2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops) | 979-8-3503-7062-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/CLUSTERWorkshops61457.2023.00010

Does Varying BeeGFS Configuration Affect the I/O Performance of HPC Workloads?

Arnav Borkar†, Joel Tony†, Hari Vamsi K. N, Tushar Barman, Yash Bhisikar, Sreenath T. M., Arnab K. Paul
BITS Pilani, K. K. Birla Goa Campus
{*f20201384, f20212077, h20220031, h20220030, f20210483, h20220018, arnabp*}*@goa.bits-pilani.ac.in*

***Abstract*—The increasing gap between processor and disk performance leads to high performance computing (HPC) applications facing I/O bottlenecks. This makes parallel file systems one of the most important components in an HPC cluster. This work analyzes the I/O performance of different workloads for various BeeGFS configurations. Our analysis shows that the default allocation strategies and striping configuration leads to an imbalanced distribution of workload data, thereby negatively affecting the I/O performance.**

***Keywords*-BeeGFS, High Performance Computing, Parallel File System**

I. Introduction

High Performance Computing (HPC) applications are becoming increasingly data intensive. Over the years, the processor performance of compute nodes has grown at a faster rate than the I/O performance, making I/O operations a limiting factor in application efficiency [1]. The Parallel File System (PFS) is a critical component of the I/O subsystem, enabling multiple clients to store and access data spread across multiple storage nodes through parallel I/O paths. BeeGFS, a POSIX-compatible PFS, is widely favored for its ease of installation and management.

BeeGFS services are distributed across multiple Object Storage Servers (OSS), each containing several Object Storage Targets (OST). When a client creates a file, it is divided among a group of OSTs, with each partition called a "chunk" and the number of OSTs involved is known as the "stripe count". Chowdhury et al. [2] examined the impact of emerging deep-learning workloads on BeeGFS, with a focus on chunk size and target count. While various studies [3], [4] have explored the effects of different PFS parameters on diverse I/O workloads, there has not been a comprehensive investigation into other parameters in BeeGFS, such as OST allocation strategy and striping.

This paper investigates the impact of altering the striping configuration and OST allocation strategy of BeeGFS on I/O performance. Specifically, our work tries to answer the following research questions.

- *RQ1*: How does the OST allocation strategy affect load imbalance across OSSs and OSTs?
- *RQ2*: How does varying chunk size and number of targets affect I/O performance?

† Made equal contribution to this work.

II. Background

1) BeeGFS: The architecture of BeeGFS has server components running in user space, and the client is a patchless kernel module [5]. Metadata Servers (MDS) manage file metadata. Object Storage Servers (OSS) store data and retrieve it from Object Storage Targets (OST). The Management Service oversees the PFS. BeeGFS allows tailoring the *chunk size* and *stripe count* for each directory and file, allowing users to optimize performance based on specific workload requirements.

2) IOR: IOR is a parallel IO benchmark that can be used to test the performance of parallel storage systems [6]. It allows for testing using various interfaces and access patterns by coordinating processes across nodes using MPI.

3) DLIO: The DLIO benchmark [7] suite emulates the I/O behavior of deep learning (DL) applications. DL training is often read-intensive due to multiple training epochs over a large dataset. DLIO, therefore is an important benchmark to profile the I/O behavior of emerging workloads.

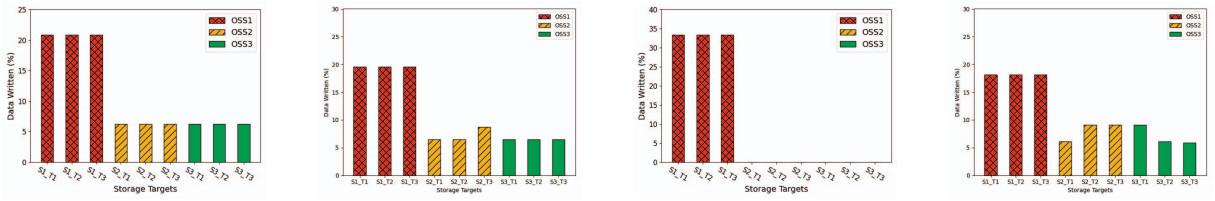
4) HACC-IO: The HACC-IO benchmark [8] captures the I/O patterns of the Hardware Accelerated Cosmology Code (HACC) simulation code, including checkpointing, restarts and the analysis outputs produced by the simulation.

III. Methodology

The testbed is an on-premise cluster in an academic lab. The cluster has a total of 10 nodes with 4 clients, 2 metadata servers (MDS), 1 management server (MGS), and 3 object storage servers (OSS), each with 3 object storage targets (OST). It is equipped with BeeGFS-7.3.2 with 4.5TB of total capacity. Each client node has 8GB of memory. All nodes in the cluster are connected via a 1Gbps Ethernet link. The OSSs are each equipped with 32GB of DRAM. Fig. 3 shows the cluster setup with corresponding the BeeGFS services. $S_x\_T_y$ represents the y^{th} target in the x^{th} server.

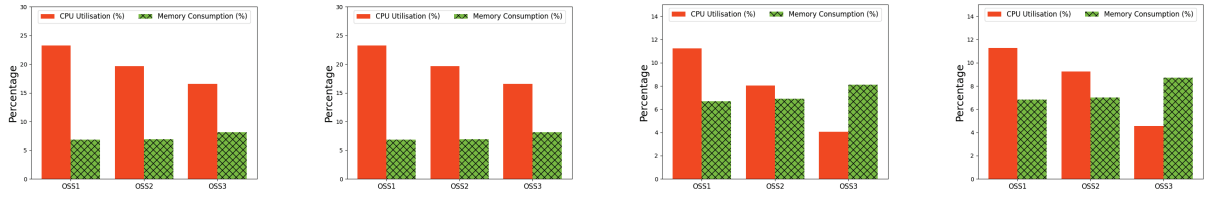
IOR is used to answer the research questions RQ1 and RQ2. For each question, we analyze the effects on a small file (7 MiB) and a large file (56 GiB). In some cases, we also used a medium-sized file (112 MiB). We vary the file access patterns—single shared file (N-1) and file per process (N-N). To study the effects of concurrently running applications, we run IOR, DLIO, and HACC-IO simultaneously. All results

979-8-3503-7062-1/23/$31.00 ©2023 IEEE
DOI 10.1109/CLUSTERWorkshops61457.2023.00010



(a) Round-Robin allocation, large file, 4 targets. **(b)** Random allocation, large files, 4 targets. **(c)** Round-Robin allocation, small files, 3 targets. **(d)** Random allocation, medium files, 3 targets.

Figure 1: OST load distribution for different allocation strategies and file sizes (chunk size = 512 KiB).



(a) Round-Robin allocation, large files, 4 targets. **(b)** Random allocation, large files, 4 targets. **(c)** Round-Robin allocation, small files, 3 targets. **(d)** Random allocation, medium files, 3 targets.

Figure 2: OSS CPU and memory utilization for different OST allocation strategies and file sizes (chunk size = 512 KiB).

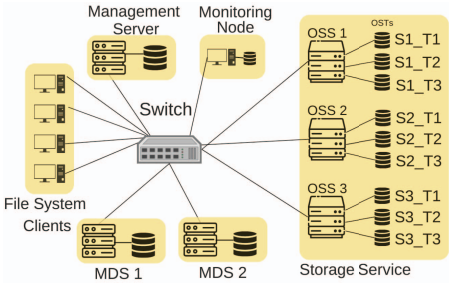


Figure 3: Architecture of the testbed.

are the average of at least three runs. Across experimental runs, caches are cleared to remove any caching effect.

The CPU, memory and network utilization of the cluster are monitored using the ELK stack [9]. The benchmark tools measure the I/O throughput and latency of the workloads. The `beegfs-df` command is used to check the load distribution of the chunks across the OSTs.

IV. Analysis

In this section, we try to objectively find answers to the research questions that will help evaluate the performance of BeeGFS, its load balancing ability, the differing stripe count and the chunk size parameters, and concurrent runs of different applications.

A. Data distribution on OSTs

To check the load distribution across OSSs and OSTs, we use the default random and the provided round-robin OST allocation strategy. The default striping configuration of BeeGFS has a chunk size of 512 KiB and a stripe count of 4.

Fig. 1a and 1b show the OST-level distribution of data written for large files with the round-robin and random OST allocation strategies respectively. Fig. 1c and 1d show the OST-level distribution of data written when the stripe count is 3, for small files with round-robin allocation and medium files with random OST allocation.

Fig. 2a and 2b show the OSS level statistics—CPU and memory utilization—for large files using round-robin and random allocation respectively. Fig. 2c and 2d similarly show the OSS level statistics for the stripe count of 3 for small files with round-robin allocation and medium files with the random OST allocation.

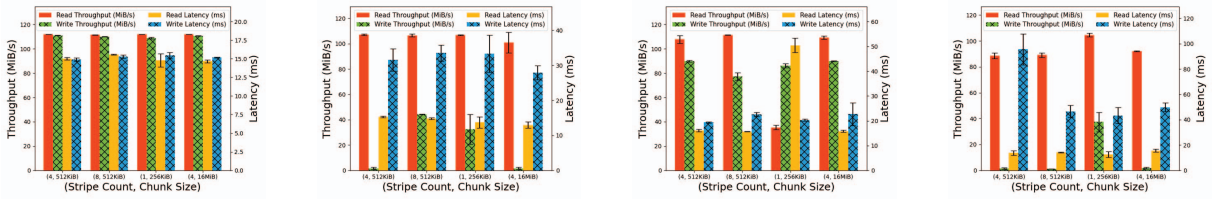
Observation: OSS 1 receives the majority of the data load. This leads to load imbalance at both the OST level and OSS level. CPU utilization of OSS 1 is higher than the CPU utilization of other OSSs. However, memory utilization does not follow any specific trend.

Lesson Learned #1

Default OST allocation strategies lead to load imbalance at OSS and OST levels. CPU utilization and network usage are correlated to the load distribution.

B. Varying Stripe Count and Chunk Size

We use the chunk sizes 256 KiB, 512 KiB, 1 MiB, 4 MiB, and 16 MiB. The stripe count is selected as 1, 4, and 8. Due to space constraints, we show only representative striping configurations.



(a) Large file, N-N access. **(b)** Small file, N-N access. **(c)** Large file, N-1 access. **(d)** Small file, N-1 access.

Figure 4: Read-write throughput and latency for different stripe counts and chunk sizes.

Fig. 4a and 4b shows the I/O throughput and latency for N-N workloads for large files and small files respectively. Fig. 4c and 4d show the throughput and latency for N-1 workloads for large and small files, respectively.

Observation: For large aggregate file sizes, varying the stripe count and chunk size does not greatly affect I/O performance. For small files, the default configuration performs sub-optimally. Writing to a single OST improves overall performance, due to lower write latency. Such a configuration is well-suited for write-intensive applications. N-1 workloads, when accessing a large file, are not significantly affected by stripe count and chunk size. However, a few combinations perform very poorly in terms of read latency—for example, (1, 256 KiB). For small files, the default striping has relatively subpar performance for both read and write operations.

1) Concurrent Workloads: Fig. 5 shows the throughput of IOR, HACC-IO, and DLIO when using the default striping configuration compared with using different striping patterns for each application.

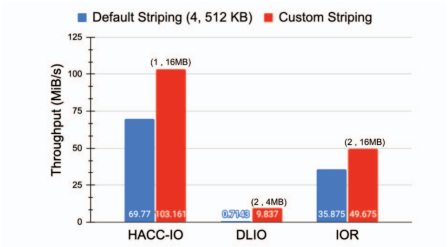


Figure 5: Performance of concurrently running applications with default and custom striping patterns.

Observation: IOR, HACC-IO, and DLIO perform better with stripe count and chunk size of (1, 16 MiB), (2, 16 MiB) and (2, 4 MiB), respectively, when compared to the default striping configuration.

Lesson Learned #2

The default striping configuration of BeeGFS leaves I/O performance on the table. Each concurrently running HPC application needs a different striping configuration for maximizing performance.

V. Conclusion

BeeGFS is gaining popularity among small-scale HPC clusters. Our work studied the effects of varying striping configurations for various I/O workloads. We observed that the default OST allocation strategies lead to load imbalance on OST and OSS levels. The default striping configuration gives sub-optimal performance for various workloads. These results highlight the need for a tool for adaptive striping configuration based on the I/O characteristics of a workload.

Acknowledgment

This work is sponsored in part by the grants BBF/BITS(G)/FY2022-23/BCPS-123, GOA/ACG/2022-2023/Oct/11, and BPGC/RIG/2021-22/06-2022/02.

References

[1] J. Bang, C. Kim, K. Wu, A. Sim, S. Byna, H. Sung, and H. Eom, "An in-depth i/o pattern analysis in hpc systems," in *HiPC Conference*, pp. 400–405, 2021.

[2] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu, "I/o characterization and performance evaluation of beegfs for deep learning," in *ICPP Conference*, pp. 1–10, 2019.

[3] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley, "Access patterns and performance behaviors of multi-layer supercomputer i/o subsystems under production load," in *HPDC Conference*, pp. 43–55, 2022.

[4] T. Patel, S. Byna, G. K. Lockwood, N. J. Wright, P. Carns, R. Ross, and D. Tiwari, "Uncovering access, reuse, and sharing characteristics of {I/O-Intensive} files on {Large-Scale} production {HPC} systems," in *USENIX FAST Conference*, pp. 91–101, 2020.

[5] "Key aspects of beegfs." https://www.beegfs.io/c/home/key-aspects-of-beegfs/. Accessed: August 21 2023.

[6] "IOR Benchmark." https://ior.readthedocs.io/. Accessed: August 21 2023.

[7] "Deep learning i/o (dlio) benchmark," https://github.com/argonne-lcf/dlio_benchmark. Accessed: August 21 2023.

[8] "Coral benchmark codes." https://asc.llnl.gov/coral-benchmarks#hacc. Accessed: August 21 2023.

[9] "Elastic stack: Elasticsearch, kibana, beats logstash." https://www.elastic.co/elastic-stack. Accessed: August 21 2023.