epcc

Hands on with Object Stores

Exercises

Adrian Jackson

1 Introduction

This sheet details the practical exercises you can undertake within this tutorial. It also describes the system we are using and how to access it. For the first exercises we provide pre-compiled applications for you to run, but you will get the chance to implement the source code for this application in the later exercises.

2 Using the system

For this tutorial we are using the Google (GCP) cloud. We have created a cloud virtual machine (VM) that will act as a login node and batch system host for compute VMs, along with Ceph and Parallelstore (Google's deployment of DAOS) storage systems that can be used from the login node and the compute VMs. You will need to provide us with an email address and then we will send you an SSH key, username, and IP address for the system. You can then access the system as follows (**note: the username you use will be different from below, this will be emailed to you along with the ssh key**):

```
ssh -AX -I ssh_key -J ngguestXX@gateway.epcc.ed.ac.uk ngguestXX@nextgenio-login1
```

The system is configured with a login node separate from the compute nodes in the system. To get the initial compiler and MPI libraries setup on the system you should do the following when you log on:

```
module load compiler mpi gnu/11.2.0
```

We use the Slurm batch system to access and enquire about the compute nodes. You can discover how many compute nodes there are using the following command:

```
sinfo
```

Or:

```
sinfo -N -l
```

Below is an example of a Slurm batch script we can use to run a job:

```
#!/bin/bash

#SBATCH --job-name=test_job
#SBATCH --output=test.%A.out
#SBATCH --error=test.%A.err
#SBATCH --tasks-per-node=48
#SBATCH --cpus-per-task=1
#SBATCH --nvram-options=1LM:1000
```

```
#SBATCH --time=00:05:00
#SBATCH --nodes=2

srun hostname
```

To run a job on the system we use the `sbatch` command, i.e. (assuming the script above is called `runtestjob.sh`)

```
sbatch runtestjob.sh
```

You can `squeue` to see running jobs (`squeue -u $USER` will show only your jobs) and `scancel` to cancel a job.

The `srun` command in the script above is the MPI job launcher which runs the executable on the selected number of nodes. The default MPI library being used is the OpenMPI library. We are specifying the number of processes we want to run using the Slurm `sbatch` configuration. The batch script above runs 48 processes per node and is requesting 2 nodes, meaning it will run the application on 96 processes spread across 2 nodes. If you want to vary the number of processes you run an application on you can change the number of nodes, i.e. this line:

```
#SBATCH --nodes=2
```

Or you can change the number of tasks per node, i.e. this line:

```
#SBATCH --tasks-per-node=48
```

A combination of these will specify the overall number of processes to use.

2.1 Available Hardware

Currently we have 20 compute nodes configured in this system with 10 separate nodes running the DAOS.You can try running jobs up to 20 compute nodes, or 960 processes in total.

3 Exploring filesystem performance

To get started on the system copy the following software into your home directory:

```
git clone https://github.com/adrianjhpc/ObjectStoreTutorial
```

Change to the `ObjectStoreTutorial/Exercises` directory, i.e.:

```
cd ObjectStoreTutorial/Exercises
```

You should be able unpack with the command:

`tar xf IOR.tar.gz`.

For IOR you need to go into the `ior` directory and type:

```
make
```

Then you can run a DAOS filesystem IOR benchmark using:

```
sbatch daos_dfs_ior.sh
```

The aim of this exercise is to run the different IOR benchmarks and compare the performance that DAOS provides. IOR is a common I/O benchmark designed to explore the maximum bandwidth a filesystem can provide to a parallel programme. It uses MPI to run many workers (processes) at once, and reports back the total bandwidth achieved for bulk I/O operations. It can be configured in various ways, but we are looking at relatively large read and write sizes to investigate filesystem performance.

Currently the batch script is setup to run on two nodes, but you can vary this to investigate how well both filesystems scale. The total number of nodes available is 20, and each has 48 cores, so the most you would be able to run is 960 MPI processes.

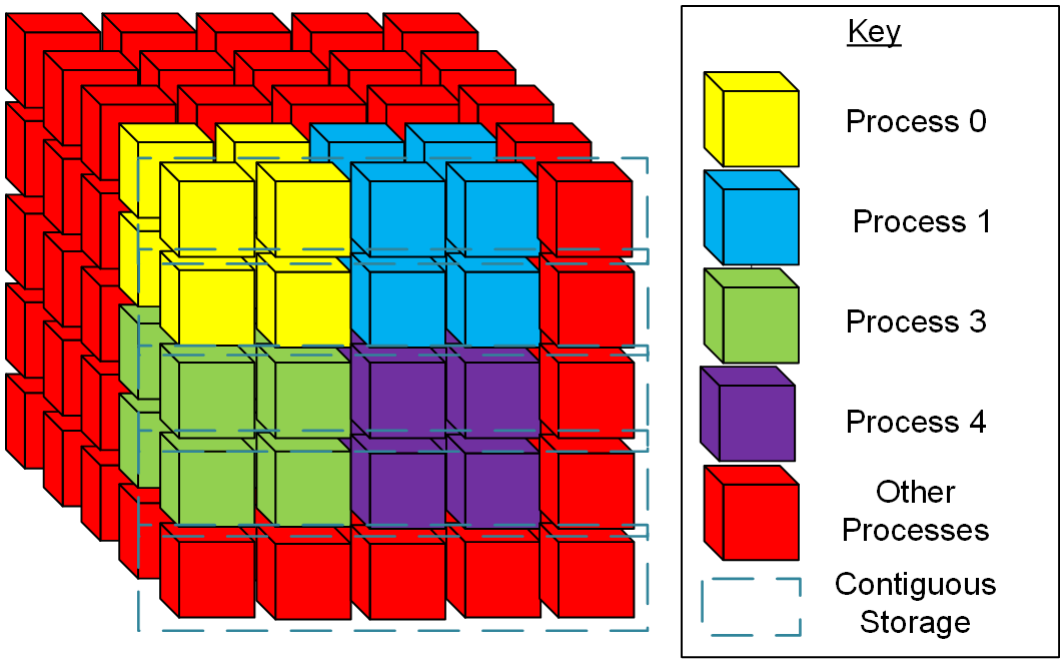
We only have DAOS configured to provide a filesystem interface at the moment (through the DFS interface). For this initial exercise simply run the `daos_dfs_ior.sh` batch file provided using varying number of nodes from 2 to 16 to see what read and write bandwidth you get.

4 Exploring DAOS

There is a separate exercise sheet/presentation in the GitHub repository for the tutorial, under the `ObjectStoreTutorial/Exercises/DAOS/Handout` directory. You can follow this for the DAOS API practical exercises.

5 Porting an application to object storage technologies

To give you experience of what is required to port a larger application to an object store we have provided you with an application that uses HDF5 to store data to a file. It follows a relatively common pattern for file I/O for large scale parallel applications, i.e. each process has it's own portion of a share dataset that it writes to a common file, an example of this is illustrated in this figure:



The question is; how do you map this pattern from a single file to an object store. There are a range of options, from creating a key-value pair (or object) for each element in the array, to creating a single array for all the data shared by all processes. What you actually do depends on how you'd like to be able to index/share/search that

data in the future. If you simply need to find the full dataset in the future than you could create a single array, although not all object stores give good performance for a single large array (i.e. Ceph might be limited in how big this array could be). You can also create an array per process, and then indexes that track those to enable searching/using the full dataset. You may also want to create an array/object per row or per column of data, or across some other decomposition that makes sense for your application.

If you know a bit about HDF5 you will be able to see that the current application we have provided uses a hyper-slab approach, which is writing the individual parts of data held by each process into a single shared dataset, ignoring the edges of the data held locally (data used for MPI halos) and ensuring it ends up in a single file.

We have a number of choices when porting this approach to an object store. DAOS supports using an array object, where each process writes their part of the array into the object, ending up with the same situation as the HDF5 hyperslab, albeit as an object rather than a file. It is possible to do this in Ceph but it is a bit more involved.

Another approach would be to write a single array or object per process, but then create a set of keys or objects that let you identify each part of the data and search/query them. This is possible in DAOS. For this practical we have provided you with two different skeleton files that you can use to implement these approaches, as well as the original HDF5 implementation:

- `hdf5.c`: The original HDF5 file
- `daos_individual.c`: The equivalent code and instructions for implementing this using the DAOS api but with an array per process
- `daos_array.c`: Code to do the same but using a single DAOS array for all process.

There are also batch scripts (the files ending in `*.sh`) and a `Makefile` to build them all. Your task is to take one (or more) of the DAOS examples and complete the code to get it to write the dataset to the object store(s). The code is commented with what needs to be added (functions and function arguments) and you can build and run them through the Slurm batch system.

This task will involve a bit of reading the code and looking up the function call syntax for the file you have chosen, but you can ask the instructors or email/message us if you have any questions. We will release sample solutions for each of these after the tutorial has finished.

Thanks for attending and we hope it was useful!