BeeGFS® Benchmarks on HP Apollo 4200 Gen9 Server

Ely de Oliveira

December 2015 – v 1.0

Table of Contents

1. Overview

This document presents a brief summary of the results of a series of benchmarks executed on BeeGFS services running on an HPE Apollo 4200 (ProLiant XL420 Gen9) server. It also presents the best system configuration identified during the experiment.

2. System Description

The HPE Apollo 4200 server had the following configuration.

- 2x Intel Xeon E5-2620 v3 running at 2.40 GHz
- 128 GB RAM
- 2x 400GiB SSD
- 2x LFF disk array, each with 12x 4TB HDDs
- 2x hardware RAID controller: HP Smart Array P440 and HP Smart Array P840ar, both with firmware version 3.00 and cache of 4 GiB
- Mellanox ConnectX-3 Pro FDR InfiniBand
- Machine BIOS revision 1.50, firmware revision 2.30
- Scientific Linux 6.7, Linux Kernel 2.6.32-573

Four compute nodes were connected to the Apollo server. These had the following configuration:

- Intel Xeon E3-1230 v3 running at 3.30 GHz
- Mellanox ConnectX-3 FDR InfiniBand

BeeGFS release 2015.03-r9 was configured as follows:

- Services on Apollo server: management, metadata, and storage.
- The storage service was configured to use two storage targets: each using a different disk array.
- The metadata service was configured to run on the SSD array.
- The client services were installed on each compute node.

3. Benchmarks

The benchmark tools used in the experiment are listed below:

- **IOR 2.10.3** for measuring the sustained throughput of the BeeGFS storage service.
- **Mdtest 1.9.3** for measuring the performance of the BeeGFS metadata service.

4. Tuning

The following tables provide the values of the system and service tuning options that led BeeGFS to achieve the highest performance during the experiment. For a detailed explanation on their meaning and how they can be set, please go to http://www.beegfs.com/documentation.

4.1 BeeGFS Storage Service Tuning

Formatting Options	Value
RAID stripe size per disk (KB)	256
RAID level	6
Disks per RAID volume	12
Disk array local Linux file system	XFS
Partition Alignment	File system created directly on the block device

XFS Mount Options	Value
Last File and Directory Access	noatime, nodiratime
Log buffer tuning	logbufs=8, logbsize=256k
Streaming performance optimization	largeio, inode64, swalloc
Streaming write throughput	allocsize=131072k
Write Barriers	nobarrier

IO Scheduler Options	Value
Scheduler	deadline
Number of schedulable requests (nr_requests)	4096
Read-ahead data (read_ahead_kb)	8192

Virtual Memory Settings	Value
Kernel dirty (write) background cache flush size (dirty_background_ratio)	5
Kernel dirty (write) cache limit (dirty_ratio)	20
inode caching priority (vfs_cache_pressure)	50
Reserved kernel memory (min_free_kbytes)	262144
Transparent huge pages support (redhat_transparent_hugepage)	never

Controller Settings	Value
Single large operation size (max_sectors_kb)	1024
Max number of requests in the scatter-gather list (sg_tablesize)	543

Concurrency Option	Value
Worker threads (tuneNumWorkers)	48

4.2 BeeGFS Metadata Service Tuning

EXT4 Mount Options	Value
Last File and Directory Access	noatime, nodiratime
Write Barriers	nobarrier

Formatting Options	Value
RAID level	1
SSDs per RAID volume	2
Disk array local Linux file system	ext4
Minimize access times for large directories	-Odir_index
Large inodes	-I 512
Number of inodes	-i 2048
Large journal	-J size=400
Extended attributes	user_xattr
Partition Alignment	File system created directly on the device

IO Scheduler Options	Value
Scheduler	deadline
Number of schedulable requests (nr_requests)	128
Read-ahead data (read_ahead_kb)	128

Concurrency and Parallel Network Requests Options	Value
Worker threads (tuneNumWorkers)	0
Requests in flight to the same server (connMaxInternodeNum)	32

4.3 BeeGFS Client Tuning

Options	Value
Requests in flight to the same server (connMaxInternodeNum)	12
Number of available RDMA buffers (connRDMABufNum)	70
Maximum size of RDMA buffer (connRDMABufSize)	8192
Remote fsync (tuneRemoteFSync)	false

4.4 BeeGFS Striping Settings Tuning

Options	Value
Chunk size (beegfs-ctl pattern option: --chunksize)	512K
Storage targets per file (beegfs-ctl pattern option: --numtargets)	2

5. Results

5.1 Sustained Streaming Throughput

In the storage throughput tests, a total of 320 GiB was written and read in each IOR execution. Each execution started a different number of processes, ranging from 1 to 128, and was repeated 5 times. Charts 1 and 2 and Table 1 show the mean write and read throughput observed in the system.

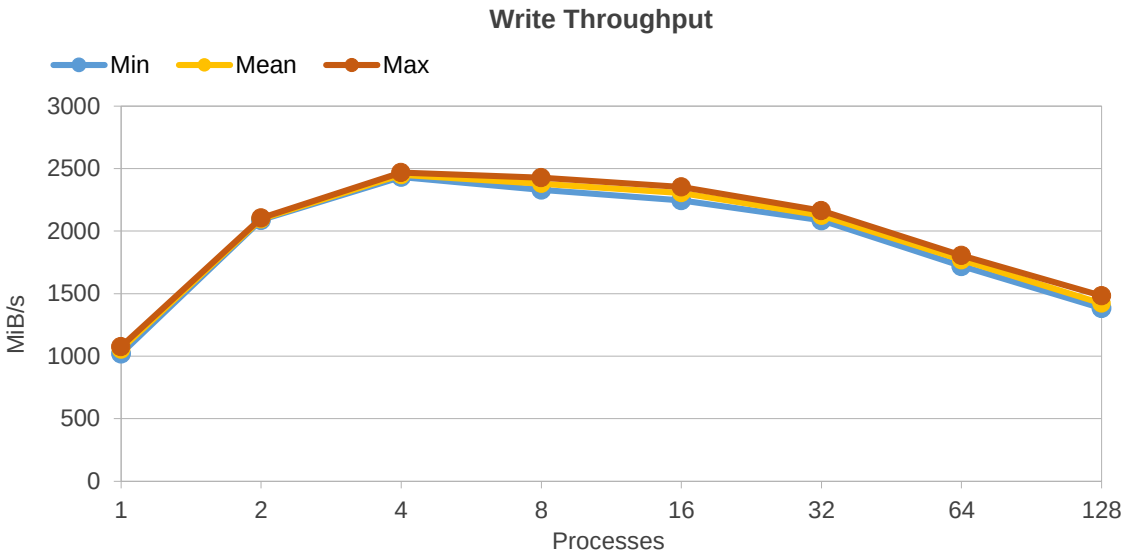

Chart 1 – Write Throughput (320 GiB, 4 Clients, 2 Storage Targets)

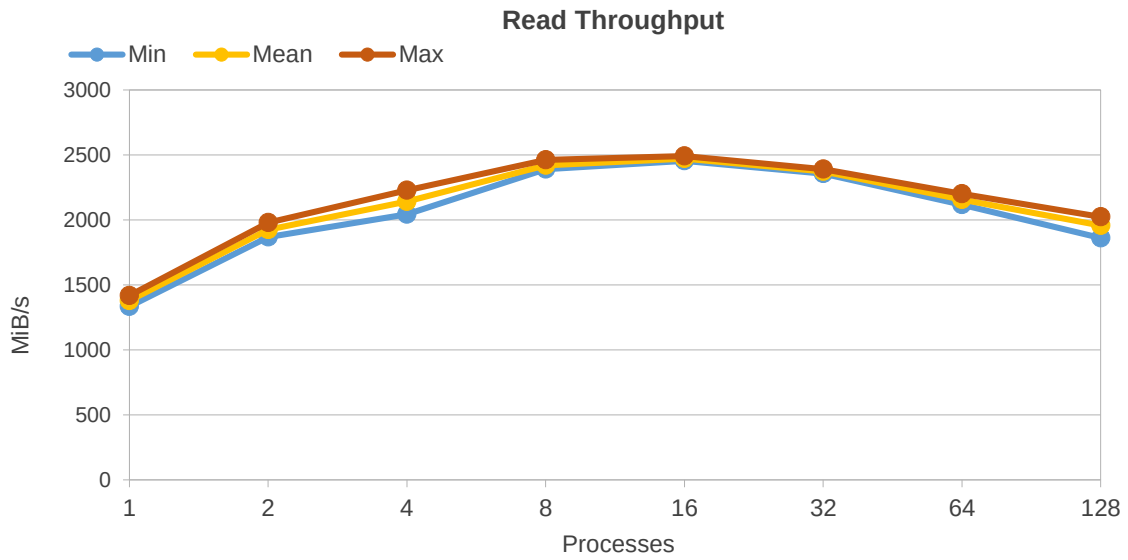

Chart 2 – Read Throughput (320 GiB, 4 Clients, 2 Storage Targets)

	Write			Read		
Processes	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
1	1019,9	1054,31	1074,9	1334,1	1377,4	1417,9
2	2088,0	2096,2	2104,6	1869,0	1925,8	1979,3
4	2432,2	2449,6	2468,4	2043,9	2142,3	2226,8
8	2330,1	2378,7	2427,4	2389,8	2420,6	2462,1
16	2245,0	2305,1	2352,3	2454,7	2472,0	2490,9
32	2086,2	2124,6	2162,8	2354,9	2372,6	2390,1
64	1719,2	1767,7	1805,9	2116,7	2155,7	2201,0
128	1382,9	1421,8	1482,6	1862,2	1957,2	2024,5

Table 1 – Read and Write Throughput Measurements (MiB/s)

The results above show that the system can deliver a (mean) maximum of 2449 MiB/s write and 2472 MiB/s read throughput.

For the streaming tests above, the 2x12 HDDs were connected to two different RAID controller models with 12 HDDs per controller. To compare the performance of the two different controllers, BeeGFS was reconfigured to use only one storage target (12 HDDs) and thus only one controller for each of the following tests.

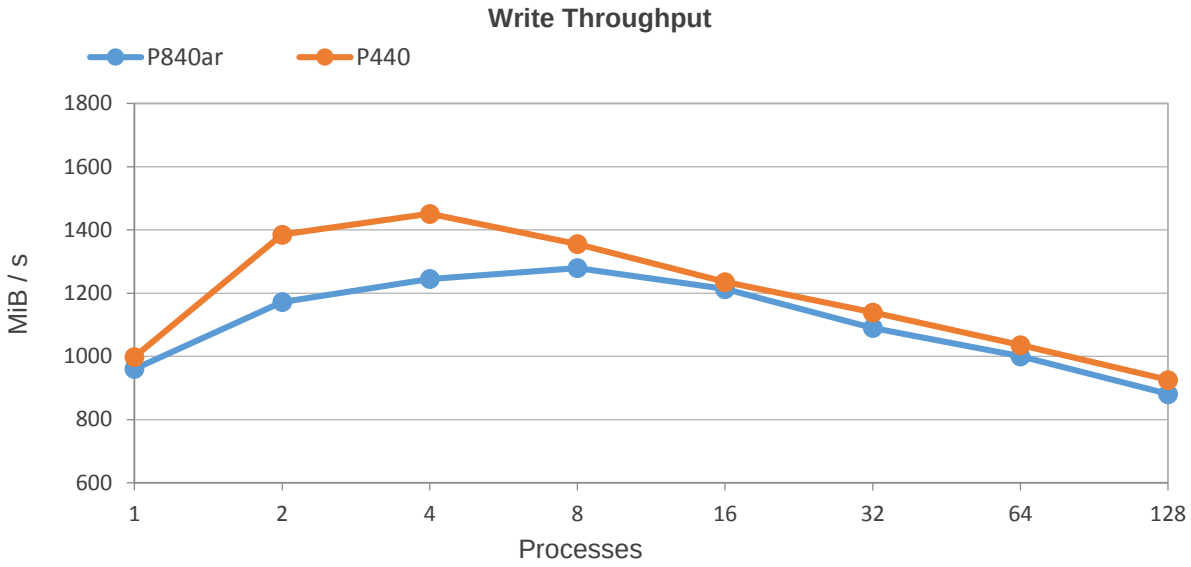

Chart 3 – Write Throughput – Single Disk Array Controller (320 GiB, 4 Clients, 1 Storage Target)

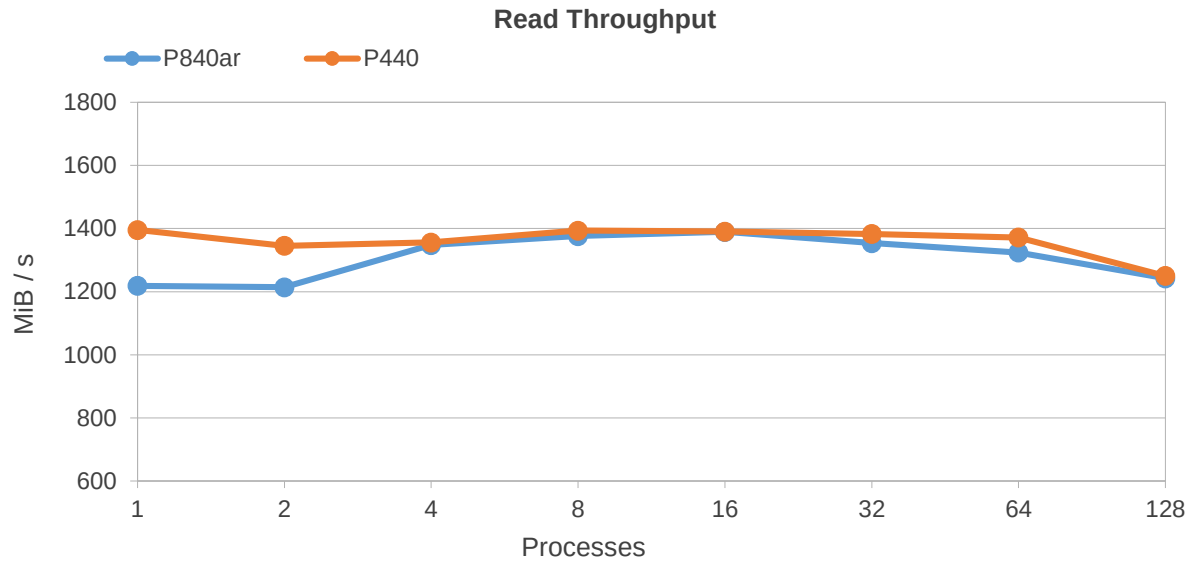

Chart 4 – Read Throughput – Single Disk Array Controller (320 GiB, 4 Clients, 1 Storage Target)

	Write		Read	
Processes	**P840ar**	**P440**	**P840ar**	**P440**
1	960	998	1218	1396
2	1172	1385	1214	1345
4	1245	1451	1348	1356
8	1280	1355	1376	1394
16	1213	1236	1390	1391
32	1090	1139	1354	1383
64	1000	1036	1324	1372
128	881	926	1242	1250

Table 2 – Read and Write Throughput Measurements (MiB/s) – Single Disk Array Controller

As shown by Charts 3 and 4 and Table 2 the controller P440 (orange) had a slightly higher performance than P840ar (blue). The difference is more visible when less processes perform the operations.

The default configuration of the Apollo server includes only the P840ar RAID controller. Thus, a new round of tests was executed with all 24 HDDs attached to the P840ar controller.

Chart 5 and Table 3 show the mean throughput observed in the system. Both read and write throughput were lower in such configuration, compared to the configuration with the additional P440 controller.

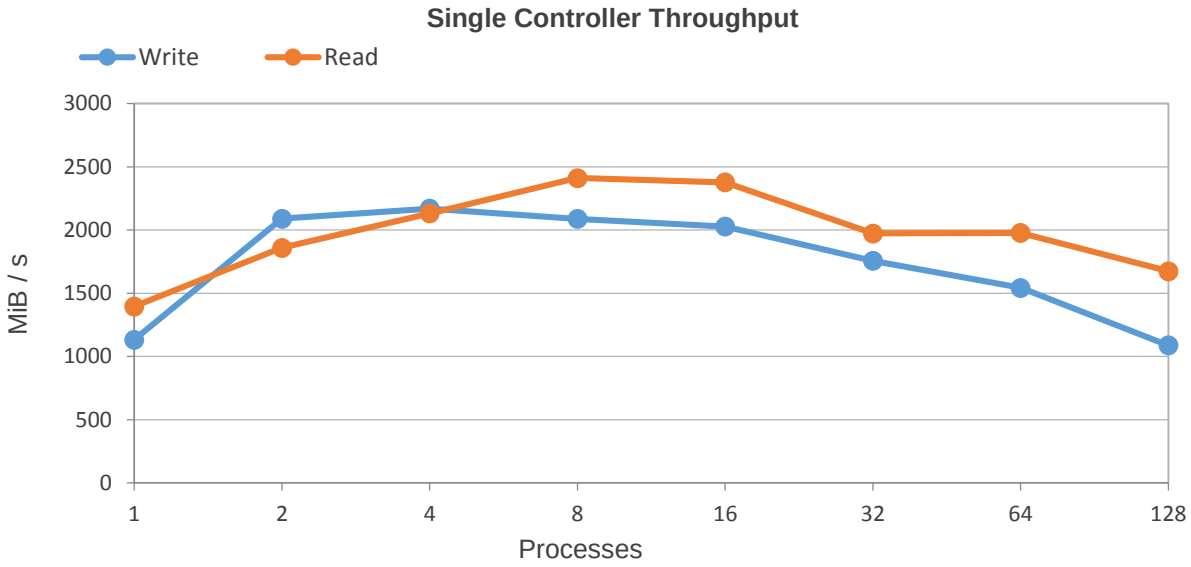

Chart 5 – Read and Write Throughput –Disk Array Controller P840ar (320 GiB, 4 Clients, 2 Storage Targets)

	P840ar	
Processes	**Write**	**Read**
1	1133	1396
2	2090	1860
4	2170	2132
8	2089	2411
16	2028	2377
32	1757	1974
64	1543	1977
128	1089	1675

Table 3- Read and Write Throughput Measurements (MiB/s) – Disk Array Controller P840ar

5.2 Metadata Throughput

Charts 6 and 7 and Table 4 show the mean throughput of the metadata service observed in the system when processing a total of 1,000,000 empty files. Similar to the streaming benchmarks, each execution started between 1 to 128 processes, and was repeated 5 times. For the sake of clarity, only the mean values are plotted on the chart, as the maximum and minimum values are very close.

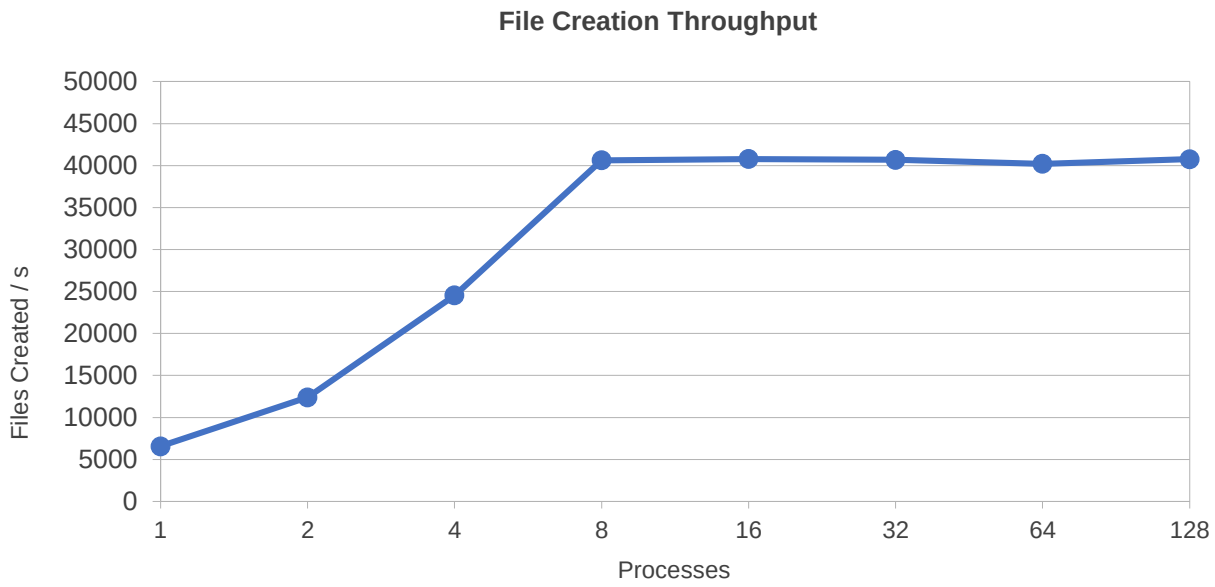

Chart 6 – File Creation Throughput (1,000,000 Files, 4 Clients, 1 Metadata Service)

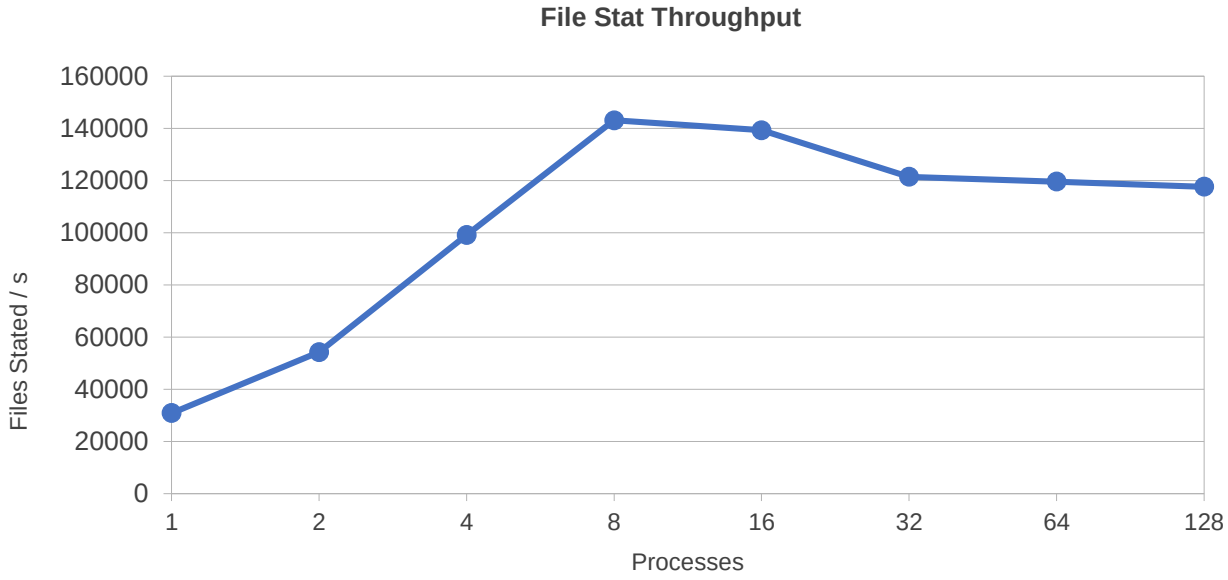

Chart 7 – File Stat Throughput (1,000,000 Files, 4 Clients, 1 Metadata Service)

Processes	File Creation	File Stat
1	6556	30866
2	12394	54246
4	24543	99096
8	40631	143129
16	40785	139289
32	40678	121482
64	40206	119591
128	40757	117679

Table 4 - File Creation and Stat Throughput Measurements (Operations/s)

6. Commands

This section shows the commands that were used on the compute nodes to run the streaming and metadata benchmarks.

6.1 IOR

```
for (( x=0; x <= 7; x++ )); do numprocs=$((2**$x)); echo $numprocs; mpirun -
hostfile /tmp/nodefile -np ${numprocs} /opt/ior/src/ior -wr -i5 -t512k -b
$((320/$numprocs))g -F -g -e -o /scratch/ior/ior-file.dat >>
/opt/ior/results/ior-results.log; done
```

6.2 Mdtest

```
for (( x=0; x <= 7; x++ )); do numprocs=$((2**$x));
filesperdir=$((1000000/64/$numprocs)); echo procs: $numprocs files:
$filesperdir; rm -rf /scratch/mdtest/*; mpirun -hostfile /tmp/nodefile -np
${numprocs}  /opt/mdtest/mdtest -C -T -d /scratch/mdtest -i 5 -I ${filesperdir}
-z 2 -b 8 -L -u -F >> /opt/mdtest/results/mdtest-results.log; done
```