Using DataWarp

Glen Overby
Cray Inc., Bloomington, MN
Email: overby@cray.com

Abstract— **Cray DataWarp is a set of technologies that accelerates application I/O in order to reduce job wall clock time. It creates a near storage layer between main memory and hard disk drives. DataWarp uses direct attached solid-state disk (SSD) storage to provide more cost-effective bandwidth than an external parallel file system (PFS), allowing DataWarp to be provisioned for bandwidth and the PFS to be provisioned for capacity and resiliency. Placing this storage between the application and the PFS allows application I/O to be decoupled from (and in some cases eliminating) PFS I/O.**

Keywords- **Filesystems & I/O, DataWarp**

I. INTRODUCTION

Cray DataWarp™ is an infrastructure creating a near storage layer between main memory and hard disk drives. It uses direct attached solid-state disk (SSD) storage to provide more bandwidth than an external parallel file system (PFS) allowing DataWarp to be provisioned for bandwidth and the PFS to be provisioned for capacity and resiliency. Placing this storage between the application and the PFS allows application I/O to be decoupled from (and in some cases eliminating) PFS I/O.

DataWarp space is requested through the workload manager with a request syntax that is unique to DataWarp, but common across workload managers. The allocation request will specify the type of request, the amount of space to allocate, and other parameters related to its use. This paper will provide examples showing how to use this syntax with applications.

There are two major types: scratch and cache. The scratch access type is a space allocation with its own directory hierarchy; each instance is a filesystem separate from the system's parallel filesystem and all other DataWarp instances. All files are contained on the SSD and any data to be copied to/from a parallel filesystem by DataWarp must be specified in the batch job or initiated by the application. Cache access type is a buffering layer in front of a parallel filesystem, much like a processor's cache is a buffering layer in front of main system memory. At the end of the batch job, any modified file data will be automatically written back to the PFS.

II. SCRATCH USAGE

A scratch filesystem is requested by specifying it and its size in special directives in a batch job. An application accesses the scratch filesystem through the environment variable $DW_JOB_STRIPED. DataWarp, through the batch scheduler, sets this environment variable to point to the directory on the compute nodes where the DataWarp filesystem can be accessed. An application or job script may need to be modified to use this directory.

The examples in this paper will use the IOR benchmark as an example application.

The first example uses two separate job steps each running IOR to write a file and check the data. Access to the filesystem that DataWarp presents on the compute nodes is done through an environment variable. Each different access mode uses a different environment variable. This request is for a per-job striped space and uses the environment variable $DW_JOB_STRIPED. This space is mounted only on the compute nodes (not on the login or batch MOM nodes).

At the end of the job, the files remaining on the filesystem are deleted when the SSD space is freed to the DataWarp pool.

```
   #!/bin/bash
   #MSUB -l nodes=16:ppn=4
   #MSUB -l walltime=1:00:00
   #DW jobdw type=scratch access_mode=striped
capacity=2TiB

   aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e
-o      $DW_JOB_STRIPED/ior_example_1      -G
1234567890 -w -k

   aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e
-o      $DW_JOB_STRIPED/ior_example_1      -G
1234567890 -W
```

DataWarp has a mechanism called staging where it will copy data to or from the PFS on behalf of the user. A batch job can specify what data is copied into DataWarp prior to the job being launched, and the copy out being done after the job completes. An application can make calls to the DataWarp library to request files be staged in while the application runs, and to mark files to be staged out, either while the application is running, or after the job completes. The staging happens asynchronously to the application's execution and I/O.

The next example extends the first example by adding staging out of the generated data file to a Lustre filesystem:

```
#!/bin/bash
#MSUB -l nodes=16:ppn=4
#MSUB -l walltime=1:00:00
#DW jobdw type=scratch access_mode=striped capacity=2TiB
#DW stage_out type=file destination=/lus/scratch/overby/example_1 source=$DW_JOB_STRIPED/ior_example_1

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/ior_example_1 -G 1234567890 -w -k

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/ior_example_1 -G 1234567890 -W -k
```

The source line must start with the DataWarp space to be staged from. Even though the name looks like a shell environment variable, on the #DW lines it really isn't. The name can only be one of the variables for striped type storage.

When this example job completes, the data file will be copied out to the PFS by the DataWarp service nodes as directed by the stage_out directive. If the job is sized to use many DataWarp service nodes, all of them will participate in copying the data.

A directory hierarchy can be copied by changing type=file to type=directory.

The next example uses staging to copy a data file in, and to copy a directory of results out.

Running IOR with the following options creates the input data file:

```
IOR -o /lus/snx11108/overby/ex3_data -k -v -b 64m -t 1m -E -C -w -G 1248
```

The batch job for this example is:

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=1
#SBATCH --job-name=example3
#DW jobdw type=scratch access_mode=striped capacity=128GiB
#DW stage_in type=file source=/lus/snx11108/overby/ex3_data destination=$DW_JOB_STRIPED/input
#DW stage_out type=directory destination=/lus/snx11108/overby/results3 source=$DW_JOB_STRIPED/output

echo DW_JOB_STRIPED $DW_JOB_STRIPED

srun -n 16 IOR -o $DW_JOB_STRIPED/input -k -v -b 64m -t 1m -E -C -W -r -G 1248

srun -n 1 mkdir  $DW_JOB_STRIPED/output

srun -n 16 IOR -o $DW_JOB_STRIPED/output/res -k -v -b 64m -t 1m -E -C -w -G 163264 -F
```

A goal of using DataWarp this way is to reduce the amount of time that a batch job requires on a system's compute nodes by staging in data prior to the job start, where it can be read from the faster SSDs, and writing the job output to SSD and staging it out to the PFS after the job has ended.

Applications can initiate or schedule staging directly, using an API. The API calls are found in libdatawarp and can be linked against by doing 'module load datawarp' or calling pkg-config cray-datawarp:

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);
```

Stage out can be immediate, or deferred to the end of the job.

One use case is for staging checkpoint files out of DataWarp. After a checkpoint file is written, the files can be marked for a deferred stage out, and the previous checkpoint files have their stage out cancelled (or the files deleted). This way if the application aborts, its latest checkpoint will be copied out to the PFS for later use in restarting (this can be extended to multiple checkpoints).

A. *Persistent instances*

DataWarp also provides storage that can be used across jobs, including jobs run by different users. The storage is requested outside of a batch job, and remains until it is deleted, or optionally its lifetime expires. Any user can request a persistent instance subject to the workload manager configuration, but standard POSIX file permissions still apply to the files.

However, failure of one SSD will result in losing the data on it. Keep a backup copy of any critical data on a parallel filesystem! Staging can be used to make a backup copy.

Each workload manager has a different way of creating persistent instances.

When using the SLURM workload manager, a persistent instance is requested by submitting a job with the following request:

```
   #BB    create_persistent    name=overby123
capacity=1GiB access=striped type=scratch
```

The following example requests access to two persistent instances. The persistent instance is named on a #DW line, and the environment variable set by DataWarp is $DW_PERSISTENT_STRIPED_name where the name is the name of the persistent instance.

```
   #!/bin/bash
   #MSUB -l nodes=16:ppn=4
   #MSUB -l walltime=1:00:00
   #DW persistentdw name=overby123
   #DW persistentdw name=overby987

   echo        DW_PERSISTENT_STRIPED_overby123
$DW_PERSISTENT_STRIPED_overby123
   echo        DW_PERSISTENT_STRIPED_overby987
$DW_PERSISTENT_STRIPED_overby987

   aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e
-o $DW_PERSISTENT_STRIPED_overby123/input1 -G
1234567890 -w -W -r

   aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e
-o $DW_PERSISTENT_STRIPED_overby987/input2 -G
1248163264 -w -W -r
```

Staging can be used to copy data into or out of a persistent instance by specifying the persistent instance's reference variable in the source or destination:

```
   #!/bin/bash
   #MSUB -l walltime=1:00:00
   #MSUB -l nodes=1
   #DW persistentdw name=overby123
   #DW persistentdw name=overby987
   #DW           stage_in           type=file
source=/lus/dal/overby/example_1_in
destination=$DW_PERSISTENT_STRIPED_overby123/
example_1B_on_dw
   #DW           stage_in           type=file
source=/lus/dal/overby/example_1_in
destination=$DW_PERSISTENT_STRIPED_overby987/
example_1_on_dw
   #DW          stage_out           type=file
destination=/lus/dal/overby/example_1_out
source=$DW_PERSISTENT_STRIPED_overby123/examp
le_1_on_dw
```

B. *Private Access Mode*

An alternative to stripe access mode is private. A private type instance has a directory hierarchy for each compute node, but the SSD space is shared. A use case for this is an application that uses temporary files, but requires more temporary space than /tmp, a ram disk on the compute node, provides. With a private access type the application on each node can use the filesystem without concern for having filename collisions.

Data can be restricted to one DataWarp service node, or striped across all service nodes.

Each namespace has its own metadata server (MDS), and the MDS are spread across the service nodes that are used for the filesystem. This is the primary difference between striped and private.

```
   #!/bin/bash
   #MSUB -l nodes=16:ppn=4
   #MSUB -l walltime=1:00:00
   #DW jobdw type=scratch access_mode=private
capacity=100GiB

   echo DW_JOB_PRIVATE $DW_JOB_PRIVATE

   aprun -n 1 df -h $DW_JOB_PRIVATE
```

Example that combines private and striped:

```
   #!/bin/bash
   #MSUB -l nodes=16:ppn=4
   #MSUB -l walltime=1:00:00
   #DW           jobdw           type=scratch
access_mode=striped,private capacity=100GiB

   echo DW_JOB_STRIPED $DW_JOB_STRIPED
   echo DW_JOB_PRIVATE $DW_JOB_PRIVATE

   aprun    -n    1    df    -h    $DW_JOB_STRIPED
$DW_JOB_PRIVATE
```

III. CACHE USAGE

A cache filesystem is requested by specifying its size and a backing PFS directory in the DataWarp job directives.

When using the cache access type, a directory on a parallel filesystem must be specified in the *pfs* parameter. The directory listed in the *pfs* parameter must be a directory that is in a white list of permitted directories. This is typically the mount points of Lustre filesystems. The white list is needed to prevent circumventing directory permissions.

# Before submitting the job, create a data file with a job that runs this command:

aprun -n 16 IOR -o /lus/snxs1/overby/example_3_input -k -v -b 64m -t 1m -E -C -w -G 163264

```
   #!/bin/bash
   #MSUB -l nodes=4:ppn=4
   #MSUB -l walltime=1:00:00
   #DW  jobdw  type=cache  access_mode=striped
capacity=4GiB pfs=/lus/snxs1

   echo                 DW_JOB_STRIPED_CACHE
$DW_JOB_STRIPED_CACHE

   aprun       -n       1       ls       -al
$DW_JOB_STRIPED_CACHE/overby/example_3_input
```

```
   # Read a file on the PFS through cache
   aprun       -n       16       IOR       -o
$DW_JOB_STRIPED_CACHE/overby/example_3_input
-k -v -b 64m -t 1m -E -C -W -G 163264

   # Write a file to cache and read it back
   aprun       -n       16       IOR       -o
$DW_JOB_STRIPED_CACHE/overby/example_3_cache
-k -v -b 64m -t 1m -E -C -w -G 163264 -W
```

A. *Load Balance*

Cache can alternatively be used in a load-balanced configuration, where each service nodes cache data independently of each other, providing multiple cache points across the system. Each service node serves a subset of the compute nodes in the job. This configuration is read-only.

This mode is useful for things like executables, libraries, and shared data.

```
   #!/bin/bash
   #MSUB -l nodes=32:ppn=4
   #MSUB -l walltime=1:00:00
   #DW jobdw type=cache access_mode=ldbalance
capacity=2GiB pfs=/lus/scratch

   echo                 DW_JOB_LDBAL_CACHE
$DW_JOB_LDBAL_CACHE

   aprun -n 1 ls -al $DW_JOB_LDBAL_CACHE
   aprun -n 1 df -h $DW_JOB_LDBAL_CACHE
```

REFERENCES

[1] Dave Henseler, Benjamin Landsteiner, Doug Petesch, Cornell Wright, and Nicholas J. Wright, “Architecture and Design of Cray DataWarp,” *Proc. Cray Users’ Group Technical Conference (CUG)*, 2016

[2] (2015) Cray DataWarp User’s Manual, Cray Inc. [Online] Available: http://docs.cray.com/books/S-2558-5204

[3] (2015, November) Slurm Burst Buffer Guide. SchedMD LLC. [Online]. Available: http://slurm.schedmd.com/burst buffer.html

Example #1: Using DataWarp between job steps

```
#!/bin/bash
#MSUB -l nodes=16:ppn=4
#MSUB -l walltime=1:00:00
#DW jobdw type=scratch access_mode=striped capacity=2TiB

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/ior_example_1 -G 1234567890 –w -k

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/ior_example_1 -G 1234567890 –W
```

Example #2: Stage out of results

```
#!/bin/bash
#MSUB -l nodes=16:ppn=4
#MSUB -l walltime=1:00:00
#DW jobdw type=scratch access_mode=striped capacity=2TiB
#DW stage_out type=file destination=/lus/scratch/overby/example_1 source=$DW_JOB_STRIPED/ior_example_1

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/example_1 -G 1234567890 –w -k

aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_JOB_STRIPED/example_1 -G 1234567890 –W -k
```

Example #3

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=1
#SBATCH --job-name=example3
#DW jobdw type=scratch access_mode=striped capacity=128GiB
```

```
    #DW stage_in type=file source=/lus/snx11108/overby/ex3_data destination=$DW_JOB_STRIPED/input
    #DW stage_out type=directory destination=/lus/snx11108/overby/results3 source=$DW_JOB_STRIPED/output

    echo DW_JOB_STRIPED $DW_JOB_STRIPED

    srun -n 16 IOR -o $DW_JOB_STRIPED/input -k -v -b 64m -t 1m -E -C -W -r -G 1248

    srun -n 1 mkdir  $DW_JOB_STRIPED/output
    srun -n 16 IOR -o $DW_JOB_STRIPED/output/res -k -v -b 64m -t 1m -E -C -w -G 163264 –F
```

Example 4

```
    #!/bin/bash
    #MSUB -l nodes=16:ppn=4
    #MSUB -l walltime=1:00:00
    #DW persistentdw name=overby123
    #DW persistentdw name=overby987

    echo DW_PERSISTENT_STRIPED_overby123 $DW_PERSISTENT_STRIPED_overby123
    echo DW_PERSISTENT_STRIPED_overby987 $DW_PERSISTENT_STRIPED_overby987

    aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_PERSISTENT_STRIPED_overby123/input1 -G 1234567890 -w -W
-r

    aprun -n 64 IOR -a POSIX -g -b 8G -t 1M -e -o $DW_PERSISTENT_STRIPED_overby987/input2 -G 1248163264 -w -W
-r
```

Example 5

```
    #!/bin/bash
    #MSUB -l walltime=1:00:00
    #MSUB -l nodes=1
    #DW persistentdw name=overby123
    #DW persistentdw name=overby987
    #DW stage_in type=file source=/lus/dal/overby/example_1_in
destination=$DW_PERSISTENT_STRIPED_overby123/example_1B_on_dw
    #DW stage_in type=file source=/lus/dal/overby/example_1_in
destination=$DW_PERSISTENT_STRIPED_overby987/example_1_on_dw
    #DW stage_out type=file destination=/lus/dal/overby/example_1_out
source=$DW_PERSISTENT_STRIPED_overby123/example_1_on_dw
```

Example 6

```
    #!/bin/bash
    #MSUB -l nodes=16:ppn=4
    #MSUB -l walltime=1:00:00
    #DW jobdw type=scratch access_mode=private capacity=100GiB

    echo DW_JOB_PRIVATE $DW_JOB_PRIVATE

    aprun -n 1 df -h $DW_JOB_PRIVATE
```

Example 7: combine private and scratch:

```
    #!/bin/bash
    #MSUB -l nodes=16:ppn=4
    #MSUB -l walltime=1:00:00
    #DW jobdw type=scratch access_mode=striped,private capacity=100GiB

    echo DW_JOB_STRIPED $DW_JOB_STRIPED
    echo DW_JOB_PRIVATE $DW_JOB_PRIVATE
```

```
aprun -n 1 df -h $DW_JOB_STRIPED $DW_JOB_PRIVATE
```

Example 8

```
#!/bin/bash
#MSUB -l nodes=4:ppn=4
#MSUB -l walltime=1:00:00
#DW jobdw type=cache access_mode=striped capacity=4GiB pfs=/lus/snxs1

echo DW_JOB_STRIPED_CACHE $DW_JOB_STRIPED_CACHE

aprun -n 1 ls -al $DW_JOB_STRIPED_CACHE/overby/example_3_input

# Read a file on the PFS through cache
aprun -n 16 IOR -o $DW_JOB_STRIPED_CACHE/overby/example_3_input -k -v -b 64m -t 1m -E -C -W -G 163264

# Write a file to cache and read it back
aprun -n 16 IOR -o $DW_JOB_STRIPED_CACHE/overby/example_3_cache -k -v -b 64m -t 1m -E -C -w -G 163264 -W
```

Example 9

```
#!/bin/bash
#MSUB -l nodes=32:ppn=4
#MSUB -l walltime=1:00:00
#DW jobdw type=cache access_mode=ldbalance capacity=2GiB pfs=/lus/scratch

echo DW_JOB_LDBAL_CACHE $DW_JOB_LDBAL_CACHE

aprun -n 1 ls -al $DW_JOB_LDBAL_CACHE
aprun -n 1 df -h $DW_JOB_LDBAL_CACHE
```