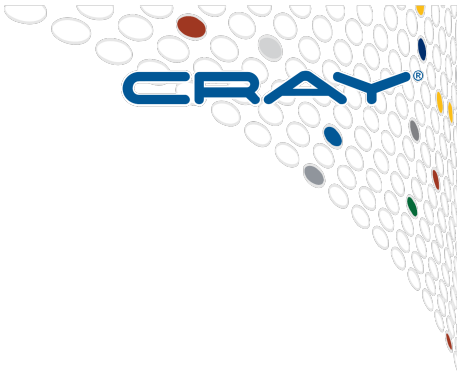


- **access_mode=striped**
  - All compute nodes see the same filesystem
  - Files are striped across all allocated DW server nodes
- **access_mode=private**
  - All compute nodes see a different filesystem
  - All filesystems share the same space allocation
  - Files are striped across all allocated DW server nodes
- **access_mode=striped,private**
  - Two mount points created on each compute node

#DW persistentdw ...

- **Requests use of a persistent DataWarp instance**
  - Lifetime different from the batch job
  - Usable by any batch job
- **name=xyz**
  - Name of persistent instance to use
  - scontrol show burst
  - ‘dwstat instances’ label column where public=true

#DW swap



- **Requests swap space**
- **nGiB**
  - Maximum size of swap for each compute node
  - Shares space with the rest of the instance
  - Space is allocated as needed
    - Running out of space for swap will kill the job

Accessing DataWarp for I/O

- **Mount points communicated through environment variables**
- **#DW jobdw access_mode=striped**
  - $DW_JOB_STRIPED
- **#DW jobdw access_mode=private**
  - $DW_JOB_PRIVATE
- **#DW persistentdw name=xyz**
  - $DW_PERSISTENT_STRIPED_xyz

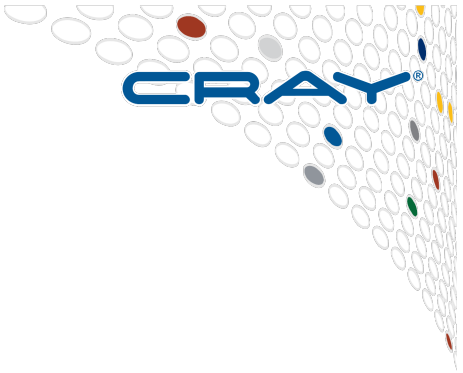
Staging

- **DataWarp Filesystem accepts requests to copy data into a namespace or out of a namespace**
- **Transfers performed asynchronous to request**
- **Concurrency tweakable via libdatawarp**
  - Concurrency controlled at the mount level
  - 512 per DWFS mount by default
- **Files can be marked for stage out at the end of the batch job's lifetime**

Staging with #DW stage_in and stage_out

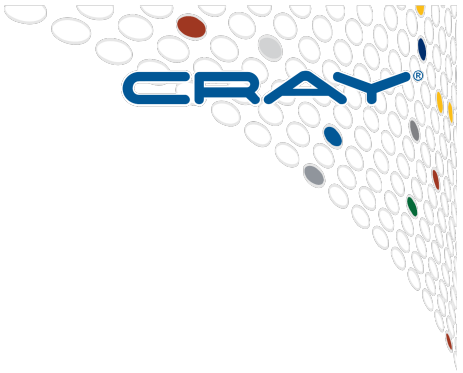
- Can request that I/O between PFS and BB happen before and after your batch job runs
- Only works for striped
  - Use libdatawarp for private
- stage_in: operation occurs before job runs
- stage_out: operation occurs after job runs
- type=file: line indicates transfer of a file
- type=directory: line indicates transfer of a directory
- type=list: line specifies a .txt file with multiple file directives

#DW stage_in type=file ...



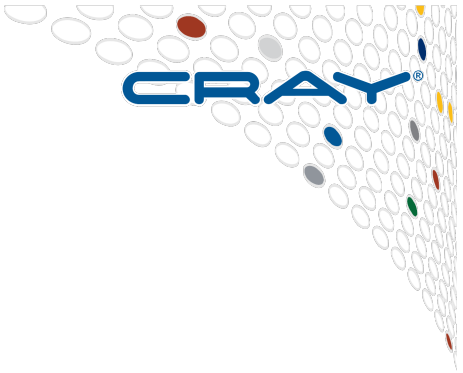
- **source=/lus/scratch/my-input-file**
  - Location of your file on the PFS
  - Must be a file or the stage_in will fail
- **destination=$DW_JOB_STRIPED/data/input-file**
  - Location in your BB where you wish to access your file

#DW stage_in type=directory ...



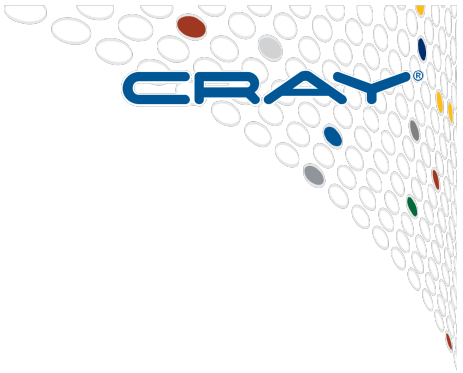
- **source=/lus/scratch/my-dir/**
  - Location of your directory on the PFS
  - Must be a directory or the stage_in will fail
- **destination=$DW_JOB_STRIPED/wherever/you/want/**
  - Location in your BB where you wish to access your directory

#DW stage_in type=list ...



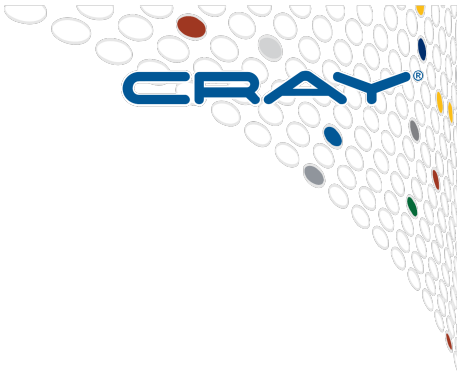
- **source=/some/file.txt**
  - A text file with ‘source=X destination=Y’ lines
- **Not useful at this time at NERSC**
  - WLM daemon processing directives needs to have access to the filesystem where the specified file lives

#DW stage_out type=file, directory, list

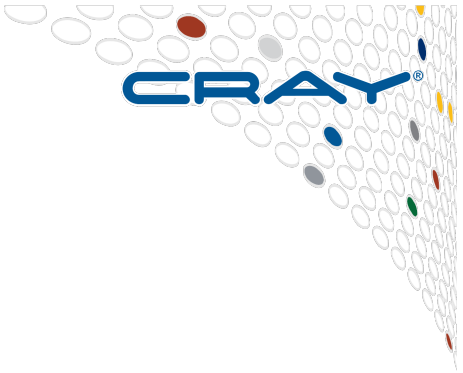


- **Nearly the same as stage_in**
- **destination is now PFS path**
  - destination=/lus/scratch/my-output-file
- **source is now BB path**
  - source=$DW_JOB_STRIPED/my-results

Problems with #DW stage_in or stage_out

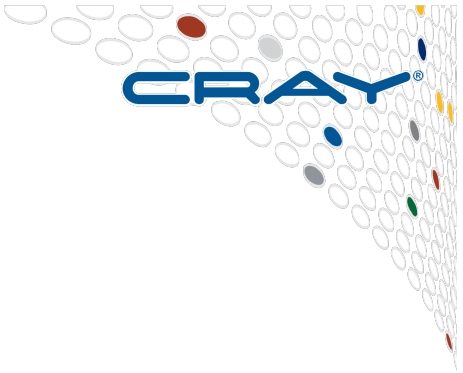


- Doesn't work with hard links
- Doesn't work with symbolic links
- Errors aren't reported to users (in log files for admins)
- Original low timeout values lead to failures when staging directories with many files



Using dwstat

dwstat - your view of DataWarp state



- Command line client to view DataWarp state
- Most useful dwstat commands for users
  - dwstat pools
  - dwstat instances
  - dwstat fragments

dwstat pools

- pool: name for pool
- units: future stuff (bytes for now)
- quantity: how much total space is in the pool
- free: how much of the total space is available for creating instances
- gran: granularity; space taken from the pool will be a multiple of the granularity

```
prompt:#> module load dws
prompt:#> dwstat pools
      pool units quantity       free      gran
test_pool bytes  5.82TiB   5.82TiB     16MiB
 wlm_pool bytes 832.5TiB 755.98TiB 212.91GiB
```

dwstat instances

```
prompt:#> module load dws
prompt:#> dwstat instances
inst state sess bytes nodes             created expiration intact label public confs
  29 CA---   36 16MiB     1 2015-08-21T13:10:05      never   true blast   true     1
  37 CA---   44 16MiB     1 2015-08-26T08:51:28      never   true I44-0  false     2
```

- **A request for DataWarp Space**
- **Interesting rows are the ones for persistent instances**
  - public=true
  - label is the name of the persistent instance
- **Other rows are instances that belong to you**
  - Created as part of a batch job

dwstat fragments

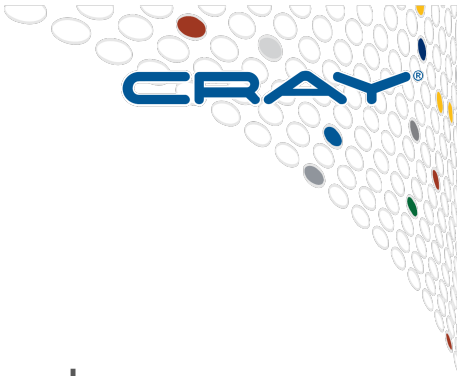
```
prompt:#> module load dws
prompt:#> dwstat fragments
frag state inst capacity     node
  38  CA--   29     16MiB nid00009
 108  CA--   37     16MiB nid00010
```

- A fragment is a portion of an instance on a DW server node
- Shows you on which nodes your instances reside
- Can show how many DW servers your job is using

An example script for extracting job usage information from dwstat can be found at:
http://www.nersc.gov/users/computational-systems/cori/burst-buffer/example-batch-scripts/#toc-anchor-6

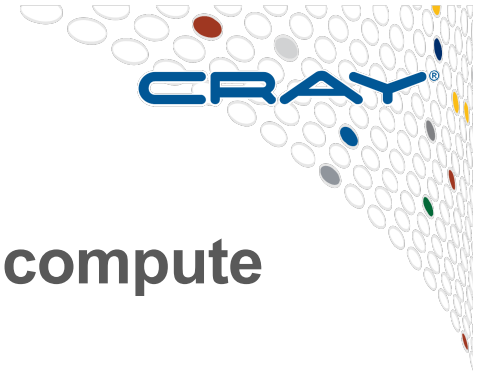
Usage Tips & Lessons Learned

Lessons Learned Page 1



- **Pool Granularity = Allocation unit size**
  - Set to different values on different systems
  - Often 200GB or higher on systems with lots of DW nodes
  - Use the dwstat command to find the value – warning: dwstat rounds up! (use dwstat pools -b to see exact byte count)
- **Job Private DW space**
  - Multiple compute nodes can share space on a single DW node
  - Designed for compute nodes using equal capacity and bandwidth
  - Can see different behavior if that is not the case
  - Wastes space for any amount more than one allocation unit size
  - Can provide better performance

Lessons Learned Page 2



- **DataWarp filesystems are only mounted on compute nodes**
  - Not accessible from login/mom nodes
- **No DVS Client Side Caching with this release**
  - Small I/O requests see high latency
  - Performance can be (much) worse than Lustre (seen up to 9x slower)

Stage_in and stage_out commands

- **Job Stage_In and Stage_out Commands**
  - Syntax is very specific
  - No Environment Variables or wildcards in file/directory names
  - Must keep entire command on one long line
  - Can use multiple stage in or stage out command lines in a job
  - Cannot stage in or out files with hard links or symlinks (See RFE 830185)
  - Need to specify full path for file or directory names
  - No messages to user if there are errors in the command line (errors may be put in logs visible to system admins)
  - ~~Currently only works with Lustre filesystem (no DVS projected filesystems like GPFS or NFS)~~

Workload Managers implement DataWarp support in different ways

- **SLURM**
  - Only supports one DW pool at this time (admin sets which one in config file)
  - Admins sets various config settings in “slurm.conf” file
  - Runs job script on first compute node, so users can directly access DW space via simple commands (cd, mkdir, df, ls, cp, mv, du, ...)
- **Moab/TORQUE**
  - Must use the “msub” command to submit DW jobs
  - Job script runs on mom node, and DW is only mounted on compute nodes, so users need to use aprun to run simple commands on DW space.

SSD write protection

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

Notes on SSD protection feature

- **SSD protection features can cause jobs to abort**
  - Customers expressed concern over rogue jobs "destroying" an SSD so Cray implemented some SSD protection features.
  - These can force the DW filesystem into “read only” mode. Error message is “KDWFS protection limit exceeded” in console log
  - By default, these limit the amount of data written to 10x the size of the requested instance in 24 hours. For small requested sizes, can hit this limit in 10 minutes of heavy I/O.
  - Users can tweak values with #DW syntax

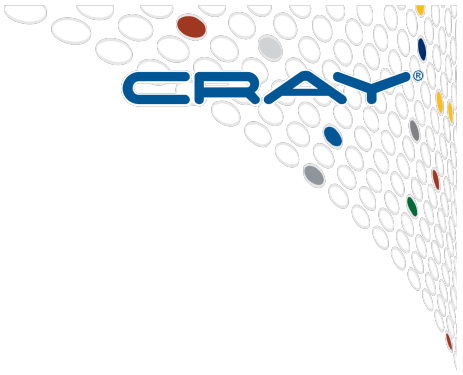
#DW jobdw options to adjust SSD protection [1]

- **write_window_length=<seconds>**
  - Number of seconds to use in write window threshold
- **write_window_multiplier=<integer>**
  - Integer multiples of capacity to use in write window threshold
- **(*capacity* * *write_window_multiplier*) bytes allowed to be written in any *write_window_length* second window before site-specific action taken**
  - Default: mark filesystem read only!
  - -EROFS

#DW jobdw options to adjust SSD protection [2]

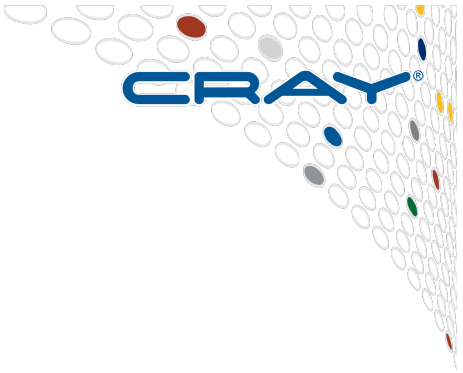
- **access_mode modifiers MFS and MFC**
  - Maximum File Size
  - Maximum Files Creatable
- **access_mode=striped(MFS=16777216)**
  - No file larger than this size (bytes) can be read/written
  - -EFBIG
- **access_mode=striped(MFC=1200)**
  - Any file creates after this number of files will fail
  - -EMFILE
- **access_mode=striped(MFS=16777216,MFC=1200)**
  - Can use both simultaneously

Upcoming Features



- Client Accounting
- Transparent Cache
- DVS Client-Side Cache

Client Accounting

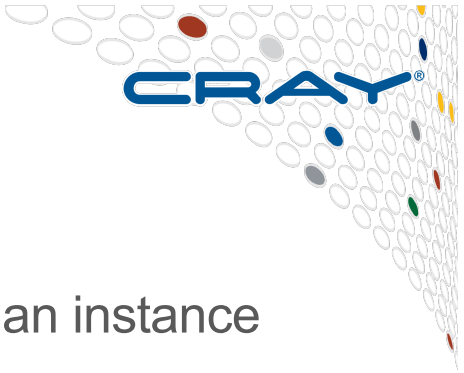


- DVS collects statistics on I/O activity
- RUR reports:
  - Bytes Read
  - Bytes Written
  - Count of files created
  - Maximum file offset read and written
  - This is all infrastructure, so we can add more if needed.

Transparent Cache

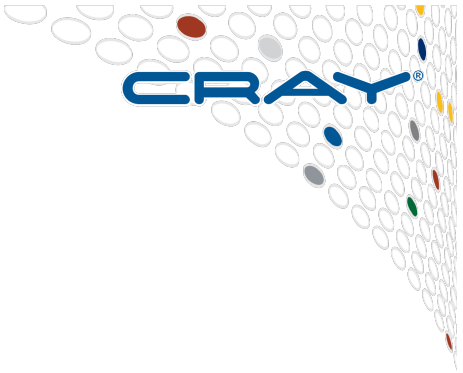
- DataWarp operates as a file-level cache of a PFS
- Applications see the directories of the PFS, not a private filesystem
  - DataWarp nodes will read-ahead files being read
  - Write-behind of files being written
    - Several flush options: close, fsync, as needed, and at the end of the job.
    - Application knobs to flush, invalidate, etc.

Two DataWarp Cache Access Modes



- **Striped ("Shared")**
  - Files are striped across all DataWarp nodes assigned to an instance
  - Files are visible to all compute nodes using the instance
  - Aggregates both capacity and bandwidth per file
  - One DataWarp node elected as the MDS per file
- **Load Balanced**
  - File contents are replicated on each DataWarp node
  - Compute nodes select which DataWarp node to use
  - Read-Only

Take-Away



- Try to use 512kb transfer sizes
- Have enough compute cores
  - 8 read streams per DataWarp node
  - 16 write streams per DataWarp node
- Try private