OpenSFS Benchmarking Working Group (BWG)

File system hero run best practices task

White paper

Zero to Hero[1]

Starting from scratch

High-performance computing (HPC) systems provide value through their ability to parallelize computational jobs and reduce the time to reach solutions. It is natural to look for performance metrics; users need to know if a given system can run their job, managers want to know if they received good value from their vendor, and so on.

When a system is procured, the vendor often provides peak performance numbers. These are based on the theoretical maximum for each component. Storage benchmarks can often run at 80% or more of the theoretical peak. Good results are almost never achieved on the first attempt. What follows here is a high-level guide for improving benchmark results to get to the desired result.

First attempts

A normal test of read and write rates begins with a program creating random data and writing it to a file. The new file can then be read by the program (and redirected to `/dev/null` for simplicity).

---

[1] By Ben Evans, TeraScala, November 2014.

The first time you try this, expect to be underwhelmed. It is possibly far from the peak that you expected. What could be wrong? Where is the bottleneck? Is the problem with I/O parameters such as the block size, or is there a problem with parallelism (e.g. number of I/O worker threads/processors or I/O targets/devices)?

Start over and restrict your test to one thread and one processor and one I/O device. Determine the optimal parameters for this basic configuration. Then, see if increasing the number of threads on a single processor helps. See if adding another processor helps. See if you can double the speed when you are using two I/O devices. Find out if one processor can drive two I/O devices at twice the speed of one device.

dd

If you have benchmarked I/O performance before, you are probably familiar with the `dd` command. If not, it's time to go to the `man` page for `dd` and check it out. Then start with a simple `dd` test, such as:

```
dd if=/dev/zero of=/mnt/filesystem/testdir/file bs=4k
count=1m
```

Compare your test program results (if you started with another test program) with `dd` results. Vary the block size. You should see improvement as you start to add threads, but the returns are probably diminishing. Check to make sure your network is not the bottleneck.

Adding more clients

It is time to add another client to the test. Take the scripts and configurations you've got from your single client `dd` script and run on multiple clients in parallel and concurrently. As you increase the number of clients and threads, the limitations of `dd` are becoming obvious, it's getting hard to script all this and keep things coordinated.

Stepping it up: Running IOR on clients

The first thing to do with IOR is to get it compiled and running. Using the parameters you found best with `dd`, run IOR and confirm that with 4 or 8 processors, IOR gives results that are very close to those from an ensemble of `dd` runs.

Simple IOR structures using POSIX interfaces are provided below as references. Please modify these to fit your system and environment.

Streaming I/O

```
mpirun -np {NP} -wd {working dir} \\
-machinefile {machine file} \\
{IOR pathname}/IOR -v -a POSIX -i5 -g -e -w -r \\
-b {filesize}g -T 10 -F -C -t 4m
```

Random I/O

```
mpirun -np {NP} -wd {working dir} \\
-machinefile {machine file} \\
{IOR pathname}/IOR -v -a POSIX -i5 -g -e -w -r \\
-b {filesize}g -T 10 -F -z -t 4k
```

where:

{NP} is Total number of threads in the test
{working dir} is the directory where files get created
{machine file} is a list of clients involved in the test, this is either created by hand, or by a scheduler
{IOR pathname} is the full path to IOR, this must be accessible on all clients
{filesize} is the size of the file each thread will use. This should be a multiple of client memory, to prevent cache effects.

Other parallel I/O benchmarking tools

IOR, IOzone and xdd are the tools that the Hero Run task group uses. Select your favorite and become familiar with it. Write scripts, run them on your cluster, and use them for testing when you get new hardware.

These codes are also useful for diagnostics. For example, if you have one drive group running at half speed and seven drive groups running at full speed, you will have difficulty finding the slow drive group if all of the tests you run are eight-way parallel.