BeeGFS® Benchmarks on IBM OpenPOWER Servers

Ely de Oliveira

October 2016 – 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 IBM OpenPOWER servers. It also presents the best system configuration identified during the experiment.

Thanks to IBM for providing the test hardware and for their support in porting BeeGFS to OpenPOWER.

2. System Description

Four IBM S812LC server machines with the ppc64le architecture were used in this experiment. The Table 1 shows their individual capabilities.

Server	CPU	RAM	RAID Volume	OS HDD	SSD
1	1 POWER8 CPU, 10 Cores	256 GB	10 X 6 TB SATA (RAID 0)	1 TB SATA	960 GB
2	1 POWER8 CPU, 10 Cores	256 GB	10 X 6 TB SATA (RAID 0)	1 TB SATA	960 GB
3	1 POWER8 CPU, 10 Cores	256 GB	10 X 6 TB SATA (RAID 0)	1 TB SATA	-
4	1 POWER8 CPU, 10 Cores	256 GB	10 X 6 TB SATA (RAID 0)	1 TB SATA	-

Table 1 – Servers Computing Capabilities

The reason for the RAID volumes to use RAID 0 is that RAID 6 is not supported by the model of the RAID controllers installed on the machines: Adaptec ASR71605E.

The servers were connected over InfiniBand EDR and 10-Gigabit Ethernet networks, and run Red Hat Enterprise Linux 7 for ppc64le, Linux Kernel 3.10.0-327.

BeeGFS release 2015.03-r17 was built directly on the machines. The compiler used was GCC 4.9, as tests showed that using the compiler IBM XL C/C++ 13.1 did not deliver better results.

The BeeGFS services were installed as shown in Table 2. Both the management and metadata services were configured to use the SSD devices. The storage services were configured to use the RAID volumes. On each server, the operating system was installed on a separate HDD.

Server	BeeGFS Services
1	Client, Storage, Metadata, Management
2	Client, Storage, Metadata
3	Client, Storage
4	Client, Storage

Table 2 – BeeGFS Services per Server

3. Benchmark Tools

The benchmark tools used in the experiment are listed below.

- **IOZone 3.465** for measuring the sustained data streaming throughput of the BeeGFS storage service.
- **Mdtest 1.9.3** for measuring the throughput of the BeeGFS metadata service.

4. Results

4.1 Sustained Data Streaming Throughput

In the data streaming throughput tests, a total of 2,560 GiB was written and read in each IOZone execution. Each execution started a different number of processes, ranging from 4 to 64, and was repeated 3 times, using a record size of 1 MB. Other record sizes have been tested, but they didn't have any noticeable performance impact.

Chart 1 and Table 3 show the mean write and read throughput observed in the system. For the sake of clarity, only the mean values are plotted on the chart, as the maximum and minimum values are very close.

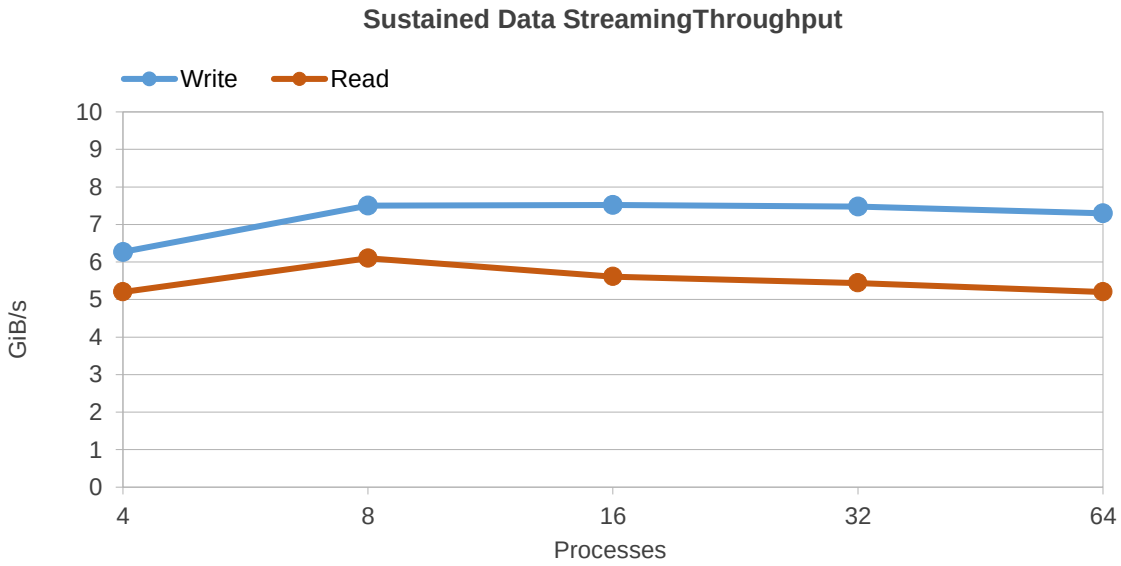

Chart 1 – Sustained Data Streaming Throughput (2,560 GiB, 4 Clients, 4 Storage Targets)

Processes	Write	Read
4	6.27	5.20
8	7.50	6.10
16	7.52	5.61
32	7.48	5.44
64	7.30	5.20

Table 3 – Sustained Data Streaming Throughput Measurements (GiB/s)

The results above show that the system can deliver a maximum (mean) throughput of 6.10 GiB/s for read operations and 7.52 GiB/s for write operations. They also show that the system scales well when more client processes are started, keeping the data transfer rates around 7.2 GiB/s (write) and 5.5 GiB/s (read).

4.2 Metadata Throughput

Charts 2 and 3 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 data streaming benchmarks, each execution started between 1 to 64 processes, and was repeated 3 times. For the sake of clarity, only the mean values are plotted on the chart, as the maximum and minimum values are very close.

The results show that the system can deliver a maximum (mean) throughput of 54,307 file creation operations per second and 185,325 file stat operations per second.

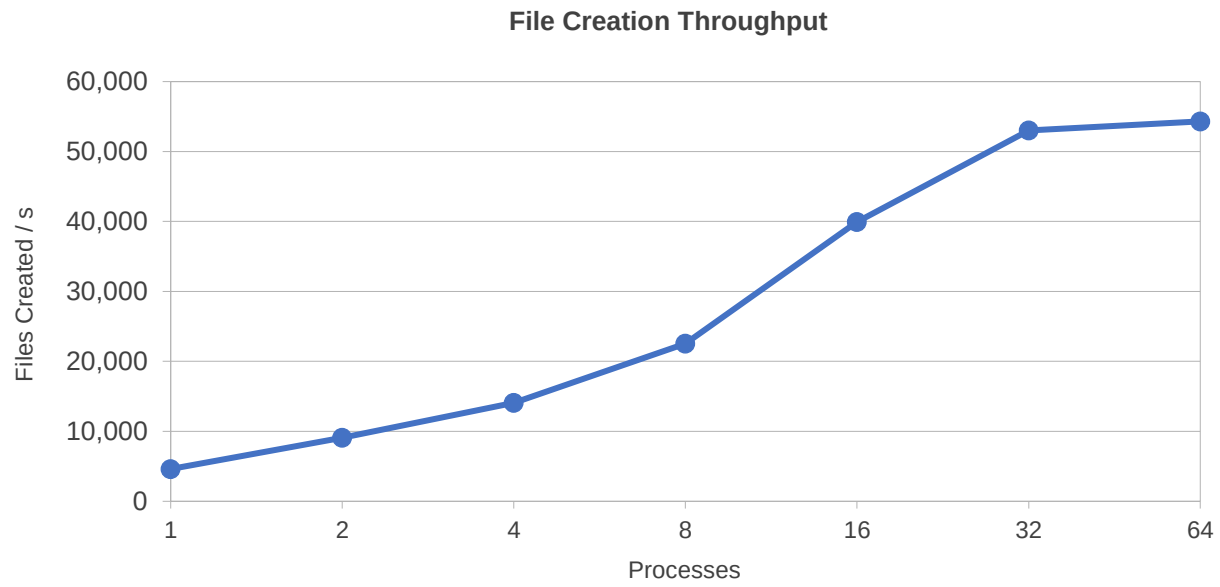

Chart 2 – File Creation Throughput (1,000,000 Files, 4 Clients, 2 Metadata Services)

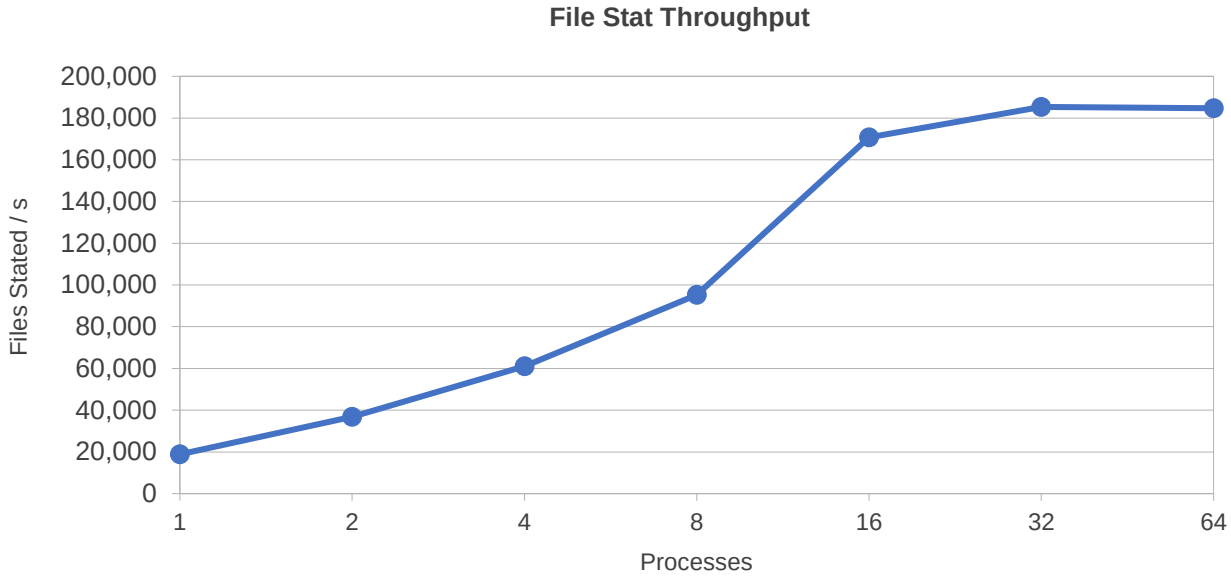

Chart 3 – File Stat Throughput (1,000,000 Files, 4 Clients, 2 Metadata Services)

Processes	File Creation	File Stat
1	4,617	18,861
2	9,081	36,757
4	14,060	61,042
8	22,550	95,205
16	39,914	170,681
32	53,024	185,325
64	54,307	184,710

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

5. 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.

5.1 BeeGFS Storage Service Tuning

Formatting Options	Value
RAID stripe size per disk (KB)	256
RAID level	0
Disks per RAID volume	10
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

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

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)	256
Performance Mode (Arcconf SETPERFORM)	1 (Dynamic/Default Performance mode)

BeeGFS Storage Service Configuration Options	Value
Worker threads (tuneNumWorkers)	48
Preferred Network Interfaces (connInterfacesList)	Ib0 (InfiniBand) enP1 (Ethernet)
File read-ahead size (tuneFileReadAheadSize)	32m
File read-ahead trigger size (tuneFileReadAheadTriggerSize)	4m
File Read Size (tuneFileReadSize)	128k
File Write Size (tuneFileWriteSize)	128k

5.2 BeeGFS Metadata Service Tuning

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

Formatting Options	Value
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)	4096
Read-ahead data (read_ahead_kb)	8192

BeeGFS Metadata Service Configuration Options	Value
Worker threads (tuneNumWorkers)	80
Requests in flight to the same server (connMaxInternodeNum)	64
Target Chooser Algorithm (tuneTargetChooser)	randomrobin (recommended for tests only)
Preferred Network Interfaces (connInterfacesList)	Ib0 (InfiniBand) enP1 (Ethernet)

5.3 BeeGFS Client Tuning

BeeGFS Client Service Configuration Options	Value
Requests in flight to the same server (connMaxInternodeNum)	32
Number of available RDMA buffers (connRDMABufNum)	70
Maximum size of RDMA buffer (connRDMABufSize)	8192
Remote fsync (tuneRemoteFSync)	True
Preferred Network Interfaces (connInterfacesList)	Ib0 (InfiniBand) enP1 (Ethernet)
Preferred Metadata (tunePreferredMetaFile)	Path to a file containing the node ID of the metadata service running on the same machine

5.4 BeeGFS Striping Settings Tuning

Options	Value
Chunk size (beegfs-ctl pattern option: --chunksize)	512K
Storage targets per file (beegfs-ctl pattern option: --numtargets)	4

6. Commands

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

6.1 IOZone

```
#!/bin/bash
work_dir=/mnt/beegfs/iozone
iozone_dir=/opt/iozone3_465
nodes_file=${iozone_dir}/nodes-list
results_file=${iozone_dir}/results.log

cd ~
rm -rf ${work_dir}
mkdir ${work_dir}
cd ${work_dir}

for (( x=2; x <= 6; x++ )); do

    num_procs=$((2**${x}));
    header="processes: ${num_procs}"

    echo ${header}
    printf "\n\n\n${header}\n\n" >> ${results_file}

    ${iozone_dir}/src/current/iozone -i 0 -i 1 -c -e -w -r 1m -t ${num_procs} \
        -s $((2560/${num_procs}))g -+n -+m ${nodes_file} >> ${results_file}
done
```

6.2 Mdtest

```
#!/bin/bash
mdtest_dir=/opt/mdtest-1.9.3
work_dir=/mnt/beegfs/mdtest
nodes_file=/tmp/nodes-list
results_file=${mdtest_dir}/results.log

cd ~
rm -rf ${work_dir}
mkdir ${work_dir}

for (( x=0; x <= 6; x++ )); do

  num_procs=$((2**${x}))
  files_per_dir=$((1000000/64/${num_procs}))
  header="processes: ${num_procs}, files per directory: ${files_per_dir}"

  echo ${header}
  printf "\n\n\n${header}\n\n" >> ${results_file}

  rm -rf ${work_dir}/*
  mpirun --mca btl self,tcp --map-by node -hostfile ${nodes_file} \
      -np ${num_procs} ${mdtest_dir}/mdtest -C -T -d ${work_dir} -i 5 \
      -I ${files_per_dir} -z 2 -b 8 -L -u -F -r >> ${results_file}
done
```