BeeGFS

Parsa Mohammadian
Arshia Akhavan

July 16, 2023

Contents

1 Introduction

BeeGFS was developed at the Fraunhofer Institute for industrial mathematics (ITWM). It was originally released as "FhGFS", but was newly labeled as BeeGFS (registered trade mark) in 2014. The formerly known FhGFS can still be found in some scripts or code. [1]

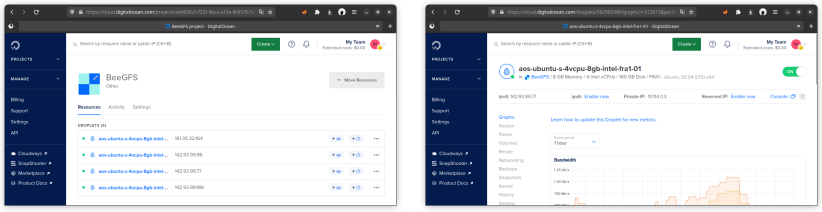

Figure 1: Overall system design

The decision to develop a new parallel file system was made in 2005 – and it was launched as a research project. The original motivation was born out of experiencing that available products neither scale the way users expect it – nor were they as easy to use and maintain as system administrators demand it. [1]

The first productive installation within the Fraunhofer Institute was made in 2007. It became very soon clear that the file system quickly evolved to a state which made it useful for HPC users. The first performing installation outside Fraunhofer followed in 2009. [1]

Since then the file system became increasingly popular in Europe and the US for being used in small, medium and large HPC systems. According to the number of installations BeeGFS is very much accepted and appreciated in academia – but also commercial customers do trust the product, too. [1]

2 Background

3 Related Work

4 System

4.1 Overview

4.2 Design

Our system design consists of 4 nodes, each with 4 Core of CPU and 8 Giga Bytes of RAM as shown in Figure 1. The CPU used in this Experiment is Intel's Xeon Bronze 3106 Processor. Each node has 150 Giga Bytes of SSD mounted as /dev/sda. Ubuntu 20.04 is installed on all of the nodes.

The file system is deployed on two nodes. management and metadata server are both deployed on the same node. Each node consist of one storage server (total of two storage servers each with 150GB capacity).

The other two nodes are used as BeeGFS clients and are configured for benchmarking.

4.3 Implementation

BeeGFS Server are installed using docker containers provided by BeeGFS itself. The directories for storage servers are mounted to bypass OverlayFS overhead for IO intensive workloads. BeeGFS clients are installed and executed using systemd. A BeeGFS kernel module is installed to further optimized the use of BeeGFS mounted filesystem.

both BeeGFS server and client are deployed using Ansible and can be seen here We developed an Ansible-galaxy role for installing BeeGFS servers using Docker and extended an already provided ansible-galaxy role for deploying BeeGFS clients.

5 Evaluation

In order to evaluate the performance of BeeGFS, we have used the IOZone benchmark. IOZone is a filesystem benchmark tool. The benchmark generates and measures 13 types of file operations which are listed below: [2]

- Read - Indicates the performance of reading a file that already exists in the filesystem.
- Write - Indicates the performance of writing a new file to the filesystem.
- Re-read - After reading a file, this indicates the performance of reading a file again.
- Re-write - Indicates the performance of writing to an existing file.
- Random Read - Indicates the performance of reading a file by reading random information from the file. i.e this is not a sequential read.
- Random Write - Indicates the performance of writing to a file in various random locations. i.e this is not a sequential write.
- Backward Read
- Record Re-Write
- Stride Read
- Fread
- Fwrite
- Freread
- Frewrite

The IOZone benchmark script can be found in the `benchmark/IOZone.sh` [1] file. The script is written in bash and it takes the following arguments:

[1] `https://github.com/Parsa2820/BeeGFS-Benchmark/blob/master/benchmark/IOZone.sh`

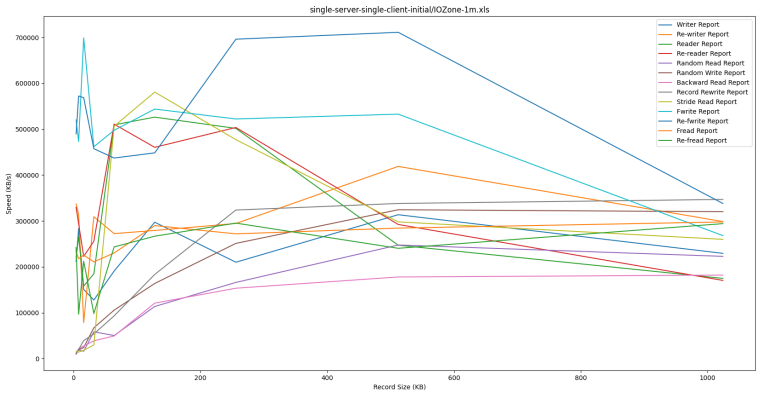

Figure 2: IOZone benchmark results on a single BeeGFS client node with a single node BeeGFS cluster mounted on it for 1MB file

- Directory in which to run IOZone (should be a BeeGFS mount point)
- Output log file name
- File size (in IOZone recognized units)

The script runs IOZone with the passed arguments and saves the standard output in the specified log file. Another file is created within the specified directory which contains benchmark results in Excel format. The latter file then is parsed by the `result/excel-to-chart.ipynb` [2] Jupyter notebook and the results are plotted in a chart. The charts are accessible both in the notebook and in the `result/excel-to-chart` [3] directory.

The initial benchmark was run on a single BeeGFS client node which had a single node BeeGFS cluster mounted on it. The benchmark was run on 1MB, 10MB, 100MB, 1GB, 10GB files. The results are plotted in Figures 2, 3, 4, 5, 6 respectively. The results show that the performance of BeeGFS is not very good for large files. The performance of BeeGFS decreases as the file size increases. However, the performance is more consistent in different operations for larger files. These results will be used as a baseline for the rest of the benchmarks and optimizations.

[2] `https://github.com/Parsa2820/BeeGFS-Benchmark/blob/master/result/excel-to-chart.ipynb`

[3] `https://github.com/Parsa2820/BeeGFS-Benchmark/tree/master/result/excel-to-chart`

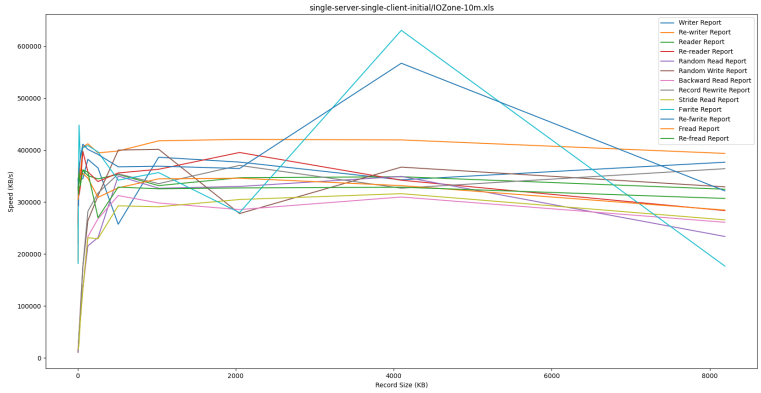

Figure 3: IOZone benchmark results on a single BeeGFS client node with a single node BeeGFS cluster mounted on it for 10MB file

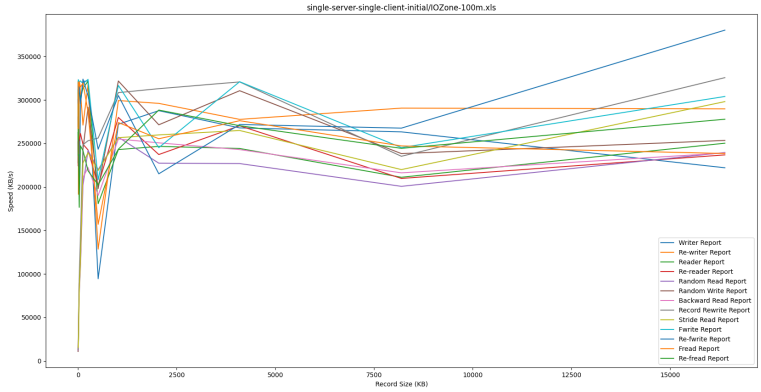

Figure 4: IOZone benchmark results on a single BeeGFS client node with a single node BeeGFS cluster mounted on it for 100MB file

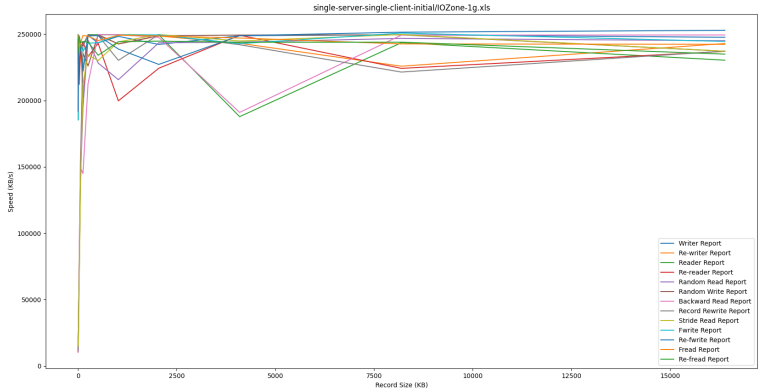

Figure 5: IOZone benchmark results on a single BeeGFS client node with a single node BeeGFS cluster mounted on it for 1GB file

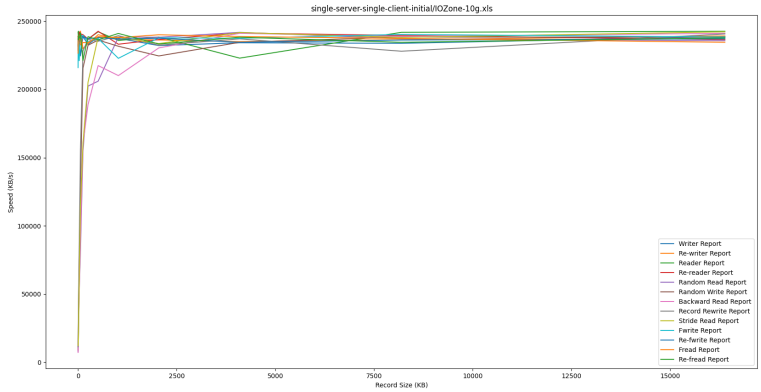

Figure 6: IOZone benchmark results on a single BeeGFS client node with a single node BeeGFS cluster mounted on it for 10GB file

6 Optimization and Tuning

6.1 Storage Tuning

There are two sets of tuning done on the storage servers, one regarding memory and one regarding the storage disk.
Regarding the servers memory we did the following tunings:

- To avoid long IO stalls (latency) for write cache flushing we limited the kernel dirty write cache size
- We also gave higher priority to inode caching to avoid disk seeks for inode loading
- By Raising the amount of reserved kernel memory we achieved faster and more reliable memory allocation (Required for buffering of file system)
- We also Enabled transparent hugepage for storage servers

Regarding the servers disks, we did the following tunings:

- We first started by using Deadline Scheduler.
- We also increased request size of io queue for storage disks.
- Given our experiments, we had lots of sequential reads workloads, so we also had to tune disks for sequential reads.
- By increasing Max chunk size of each io request, we were able to reduce total amount of IO operations

The script used for doing such a tunings can be found in `tuning/storage-tuning.sh` [4] file.

The result of the tuning for 1MB, 10MB, 100MB, 1GB, 10GB files can be seen in Figures 7, 8, 9, 10, 11 respectively.

As we can see in the results, the performance of BeeGFS has improved significantly after the storage tuning. Especially for larger files.

6.2 Data Stripping

In order to Improve the overall performance of the system, BeeGFS recommends the usage of stripping in file system. We conducted an experiment to measure the effectiveness of stripping on IO performance. BeeGFS provides a per directory stripping configuration. We created 4 directories, named s1 to s4, each with stripping level of 1 to 4 receptively.

The results of the experiment in the s1 directory for 1MB, 10MB, 100MB files can be seen in Figures 12, 13, 14 respectively. The results of the experiment

[4] `https://github.com/Parsa2820/BeeGFS-Benchmark/blob/master/tuning/storage-tuning.sh`

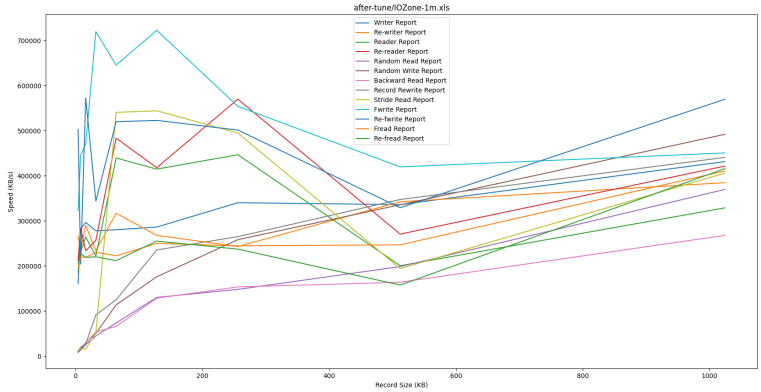

Figure 7: IOZone benchmark results for 1MB file after storage tuning

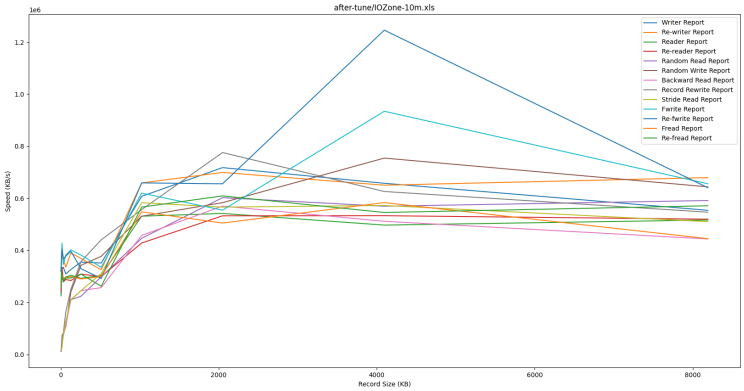

Figure 8: IOZone benchmark results for 10MB file after storage tuning

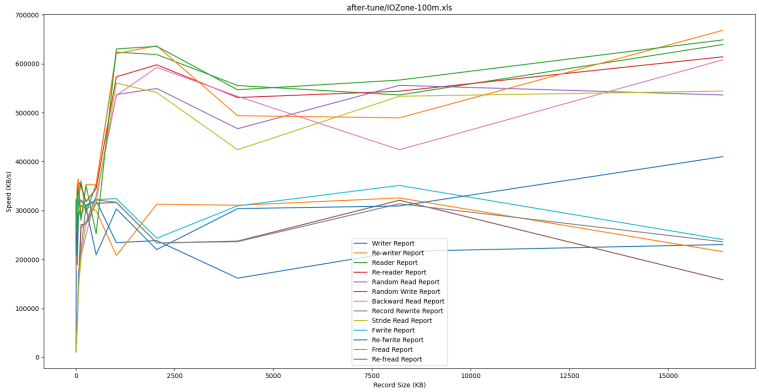

Figure 9: IOZone benchmark results for 100MB file after storage tuning

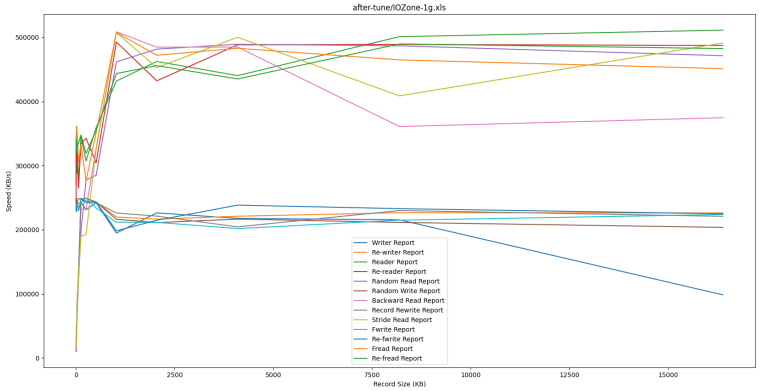

Figure 10: IOZone benchmark results for 1GB file after storage tuning

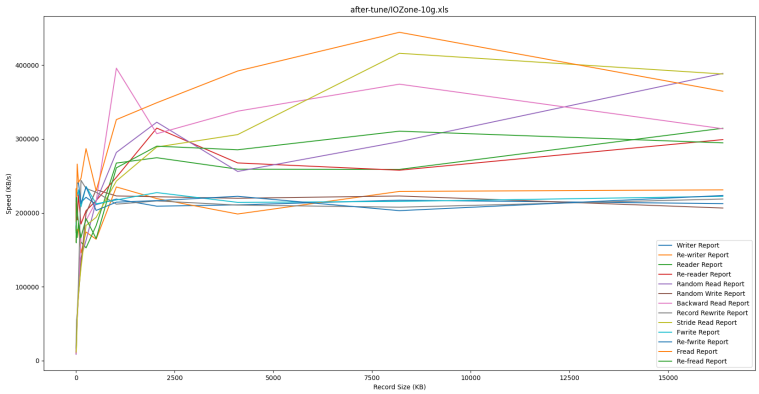

Figure 11: IOZone benchmark results for 10GB file after storage tuning

in the s2 directory for 1MB, 10MB, 100MB files can be seen in Figures 15, 16, 17 respectively. The results of the experiment in the s3 directory for 1MB, 10MB, 100MB files can be seen in Figures 18, 19, 20 respectively. The results of the experiment in the s4 directory for 1MB, 10MB, 100MB files can be seen in Figures 21, 22, 23 respectively.

As we can see in the results, even levels of stripping have absolutely better performance than odd levels of stripping. The size of the files has its effect on the performance, but the previous statement is true for all file sizes.

7 Utilize BeeGFS Cache

BeeGFS provides a mechanism for client-side caching. at the time of this research, BeeGFS provides support for two different caching scheme, buffered caching and native caching. Buffered caching provides a mechanism that results in higher streaming throughput compared to native caching, but it is limited by the cache size which is only few hundreds of kilo bytes. As for native caching scheme, BeeGFS client uses Linux's native page cache mechanism which can scale proportional to the size of the node's memory. In our experiment we used both scheme and conducted an experiment to evaluate the effects of BeeGFS client side caching on IO performance.

7.1 Buffered Caching Results

Bufferd caching results for 1MB, 10MB, 100MB, 1GB files can be seen in Figures 24, 25, 26, 27 respectively.

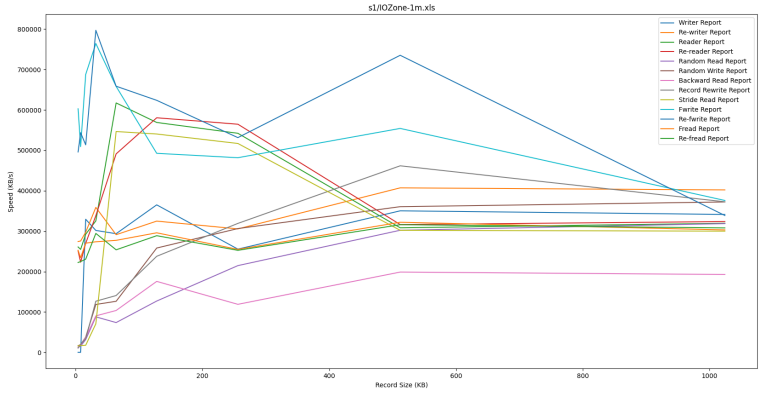

Figure 12: IOZone benchmark results for 1MB file in s1 directory

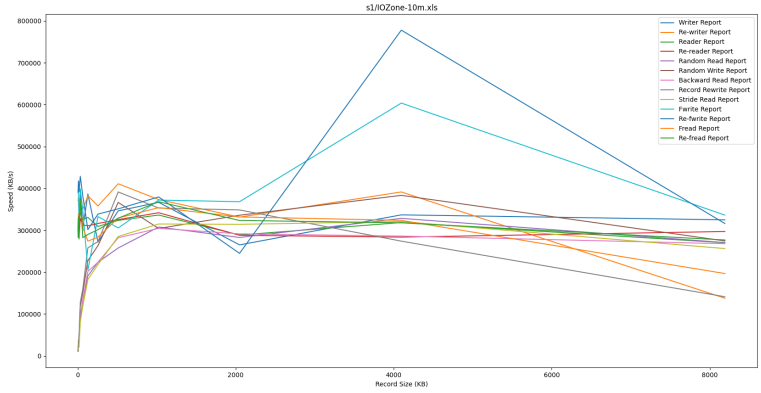

Figure 13: IOZone benchmark results for 10MB file in s1 directory

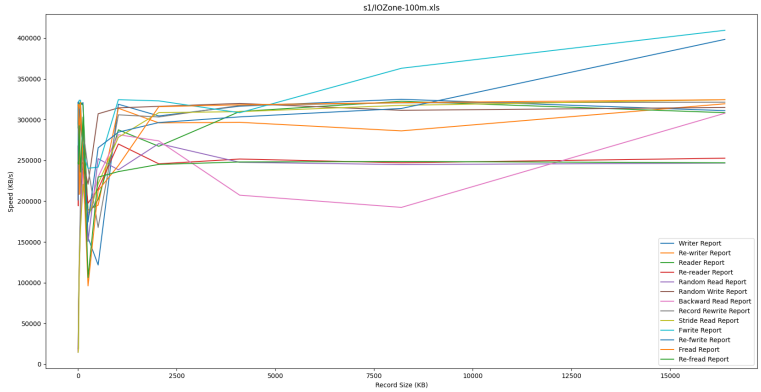

Figure 14: IOZone benchmark results for 100MB file in s1 directory

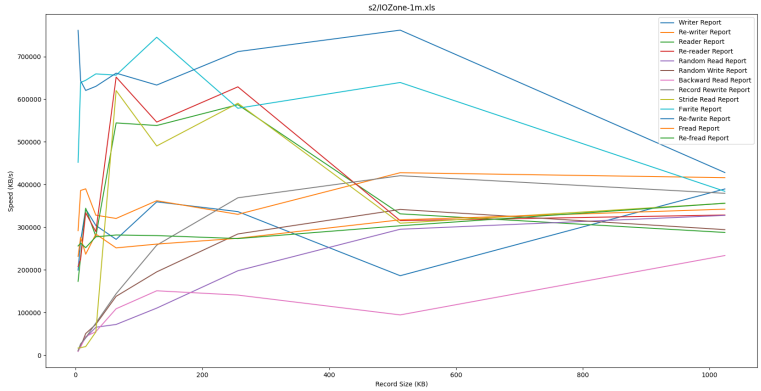

Figure 15: IOZone benchmark results for 1MB file in s2 directory

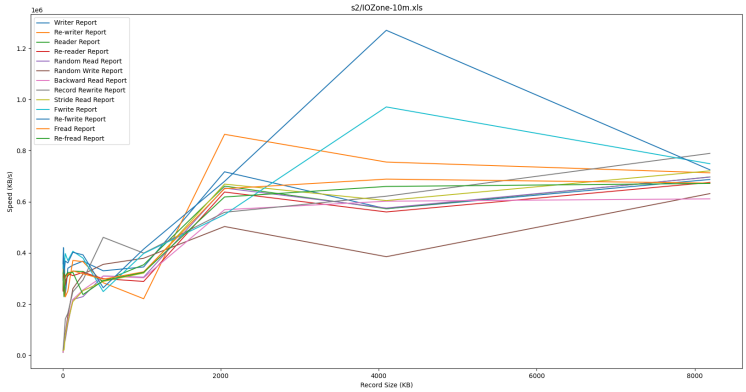

Figure 16: IOZone benchmark results for 10MB file in s2 directory

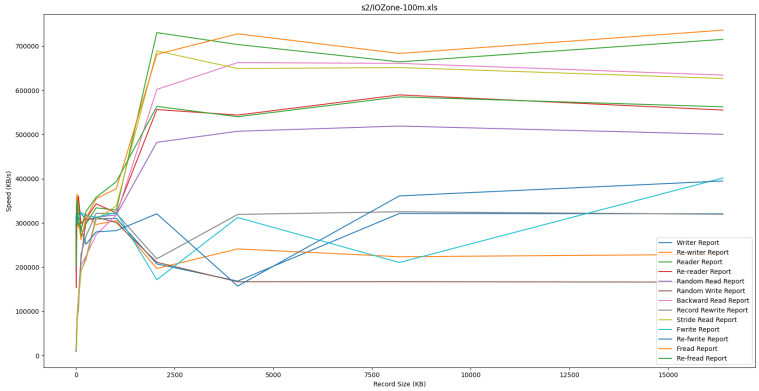

Figure 17: IOZone benchmark results for 100MB file in s2 directory

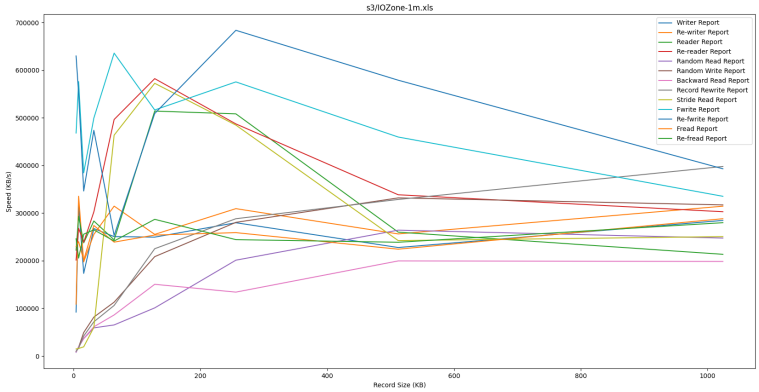

Figure 18: IOZone benchmark results for 1MB file in s3 directory

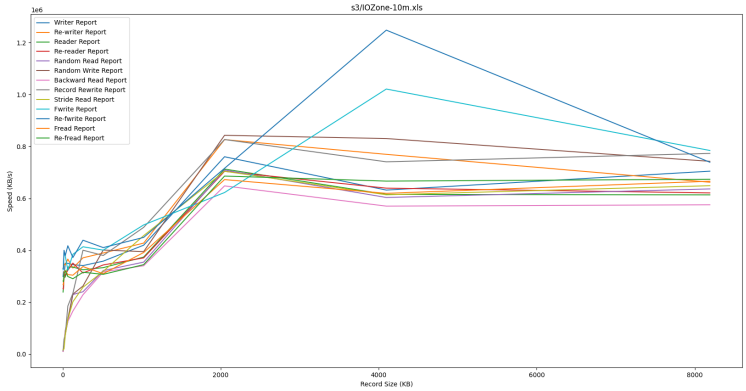

Figure 19: IOZone benchmark results for 10MB file in s3 directory

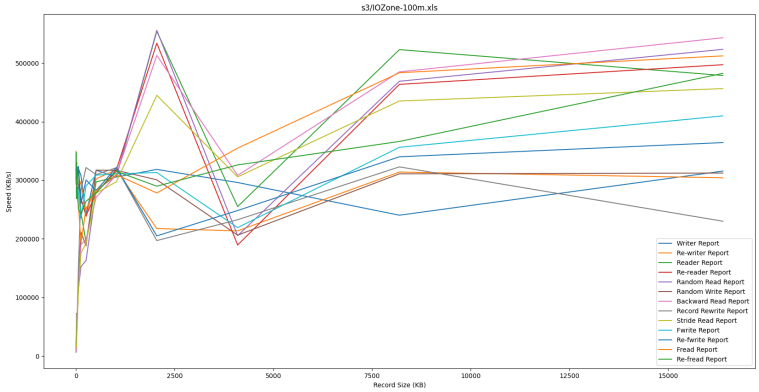

Figure 20: IOZone benchmark results for 100MB file in s3 directory

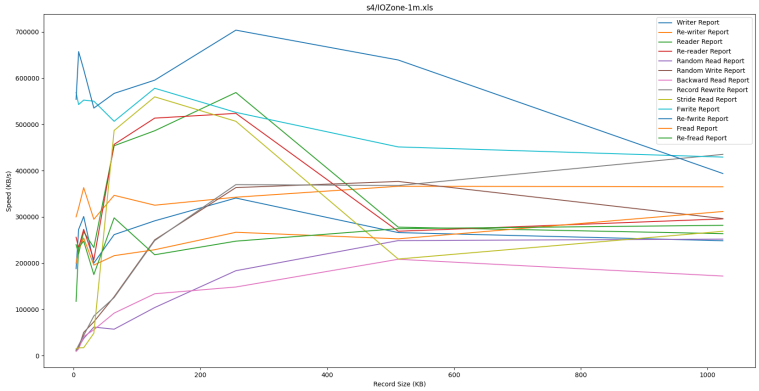

Figure 21: IOZone benchmark results for 1MB file in s4 directory

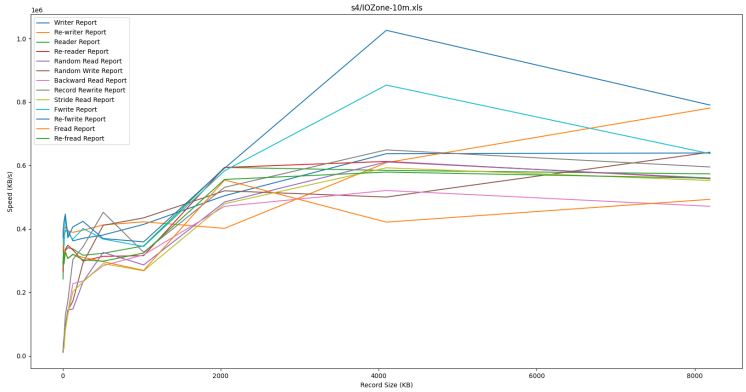

Figure 22: IOZone benchmark results for 10MB file in s4 directory

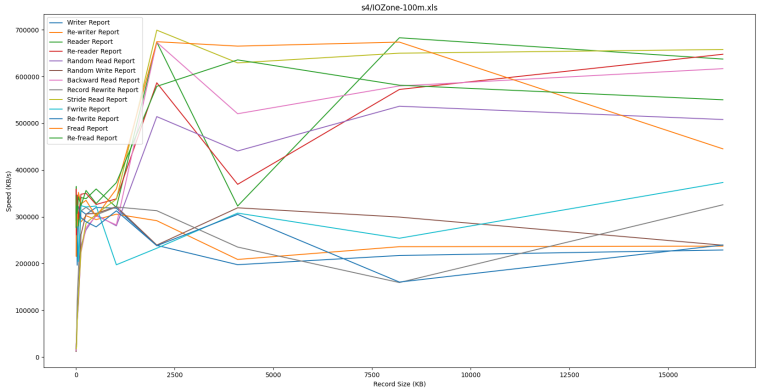

Figure 23: IOZone benchmark results for 100MB file in s4 directory

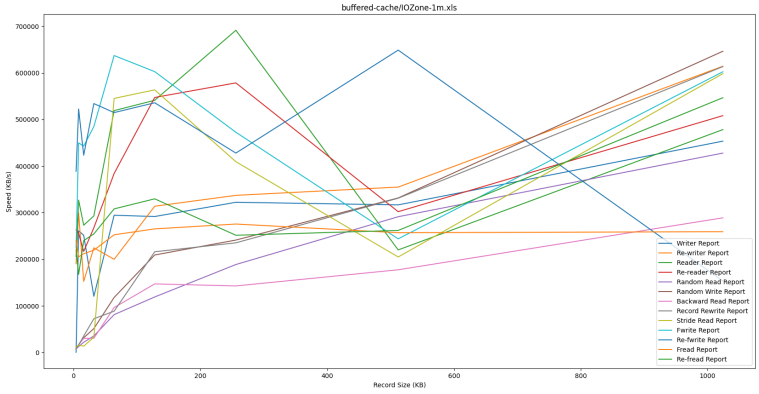

Figure 24: IOZone benchmark results for 1MB file with buffered caching

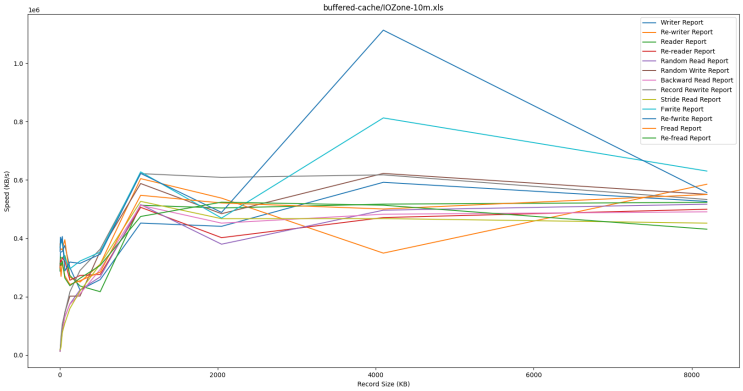

Figure 25: IOZone benchmark results for 10MB file with buffered caching

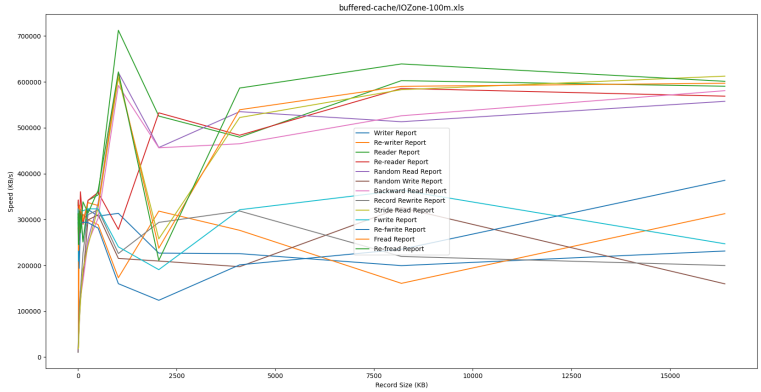

Figure 26: IOZone benchmark results for 100MB file with buffered caching

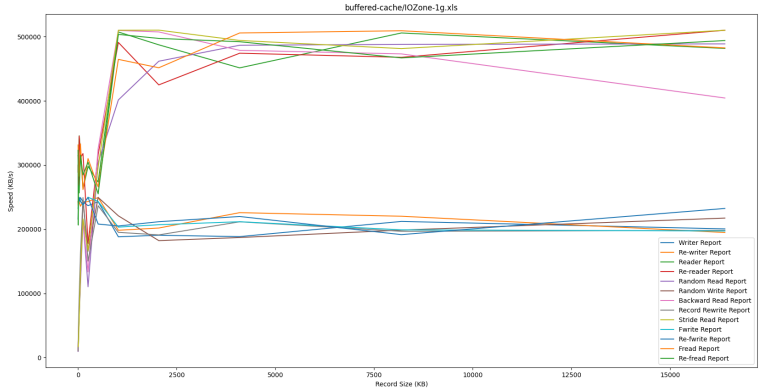

Figure 27: IOZone benchmark results for 1GB file with buffered caching

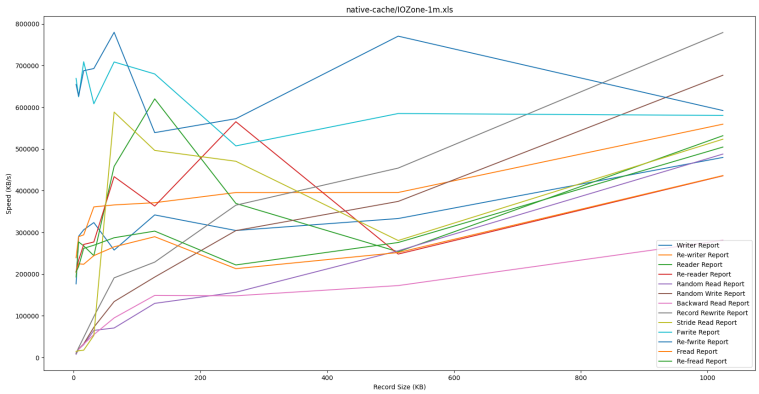

Figure 28: IOZone benchmark results for 1MB file with native caching

7.2 Native Caching Results

Native caching results for 1MB, 10MB, 100MB, 1GB files can be seen in Figures 28, 29, 30, 31 respectively.

7.3 Comparison

As we can see in the results, both caching schemes have improved the performance of BeeGFS. In larger files the performance boost is more significant. Sometimes even there is performance degradation in smaller files due to the overhead of caching. Overally buffered caching seems to have slightly better performance than native caching.

8 Utilize OpenCAS Cache

OpenCAS is a caching solution for Linux. It provides a block device which can be used as a cache for other block devices. Since BeeGFS is a file system, we used OpenCAS to cache the storage devices of the BeeGFS servers. The script used for installing and configuring OpenCAS can be found in `tuning/open-cas.sh` [5] file.

The results of the experiment for 1MB, 10MB, 100MB, 1GB files can be seen in Figures **??**, **??**, **??**, **??** respectively.

9 Conclusion

[5] `https://github.com/Parsa2820/BeeGFS-Benchmark/blob/master/tuning/open-cas.sh`

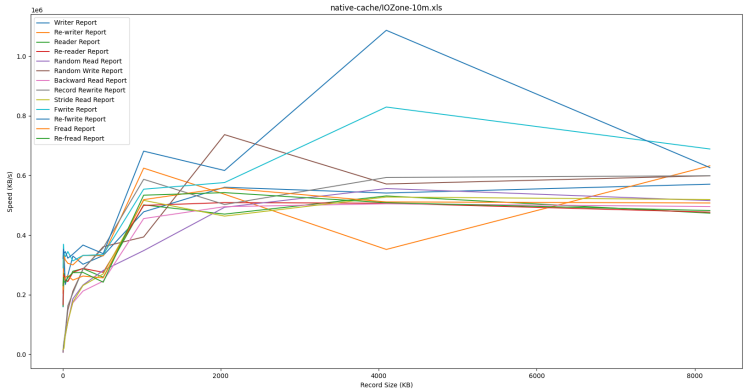

Figure 29: IOZone benchmark results for 10MB file with native caching

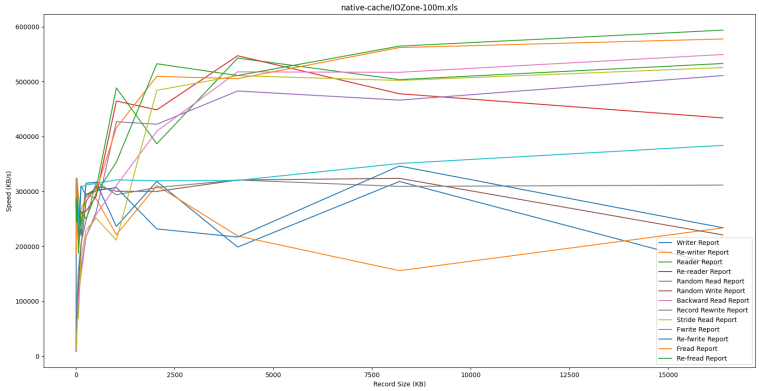

Figure 30: IOZone benchmark results for 100MB file with native caching

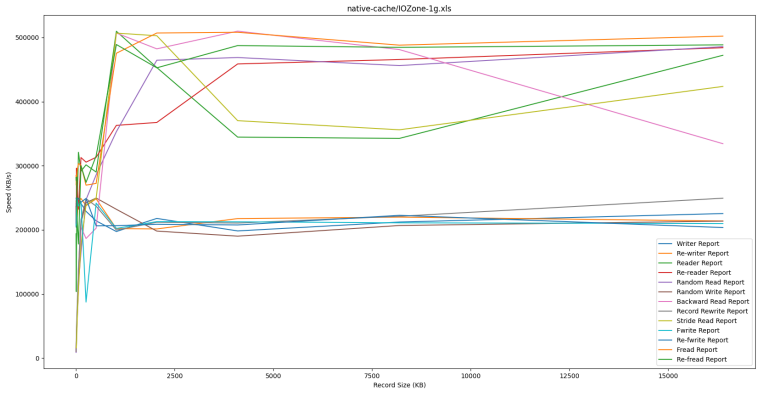

Figure 31: IOZone benchmark results for 1GB file with native caching

References

[1] Jan Heichler. “An introduction to BeeGFS”. In: *Introductionto BeeGFS by ThinkParQ. pdf* (2014).

[2] Ramesh Natarajan. *10 iozone Examples for Disk I/O Performance Measurement on Linux*. 2011. URL: `https://www.thegeekstuff.com/2011/05/iozone-examples/` (visited on 05/02/2011).