GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

# On Demand File Systems with BeeGFS

Zoya Masih

MatrNr: 19034321

Supervisor: Prof. Dr. Julian Kunkel

Georg-August-Universität Göttingen
Institute of Computer Science

April 3, 2023

# Abstract

BeeGFS, originally released as FhGFS in 2005, is a parallel file system, specifically designed for HPC, which aggregates the capacity and performance of many servers in a single namespace. It therefore can scale up performance and capacity to a required level. BeeGFS splits metadata from user's stored data, and provides an on demand file system too, BeeOND, which integrates with cluster batch systems to create temporary parallel file system instances on a per-job basis on the internal SSDs / NVMe of compute nodes, which are part of a compute job.

In this report, an overview and theoretical concepts of the parallel file system BeeGFS are presented. Additionally, the setup procedure and use of some other tools of BeeGFS on virtual machine are described to help others know it works and start it.

# Contents

# List of Tables

# List of Figures



# List of Listings

# List of Abbreviations

**AI** Artificial Intelligence

**GWDG** Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

**HPC** High-Performance Computing

**I/O** Input/Output

**BeeGFS** Bee Global File System

**NFS** Network File System

**ext4** fourth extended filesystem

**POSIX** Portable Operating System Interface for uniX

# 1 Introduction

BeeGFS is a hardware-independent POSIX parallel file system, developed with a strong focus on performance and designed for ease of use, simple installation, and management. BeeGFS is widely seen as an easy-deployable alternative to other parallel file systems and is deployed at thousands of sites around the globe. It is created on an Available Source development model. BeeGFS is designed for all performance-oriented environments including HPC, AI and Deep Learning, Media & Entertainment, Life Sciences, and Oil & Gas.
The BeeGFS user space architecture is "state-of-the-art" allowing users to manage any IO profile requirements without performance restrictions and provides the scalability & flexibility that is required to run the most demanding HPC, AI, or business-critical applications. BeeGFS allows customers to invest in scalable HPC and AI infrastructures, that deliver from small sites to large scale-out environments, relieving the full bandwidths of their hardware components. BeeGFS increases productivity by delivering faster results, enabling new data analysis methods without changing workflows or applications.
By increasing the number of servers and disks in the system, users can simply scale performance and capacity of the file system to the level that they need, seamlessly from small clusters up to enterprise-class systems with thousands of nodes. Therefore, BeeGFS is being used on diverse computer clusters, ranging from installations with only a few machines to several systems of the Top500 of the world's fastest supercomputers. Furthermore, the file system is a fundamental component of lots of research projects led by different research organizations and governmental institutions.

## 1.1 Usage in HPC

In 2005, the founders came to the idea that available parallel file systems neither meet the users' expectation, nor were as easy to use and maintain as system administrators demand it. This motivated them to develop a new parallel file system.
It became soon clear that the file system quickly evolved to a state which made it useful for HPC users. BeeGFS was developed at the Fraunhofer Institute for industrial mathematics (ITWM)[1]. It was originally released as "FhGFS", but was later labelled as BeeGFS in 2014 [Jan14].
A spin-off of Fraunhofer ITWM, ThinkparQ, supplies worldwide commercial support for BeeGFS and manages further development from a customer perspective. BeeGFS is an important part of three European HPC projects, which develop specialized computer architectures for the exa-scale regime: DEEP-ER, EXANODE and EXANEST. For the DEEP-ER project, which proposes a cluster-booster architecture, the parallel file system BeeGFS is extended to use the different hierarchy levels of the storage systems efficiently. The EXANODE and EXANEST projects, use energy-efficient processors and nanotechnologies. The projects use a system-wide, uniform memory concept. Many of the presented ideas will be used for the first time in high performance computing.
BeeGFS is used worldwide these days. NASA Ames Research Center, and European Organization for Nuclear Research are two organizations which use this file system in HPC.

[1] https://www.itwm.fraunhofer.de/

## 1.2 Literature Review

In the following, a brief review of papers on BeeGFS is presented. In [PC21], the impact of some parameters such as the transfer size and the number of processes on optimizing I/O performance of geophysical applications are studied. A comparison between different parallel file systems is provided by [FL18], in which BeeGFS shows the best throughput performance. The paper concludes that BeeGFS could be part of a solution for maximizing the performance of scientific data transfer between different clusters. In another research, Chowdhury et al. [Fah+19] evaluated the impact of the number of storage targets and concluded that increasing the stripe count has limited benefits for application performance. Although, the results of a related study on the impact of the stripe count in write performance [PT22], conflict with some recommendations of [Fah+19].

## 1.3 Goals of the project

The current document is a report for a project in the practical seminar course 'High-Performance Computing System Administration'. The goals which were expected to be fulfilled during this project are as follows.

1. Study the architecture and all theoretical features of BeeGFS
2. Setting up the file system in virtual machines to allow testing of the features and providing a replicable environment for code developers

The outline of the report is as follows. In Section two, the architecture and theoretical features of the file system are described. In section 3 the procedure of setting up the file system on virtual machines is presented. Section four provides some practical executions on the VMs.

# 2 Architecture

In this section, the architecture and theoretical features of the file system are studied.

## 2.1 Overview

The BeeGFS architecture is composed of four main services:

1. Management service: A registry and watchdog for all other services
2. Storage service: Stores the distributed user file contents
3. Metadata service: Stores access permissions and striping information
4. Client service: Mounts the file system to access the stored data

The figure 1 shows main services of the system and how they connect to each other.
The optional monitoring service collects performance data from the servers and feeds it to a time series database, for example InfluxDB. From this, real time monitoring and statistical analysis of the system is possible with tools like Grafana. For high flexibility, it is possible to run multiple services instances with any BeeGFS service on the same

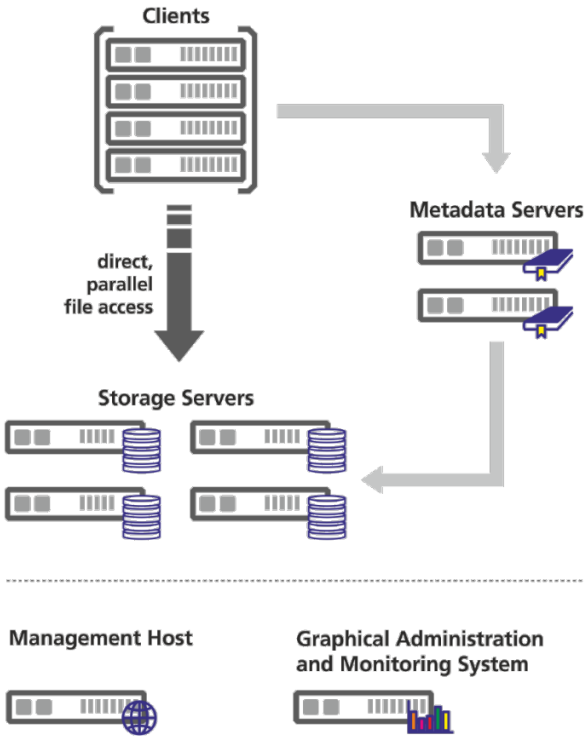


Figure 1: Architecture [Bee]

machine. These instances can be part of the same BeeGFS file system instance or as well of different file system instances. For example, client service can mount two different BeeGFS file systems (e.g., an old one and a new one) on the same compute node. The flexibility is also seen in the sense of hardware. The software-based approach, without any strict requirements for the hardware, provides the possibility to choose from a very wide range of hardware components.

## 2.2 Management Service

The management service can be figured as a joint for the BeeGFS metadata, storage, and client services. It is very lightweight and typically not running on a dedicated machine, as it is not critical for performance and stores no user data. It observes and checks how all the components are working and manages the relations between every two parts. Therefore, it is the first service, which needs to be set up in a newly deployed environment. The management service maintains a list of all other BeeGFS services and their state.

## 2.3 Metadata Service

The metadata service stores information about the data, e.g., directory information, file directory ownership, and the location of user file contents on storage targets. It provides information about the stripe pattern for an individual user file. The metadata service is not involved in data access, i.e., for file read and write operations.
There can be one or many metadata services in a BeeGFS file system. Each metadata service is responsible for its exclusive fraction of the global namespace, so that having more metadata servers improves the overall system performance. Adding more metadata servers later is always possible. Each metadata service instance has exactly one metadata

target to store its data. On the metadata target, BeeGFS creates one metadata file per user-created file.
Usually, not necessarily, a metadata target is an ext4 file system based on a RAID1 or RAID10 of flash drives, as low metadata access latency improves the responsiveness of the file system. BeeGFS metadata is very small and grows linear with the number of user-created files. 512 GB of usable metadata capacity are typically good for about 150 million user files. Normally, 0.5 percent of all data stored is recommended for metadata, however, obviously it may differ in different systems based on the system specific features.

## 2.4 Storage Service

The storage service is the main service to store striped user file contents. There can be one or multiple storage services per BeeGFS file system instance, so that each storage service adds more capacity and especially also more performance to the file system. A storage service instance has one or multiple storage targets. While such a storage target can generally be any directory on a local file system, and storage service works with any local Linux POSIX file system, a storage target typically is a hardware RAID-6 or zfs. As BeeGFS uses all the available RAM on the storage servers automatically for caching, it can also aggregate small IOs requests into larger blocks before writing the data out to disk. Furthermore, it is able to serve data from the cache if another client has already recently requested it. The capability to quickly write bursts of data into the server RAM cache or to quickly read data from it is also the reason it makes sense to have a network that is significantly faster than the disk streaming throughput of the servers. By default, BeeGFS picks the storage targets for a file randomly, as this has shown to provide best results in multi-user environments where the different users are also concurrently creating a random mix of large and small files. If necessary, different target choosers are available in the metadata service configuration file. To prevent storage targets running out of free space, BeeGFS has three different labels for free target capacity: normal, low and emergency. The target chooser running on the metadata service will prefer targets labeled as normal. As long as such targets are available, and it will not pick any target labeled as critical before all targets entered that state. With this approach, BeeGFS can also work with storage targets of different sizes. The thresholds for low and emergency can be changed in the management service configuration file.

## 2.5 Client Service

The BeeGFS client is implemented as a Linux kernel module which provides a normal mount-point so that your applications can directly access the BeeGFS storage system and do not need to be modified to take advantage of BeeGFS. The module can be installed on all supported Linux kernels without the need for any patches.

## 2.6 More about the Architecture

When a file arrives to the storage service, it gets split up into chunks of fixed size and those chunks are distributed across multiple storage targets. The chunk size and number of targets per file is decided by the responsible metadata service when a file gets created. This information is called the stripe pattern. The stripe pattern can be configured per directory or even for individual files. The files on the storage targets containing the user

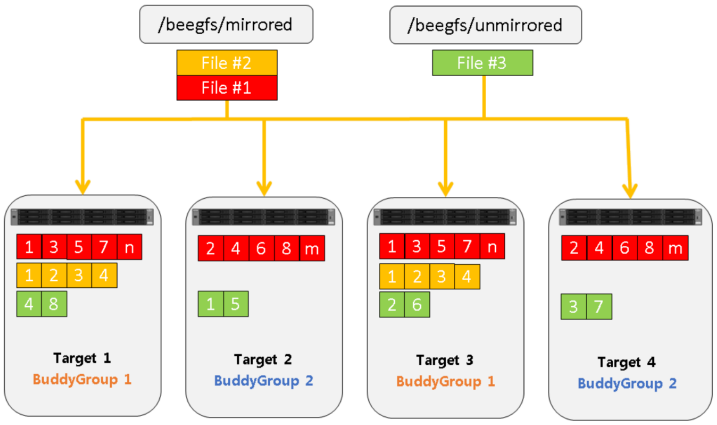


Figure 2: Striped files

data are called chunk files. For each user file, there is exactly one chunk file on the corresponding storage targets. To not waste space, BeeGFS only creates chunk files when the client actually writes data to the corresponding target. And also, for not wasting space, the chunk size is not statically allocated, meaning when the user writes only a single byte into the file, BeeGFS will also create only a single chunk file of 1 byte in size.

BeeGFS also provides support for metadata and file contents mirroring. Mirroring capabilities are integrated into the normal BeeGFS services, so that no separate services or third-party tools are needed. Both metadata mirroring and file contents mirroring can be used independently of each other. Mirroring also provides some high availability features. Storage and metadata mirroring with high availability is based on so-called buddy groups. In general, a buddy group is a pair of two targets that internally manage data replication between each other. The buddy group approach allows one half of all servers in a system to fail while all data is still accessible. It can also be used to put buddies in different failure domains or different fire domains, e.g., different racks or different server rooms. Note that mirroring is not a replacement for backups. If files are accidentally deleted or overwritten by a user or process, mirroring will not help to bring the old file back. Mirroring is not activated by default, and needs to be set up manually, or one can tell BeeGFS to create them automatically. Figure 2 may help to understand the concepts of mirroring and buddy groups.

Another option in BeeGFS is storage pools. This allows the cluster admin to group targets and mirror buddy groups together in different classes. For instance, there can be one pool consisting of fast, but small solid state drives, and another pool for bulk storage, using big but slower spinning disks. When a file is created by an application or user, they can choose which storage pool the file contents are stored in. Since the concept of storage pools is orthogonal to the file system's directory structure, even files in the same directory can be in different storage pools.

## 2.7 BeeOND

BeeOND, BeeGFS On Demand, enables easy creation of one or multiple BeeGFS instances on the fly. This ability can be useful in many use-cases, for example, in cloud environments, but especially to create temporary work file systems. BeeOND is typically

used to aggregate the performance and capacity of internal SSDs or hard disks in compute nodes for the duration of a compute job. This provides additional performance and a very elegant way of burst buffering.

BeeOND, dislike most of the HPC file systems, creates a shared parallel file system for each job across all compute nodes which are involved in the job, to provide the advantage of a single name space across multiple machines, and the flexibility and performance of a shared parallel file system. This way, the internal compute node drives are turned into a shared parallel file system, which can easily be used for distributed applications. Here applications can complete faster, because with BeeOND, they can be running on SSDs or a RAM-disk, while they might only be running on spinning disks on a normal persistent global file system. Combining the SSDs of multiple compute nodes not only gets to high bandwidth easily, it also gets to a system that can handle very high IOPS.

# 3 Setup

In this section, we describe how to set up the File system on virtual machines.

## 3.1 Installation

In this project, Management, storage, and metadata services are set up on one virtual machine and two client services are also installed on two other machines. All the operating systems in this test system are Ubuntu 20.04. node recognition among the service nodes is a vital point here, and therefore all the virtual machines must be set on bridge mode, and 'Allow All' for the promiscuous. To set up the file system, follow the steps described as follows.

1. Get the Sources

   ```
   $git clone https://git.beegfs.io/pub/v7 beegfs-v7
   $cd beegfs-v7
   ```

2. Install the dependencies
   `$less README.md`

   In Debian distribution, for instance:`$ sudo apt install build-essential autoconf automake pkg-config devscripts debhelper libtool libattr1-dev xfslibs-dev lsb-release kmod librdmacm-dev libibverbs-dev default-jdk ant dh-systemd zlib1g-dev libssl-dev libcurl4-openssl-dev libblkid-dev uuid-dev`

3. Build
   `$make`

4. Add the public BeeGFS GPG key to your package manager:
   https://www.beegfs.io/release/beegfs_7.3.3/gpg/GPG-KEY-beegfs

   In Debian distribution, for instance: `$wget -q -O - https://www.beegfs.io/release/beegfs_7.3.2/gpg/GPG-KEY-beegfs | apt-key add -`

5. Download the repository from https://www.beegfs.io/release/beegfs_7.3.2/dists/

   In Debian distribution, for instance:
   ```
   $wget https://www.beegfs.io/release/beegfs_7.3.2/dists/beegfs-focal.list
   -O /etc/apt/sources.list.d/beegfs.list
   $ apt install apt-transport-https
   $ apt update
   ```

6. Install the following BeeGFS services
   ```
   beegfs-mgmtd
   beegfs-storage
   beegfs-meta
   beegfs-client
   beegfs-helperd
   ```

7. Configurations: Save repository files in an appropriate path in all nodes, referring to: https://doc.beegfs.io/latest/advanced_topics/manual_installation.html

   In Debian distribution, and for the metadata service for instance:
   On node02: `/beegfs/sbin/beegfs-setup-meta -p /data/beegfs/beegfs_meta -s 2 -m node01`

   Here Node01 to Node04 are the nodes with Management, Metadata, Storage, and Client services in order. The term 'node 0x' in the configuration command line must be replaced to the ip address of the node.

8. Connection authentication

   - Create a file which contains a shared secret
     `$ dd if=/dev/random of=/etc/beegfs/connauthfile bs=128 count=1`
   - Ensure the file is only readable by the root user
     ```
     $ chown root:root /etc/beegfs/connauthfile
     $ chmod 400 /etc/beegfs/connauthfile
     ```
   - Copy the file to all hosts in the cluster (mgmtd, meta, storage, client, mon).
   - Edit `connAuthFile=/etc/beegfs/connauthfile` on configuration files of all services. Note that Configuration files are in `/etc/beegfs/beegfs-*.conf`.

   This step can be skipped by changing 'connDisableAuthentication = true' in all configuration files, however, it is highly recommended not to do so. By disabling this authentication part, the services may not work for many operations. Also note that after adding the authentication file to a new node, all the services must be restarted by `systemctl restart beegfs-*`.

9. Start the services

   ```
   $systemctl start beegfs-mgmtd
   $systemctl start beegfs-meta
   $systemctl start beegfs-storage
   $systemctl start beegfs-helperd
   $systemctl start beegfs-client
   ```

Now everything is ready to start working with the file system.

# 4 Getting Started

Before starting, or whenever needed, all available services for being installed can be checked by apt-search beegfs. As shown in Figure3, two services, `beegfs-client` and `beegfs-client-dkms` are provided. The latter package allows building the client kernel module using the DKMS infrastructure. The two packages are mutually exclusive, and only one of them can be installed on a system at any given time.

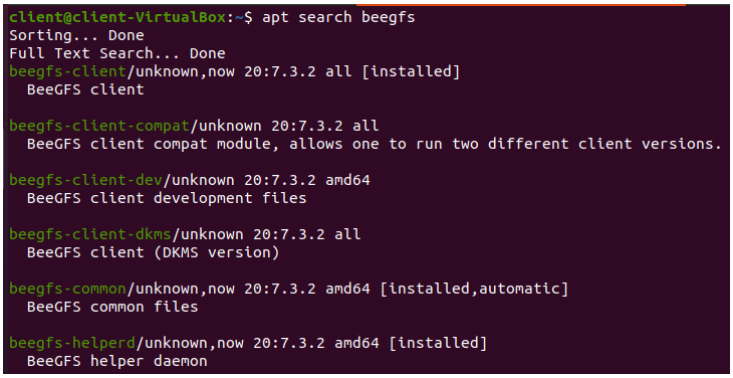


Figure 3: `apt search beegfs`

BeeGFS provides a set of tools that can be used to manage and monitor the file system. In this section, some of the tools are described and illustrated by screenshots.
Checking the connectivity can be a good start point. To use that, beegfs-utils is needed to be installed on the client nodes, for example, in Debian distribution we may use: `$apt install beegfs-utils`. Then the following command may help to consider the connection net.

- `beegfs-net` # connections the client is using
- `beegfs-check-servers` # possible connectivity of the services
- `beegfs-df` # free space & inodes of storage & metadata

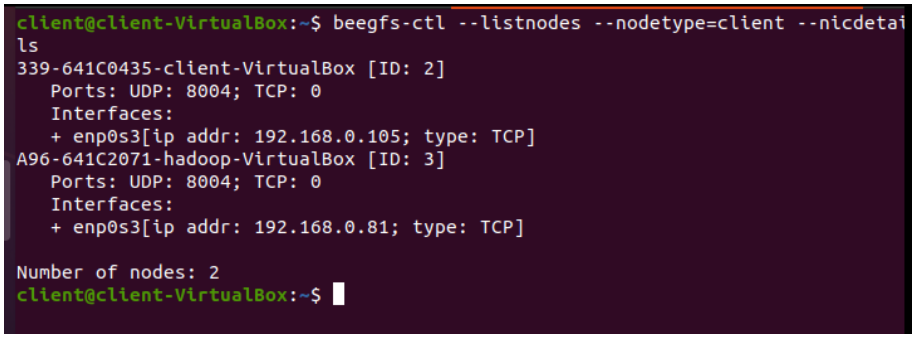


Figure 4: list of nodes

As we can see in Figure 5, in our case the three management, metadata, and storage services are installed on a same VM. Here the ID for each node is defined automatically by default, however it can be defined manually as an option. BeeGFS clients establish connections only when they are needed, and drop them after some idle time. Therefore,

the command beegfs-net can be here used on a client to see the number of currently established connections to each of the servers.

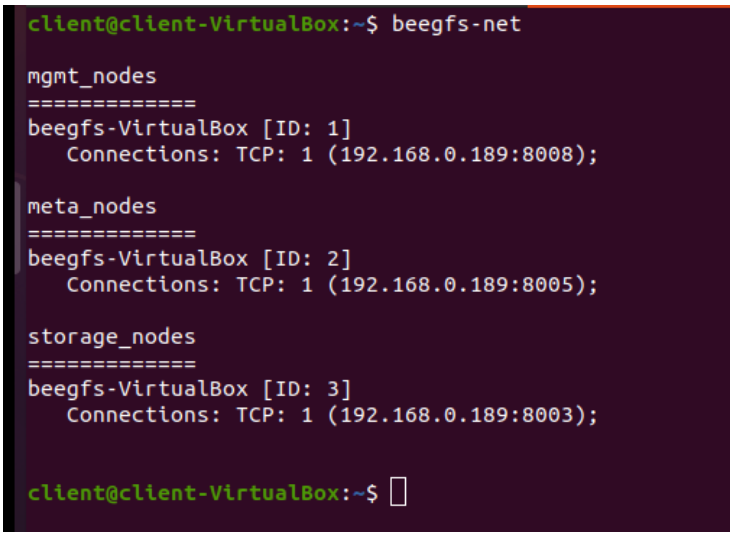


Figure 5: `beegfs-net`

In Figure 4, we can see that there are two client nodes in our small net. In a same way, storage and metadata nodes can be listed by following commands.

- `$beegfs-ctl - -listnodes - -nodetype=meta - -nicdetails`
- `$beegfs-ctl - -listnodes - -nodetype=storage - -nicdetails`
- `$beegfs-ctl - -listnodes - -nodetype=client - -nicdetails`

BeeGFS includes a built-in storage targets benchmark (StorageBench) and a built-in network benchmark (NetBench). The storage benchmark is started and monitored with the `beegfs-ctl` tool. For instance, the following example starts a write benchmark on all targets of all BeeGFS storage servers with an IO blocksize of 512 KB, using 10 threads per target, each of which will write 100 GB of data to its own file, see Figure 6, `$ beegfs-ctl -storagebench -alltargets -write -blocksize=512K -size=100G -threads=10`
In Figure 6, it can be seen that before invoking the benchmark command, the benchmark status is 'Not Run'. Additionally, IOR is a benchmark tool to measure the performance of a single or multiple clients with one or more processes per client. To use this benchmark, installing `beegfs-client-devel` is needed. Another provided tool for benchmarking is mdtest, which is a metadata benchmark tool, it needs MPI for distributed execution and can be used to measure values like file creations per seconds or stat operations per second of a single process or of multiple processes.
To add a new node to the net, it suffices to repeat all setup steps in the new node, and the system does not need a downtime.
In a case of problem in adding a node, make sure that in `/etc/beegfs/beegfs-mgmtd.conf` you have `sysAllowNewServers = true` and `sysAllowNewTargets = true`.

The storage services can be run on different machines and/or multiple storage targets on a same node, e.g., multiple RAID volumes. For having a second storage target on a same node, one can use the configuration command with replacing myraid2 to myraid1.


Figure 6: StorageBench

In this sense, data transferring from one to all other targets is possible by `$beegfs-ctl -migrate -targetid=<targetID> /mnt/beegfs/`. This can free up storage targets, to allow the removal of a target.

There is also another concept, Storage pools, which can be considered a template for the stripe pattern BeeGFS uses to assign file chunks to specific storage targets. Instead of directly assigning a stripe pattern, a user can now assign a storage pool. The tool `beegfs-ctl` can be used to check the target assignments and also for migrating files from one to another storage pool. Figure 7.

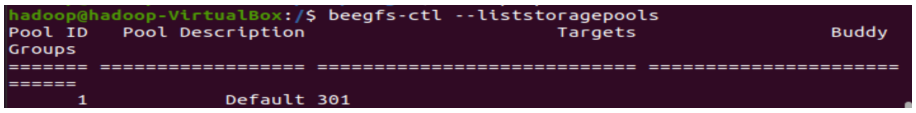


Figure 7: `liststoragepool`

The tool mirroring was explained in 2.6. Mirroring is disabled by default and can be enabled for both meta and storage separately. In figure 8, mirroring is activated for meta service. Before that, we have added meta service to a new node. The command here defines Buddy Groups automatically which suffices for our small system, however by adding `-primary= -secondary= -groupid=`, we can define the buddy targets and a group ID manually.
For retrieving information about a certain file or directory, such as entry ID and striping details, the command line `$ beegfs-ctl getentryinfo /mnt/beegfs/filename` can be used. For more details and other options of the command, `$ beegfs-ctl getentryinfo - -help` is useful. This mode is shown in Figure 9. In this example, the desired storage target is 4, which means if we have 4 storage targets, the file will be striped across them, in this experiment, we had just one storage target and therefore the actual number of storage targets is 1.

# 5 Conclusion

The parallel file system BeeGFS was provided to deal with the slow growth in IO performance, in the era of HPC emergence, in which compute-intensive applications spent a large portion of their execution time on I/O operations. A BeeGFS file system is a parallel

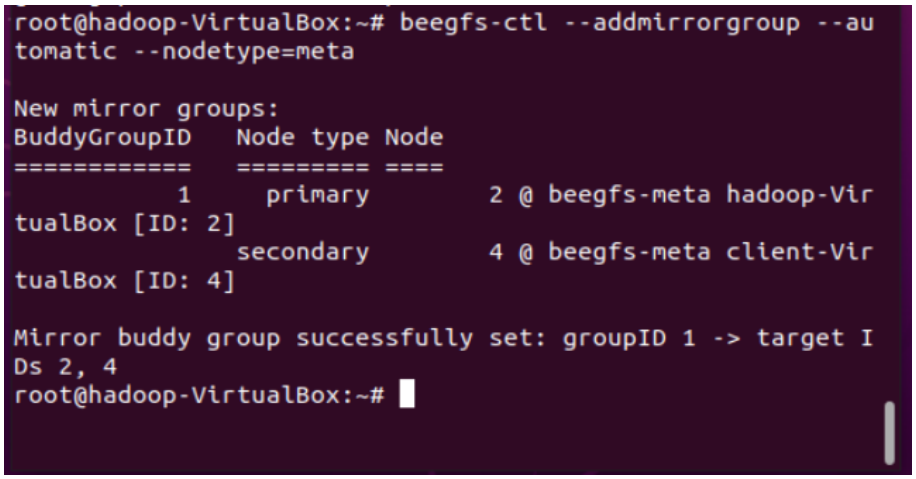


Figure 8: Mirroring

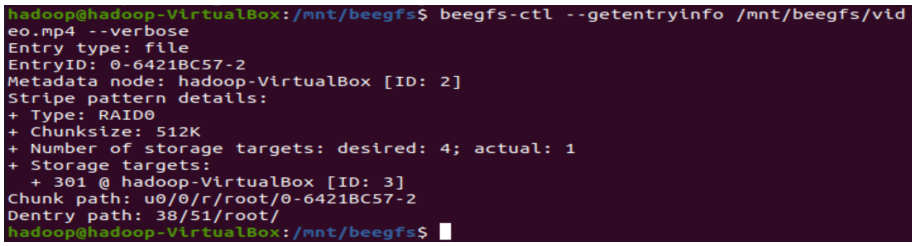


Figure 9: A file information

cluster file system managed with the BeeGFS software. BeeGFS architecture is composed of four main services: Management, Storage, Metadata, and Client services. BeeGFS ON Demand is one of many efficient tools in the file system which easily removes I/O load and nasty I/O patterns from a persistent global file system. By increasing the number of servers and disks in the system, it is possible to scale performance and capacity of the file system to the level that you need, seamlessly from small clusters up to enterprise-class systems with thousands of nodes.

In the current report, The architecture and some features of the file system were studied. The file system in this project was set up on virtual machines of Ubuntu OS. Finally, some start use of BeeGFS' tools were provided.

# References

[Bee] Official BeeGFSPage. In: URL: http://doc.beegfs.io/latest/architecture/overview.html.

[Fah+19] Fahim Chowdhury et al. "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning". In: (2019).

[FL18] Nicholas Mills. Alex Feltus and Walter Ligon. "Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks". In: (2018).

[Jan14] Jan Heichler. "An Introduction to BeeGFS". In: (2014).

[PC21] Jared Brzenski. Christofer Paolini. and Jose Castillo. "Improving the I/O of large geophysical models using PnetCDF and BeeGFS". In: (2021).

[PT22] Francieli Boito. Guillaume Pallez and Luan Teylo. "The role of storage target allocation in applications' I/O performance with BeeGFS". In: (2022).