dt Performance Workloads

1. Overview

With more and more folks using dt (thank you!), multiple people have asked about best performance options to quickly fill terabyte (TB) storage.

This document will outline the key dt options and describe similar performance options for other tools.

Note: dt was **not** designed for performance testing, but rather for device qualification with high data validation.

2. Performance

There are many factors that influence the ability to get optimal performance, and while I don't consider myself a performance expert, and do know a few key things to acquire better performance.

2.1. Hardware Configuration

1. Use *real* hardware, not virtualized anything.
2. Use a "beefy" host, with sufficient resources (multiple cores, sufficient memory, and storage connectivity)
3. Know the limitations of the storage under test, since overdriving low/mid-range will kill your performance.
4. Know your storage connectivity port speeds, and how many paths the host will utilize for sending I/O.

2.2. Software Configuration

1. Multiple paths configured with Linux DM-MP or Windows MPIO.
2. While some OS's are designed for performance, like AIX, most of us only use Linux/Windows.
3. Verify for yourself using dt, fio, and/or vdbench or your favorite I/O tool.
4. Know which tools you'll need to monitor array and host performance.

3. dt Performance Guidelines

The key *dt* options that improve performance are:

1. Multiple threads/slices (with rotating disks, multiple threads w/iolock is recommended)
2. Larger block sizes (varies by storage, but a block size of 256k may be a good starting point)
3. Sequential I/O, unless measuring Random I/O performance (please use same random seed)
4. Asynchronous I/O, this can help on some OS's, but POSIX AIO on Linux generally will **not** help!
5. Proper data pattern, some pattern options create more CPU load (block tags and IOT, for example)
6. Consider whether compression and/or deduplication will be involved with the data pattern chosen.
7. If only interested in write performance, disable the read verify pass and buffer fills per write.

3.1.1. dt Performance Workload

At the time of this I/O tool performance comparison, my LUN is 5TB. Data limits were imposed to reduce runtimes.

Here's a workload I found provided the best performance for **Linux iSCSI with two paths each 10Gbit**:

dt of=/dev/mapper/mpathdw bs=256k disable=verify limit=625g slices=25 workload=job_stats_only workload=log_timestamps

This achieved **883.053 Mbytes/sec** and **3532.210 I/O's per second**, with an **elapsed time of 12m4.794s.**

Disabling write buffer prefills via **disable=compare,verify,poison** did *not* affect overall performance. (YMMV)

Now I wish I could tell you an exact block size and/or slice count, but I'm afraid this came from "trial and error". 😊

Many years ago, while working on Tru64 Unix, I found a block size of 64k provided the best performance.

Note: I ran the same workload on Windows, including AIO's, but Linux achieved higher performance, so I'm excluding those results (for now).

4. fio Performance Workload

This is the workload I used for the *fio* tool:

/usr/bin/fio --rwmixread=0 --rwmixwrite=100 --thread --group_reporting --continue_on_error=none --status-interval=600 --direct=1 --rw=write --iodepth=25 --size=625g --blocksize=256k --refill_buffers --filename=/dev/mapper/mpathdw --name=RobinTest --ioengine=libaio

This provided **849MiB/s** and an average **3394.34 I/O's per second,** with an **elapsed time of 12m35.289s**

While the native Linux libaio generally boosts performance, this fio workload proved to be slightly faster: (**psync** engine is used by default, but as you may know, *fio has* many I/O engines)

/usr/bin/fio --rwmixread=0 --rwmixwrite=100 --thread --group_reporting --continue_on_error=none --status-interval=600 --direct=1 --rw=write --iodepth=25 --size=25g --blocksize=256k --refill_buffers --filename=/dev/mapper/mpathdw --name=RobinTest --numjobs=25

This provided **869MiB/s** and **35705.59 I/O's per second**, with an elapsed time of **12m17.186s**, so higher performance but longer elapsed time (go figure!).

5. vdbench Performance Workload

Note: I'm not a vdbench user, but I snarl'ed a started workload from automation and augmented it thusly:

cat 256K_SEQ_WRITE_parameters_file

monitor=/local/vdbench/vdbench50406/1602865053.5186489_monitor_file

compratio=2

hd=<host name>,system=<system name>,vdbench=/local/vdbench/vdbench50406,**user=Administrator**,shell=vdbench

5TB LUN

sd=sd1,host=<host name>,lun=/dev/mapper/mpathdw,openflags=o_direct,**size=1t,threads=25**

wd=wd_sd1_1,sd=sd1,**rdpct=0**,seekpct=0,**xfersize=256k**

rd=rd1,wd=*,iorate=max,interval=10,warmup=10,elapsed=1h,**maxdata=625g**

 # time /local/vdbench/vdbench50406/vdbench -f 256K_SEQ_WRITE_parameters_file

The last performance report showed:

Oct 19, 2020 interval i/o MB/sec bytes read resp read write resp resp queue cpu% cpu%

rate 1024**2 i/o pct time resp resp max stddev depth sys+u sys

12:25:37.045 177 2034.10 508.53 262144 0.00 12.192 0.000 12.192 48.610 5.467 24.8 6.8 4.4

12:25:37.052 **Reached maxdata=625g. rd=rd1 shutting down after next interval.**

This completed in an elapsed time of **29m34.433s**, thus the performance was not stellar. Likely needs different parameters.