thinkparQ

An introduction to BeeGFS®

Frank Herold, Sven Breuner

June 2018 – v2.0

Abstract

The scope of this paper is to give an overview on the parallel cluster file system BeeGFS and its basic concepts. It is intended as a first source of information for people interested in using BeeGFS and before moving on to more technical documentation or to a first installation.

Index

At-A-Glance

BeeGFS is a scale-out network file system based on POSIX, so applications do not need to be modified.

BeeGFS spreads data across multiple servers to aggregate capacity and performance of all servers.

1. Overview

BeeGFS is the leading parallel cluster file system. It has been developed with a strong focus on maximum performance and scalability, a high level of flexibility and designed for robustness and ease of use.

BeeGFS is a software-defined storage based on the POSIX file system interface, which means applications do not have to be rewritten or modified to take advantage of BeeGFS. BeeGFS clients accessing the data inside the file system, communicate with the storage servers via network, via any TCP/IP based connection or via RDMA-capable networks like InfiniBand (IB), Omni-Path (OPA) and RDMA over Converged Ethernet (RoCE). This is similar for the communication between the BeeGFS servers.

Furthermore, BeeGFS is a parallel file system. By transparently spreading user data across multiple servers and increasing the number of servers and disks in the system, the capacity and performance of all disks and all servers is aggregated in a single namespace. That way the file system performance and capacity can easily be scaled to the level which is required for the specific use case, also later while the system is in production.

BeeGFS is separating metadata from user file chunks on the servers. The file chunks are provided by the storage service and contain the data, which users want to store (i.e. the user file contents), whereas the metadata is the “data about data”, such as access permissions, file size and the information about how the user file chunks are distributed across the storage servers. The moment a client has got the metadata for a specific file or directory, it can talk directly to the storage service to store or retrieve the file chunks, so there is no further involvement of the metadata service in read or write operations.

BeeGFS adresses everyone, who needs large and/or fast file storage. While BeeGFS was originally developed for High Performance Computing (HPC), it is used today in almost all areas of industry and research, including but not limited to: Artificial Intelligence, Life Sciences, Oil & Gas, Finance or Defense. The concept of seamless scalability additionally allows users with a fast (but perhaps irregular or unpredictable) growth to adapt easily to the situations they are facing over time.

An important part of the philosophy behind BeeGFS is to reduce the hurdles for its use as far as possible, so that the technology is available to as many people as possible for their work. In the following paragraphs we will explain, how this is achieved.

BeeGFS is open-source and comes with optional Enterprise features like high availability.

BeeGFS is open-source and the basic BeeGFS file system software is available free of charge for end users. Thus, whoever wants to try or use BeeGFS can download it from www.beegfs.io. The client is published under the GPLv2, the server components are published under the BeeGFS EULA.

The additional Enterprise Features (high-availability, quota enforcement, and Access Control Lists) are also included for testing and can be enabled for production by establishing a support contract with ThinkParQ. Professional support contracts with ThinkParQ ensure, that you get help when you need it for your production environment. They also provide the financial basis for the continuous development of new features, as well as the optimization of BeeGFS for new hardware generations and new operating system releases. To provide high quality support around the globe, ThinkParQ cooperates with international solution partners.

System integrators offering turn-key solutions based on BeeGFS are always required to establish support contracts with ThinkParQ for their customers to ensure, that help is always available when needed.

BeeGFS can be accessed from different operating systems and runs on different platforms, including ARM and OpenPOWER.

Originally, BeeGFS was developed for Linux and all services, except for the client are normal userspace processes. BeeGFS supports a wide range of Linux distributions such as RHEL/Fedora, SLES/OpenSuse or Debian/Ubuntu as well as a wide range of Linux kernels from ancient 2.6.18 up to the latest vanilla kernels. Additionally, a native BeeGFS Windows client is currently under development to enable seamless and fast data access in a shared environment.

Another important aspect of BeeGFS is the support for different hardware platforms, including not only Intel/AMD x86_64, but also ARM, OpenPOWER and others. As the BeeGFS network protocol is independent of the hardware platform, hosts of different platforms can be mixed within the same file system instance, ensuring that sysadmins can always add systems of a new platform later throughout the life cycle of the system.

2. General Architecture

The BeeGFS architecture is composed of four main services:

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

BeeGFS file systems consist of 4 services: The management, storage, metadata, and client service.

In addition to the main services list above, BeeGFS also comes with an optional graphical administration and monitoring service (the so-called “admon”).

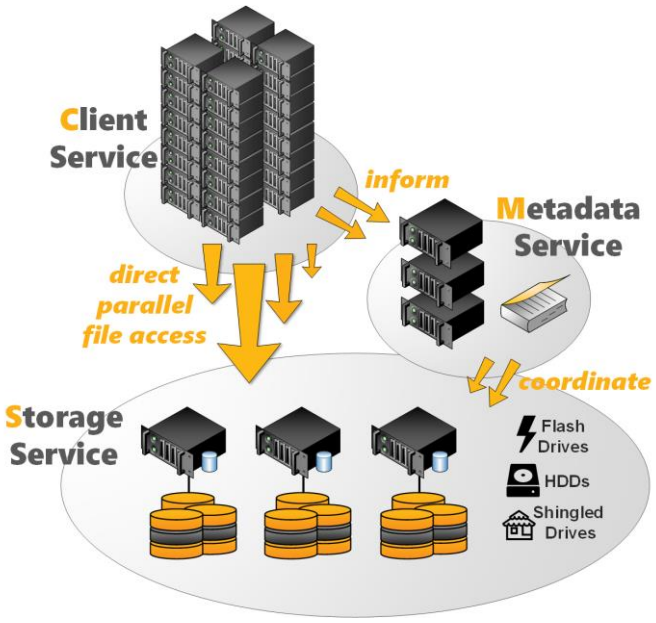

Figure 1: BeeGFS Architecture Overview

All BeeGFS services can run either exclusively on dedicated machines or side-by-side on the same machines.

All BeeGFS services write a log file with the corresponding service name to /var/log/beegfs-*.log

For high flexibility, it is possible to run multiple instances with any BeeGFS service on the same machine. These instances can be part of the same BeeGFS file system instance or as well of different file system instances. One typical example is the client service, that can mount two different BeeGFS file systems (e.g. an old one and a new one) on the same compute node.

BeeGFS runs on top of local Linux file systems, such as ext4, xfs or zfs.

High flexibility and easy administration is also given since the BeeGFS management, meta, and storage services do not access the disks directly. Instead, they store data inside any local Linux POSIX file system, such as ext4, xfs or zfs. This provides the flexibility to choose the underlying file system which works best for the given service, use case or hardware and makes it also easy to explore how BeeGFS stores files.

The underlying file system in which the BeeGFS services store their data are called management, metadata, or storage targets. These correspond to the name of the BeeGFS service, that uses the target to store its data. While the BeeGFS management and metadata service each use a single target per service instance, the storage service supports one or multiple storage targets for a single storage service instance.

This software-based approach without any strict requirements for the hardware provides the possibility to choose from a very wide range of hardware components. In the following chapters, we will discuss the BeeGFS services and flexibility in more detail.

2.1 The Management Service

Figure 2: Management Service

The management service is only the "meeting point" for the other services and not involved in actual file operations.

The management service can be figured as a "meeting point" for the BeeGFS metadata, storage, and client services. It is very light-weight and typically not running on a dedicated machine, as it is not critical for performance and stores no user data. It is watching all registered services and checks their state. Therefore, it is the first service, which needs to be setup in a newly deployed environment.

The management service maintains a list of all other BeeGFS services and their state.

2.2 The Metadata Service

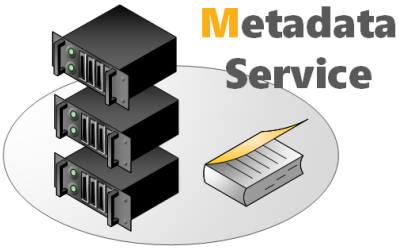

Figure 3: Metadata Service

The metadata service is a scale-out service, meaning there can be an arbitrary number of metadata services to increase performance.

The metadata service stores information about the data e.g. directory information, file and directory ownership and the location of user file contents on storage targets. It provides information about the location (the so-called "stripe pattern") for an individual user file to a client when the client opens the file, but afterwards the metadata service is not involved in data access (i.e. for file read and write operations) until the file is closed.

The BeeGFS metadata service is a scale-out service, meaning 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. This is an important design decision of BeeGFS to avoid the case of storing all metadata inside a single database that could possibly get corrupted.

Metadata is very small and typically stored on flash drives.

Usually, 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. 512GB of usable metadata capacity are typically good for about 150 million user files.

As low metadata access latency is a major benefit for performance of the overall system, faster CPU cores will improve latency.

2.3 The Storage Service

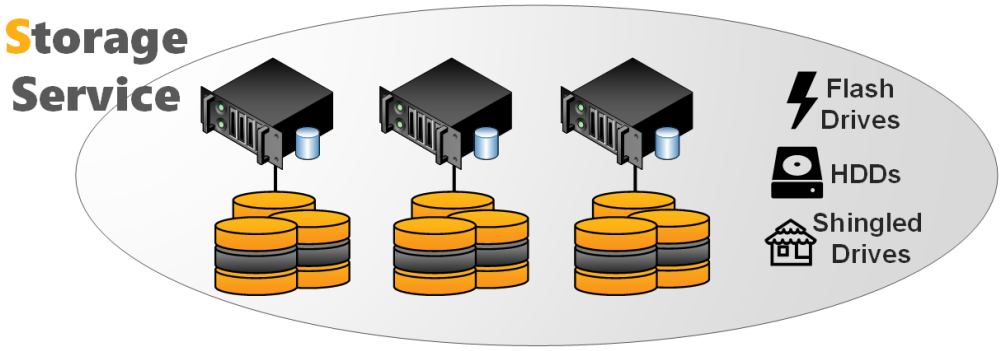

Figure 4: Storage Service

The storage service stores the user file contents.

The storage service (sometimes also referred to as the “object storage service”) is the main service to store striped user file contents, also known as data chunk files.

Similar to the metadata service, the BeeGFS storage service is based on a scale-out design. That means, you can have 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 filesystem, a storage target typically is a hardware RAID-6 (typically composed of 8+2 or 10+2) or zfs RAIDz2 volume, of either internal or externally attached drives.

The storage service typically uses RAID6 or RAIDz2 volumes as storage targets.

The storage service works with any local Linux POSIX file system. Usually, the storage targets are based on xfs in the case of hardware RAID controllers.

In contrast to the metadata service, many people try to optimize the traffic on the storage targets for large sequential access to have optimal performance on spinning disks. However, as BeeGFS uses all the available RAM on the storage servers (which is not otherwise allocated by processes) 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 it has already been recently requested by another client.

The capability to quickly write bursts of data into the server RAM cache or to quickly read data from it is also the reason why it makes sense to have a network that is significantly faster than the disk streaming throughput of the servers.

To distribute the used space and to aggregate the performance of multiple servers even for a single large file, BeeGFS uses striping, which means the file gets split up into chunks of fixed size and those chunks are distributed across multiple storage targets.

Even a single large file is distributed across multiple storage targets for high throughput.

The chunksize 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 (e.g. by using the beegfs-ctl command line tool) or even for individual files (e.g. by using the BeeGFS Striping API).

The files on the storage targets containing the user 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.

Different target chooser algorithms are available to influence the placement of files on storage targets.

By default, BeeGFS picks the storage targets for a file randomly, as this has shown to provide best results in multi-user environments where (from the point of view of the file system) the different users are also concurrently creating a random mix of large and small files. If necessary, (e.g. to have deterministic streaming benchmark results) 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 latter meaning only very little space left or the target is unreachable). 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.

2.4 The Client Service

The client is automatically built for the currently running Linux kernel, so that kernel updates require no manual intervention.

BeeGFS comes with a client that registers natively with the virtual file system interface of the Linux kernel for maximum performance. This kernel module has to be compiled to match the used kernel, but don't worry: The kernel module source code is included in the normal client package and compilation for the currently running Linux kernel happens fully automatically, so there are no manual steps required when you update your Linux kernel or when you update the BeeGFS client service. The installation or a BeeGFS client update can even be done without rebooting the machine.

The client kernel module uses an additional userspace helper daemon for DNS lookups and to write the log file.

When the client is loaded, it will mount the file systems defined in beegfs-mounts.conf instead of the usual Linux approach based on /etc/fstab (which is also possible with BeeGFS, but not recommended). This is an approach of starting the beegfs-client like any other Linux service through a service start script. It enables the automatic recompilation of the BeeGFS client module after system updates and makes handling of the BeeGFS client service generally more convenient.

Access to BeeGFS through NFS, CIFS or from Hadoop is also possible.

The native BeeGFS client should be used on all hosts that are supposed to access BeeGFS with maximum performance. However, it is also possible to re-export a BeeGFS mountpoint through NFSv4 or through Samba or to use BeeGFS as a drop-in replacement for Hadoop's HDFS. Upcoming releases of BeeGFS will also provide a native BeeGFS client for Windows.

2.5 Admon: Administration and Monitoring System

In addition to the beegfs-ctl command line tool, the optional BeeGFS **Ad**ministration and **Mon**itoring system (short: **admon**) provides a graphical interface to perform basic administrative tasks and to monitor the state of the file system and its components.

The BeeGFS admon consists of two parts:

- The admon backend service, which runs on any machine with network access to the metadata and storage services. This service gathers the status information of the other BeeGFS services and stores it in a database.

- The graphical Java-based client, which runs on your workstation. It connects to the remote admon daemon via http.

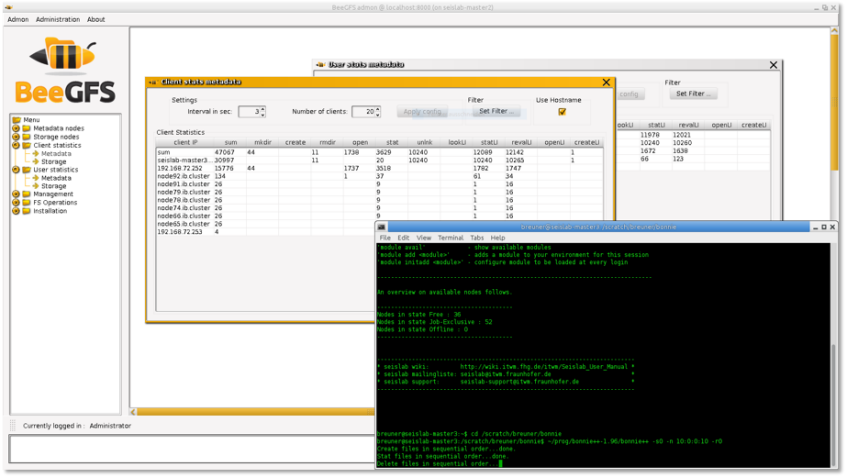

Figure 5: Admon GUI - Client Operation Statistics

3. Built-in Replication: Buddy Mirroring™

BeeGFS can easily reintegrate servers that were temporarily unavailable.

With BeeGFS being a high-performance file system, many BeeGFS users try to optimize their systems for best performance for the given budget. This is because with an underlying RAID-6, the risk of loosing data on the servers is already very low. So the only remaining risk for data availability is the relatively low risk of server hardware issues like mainboard failures. But even in this case, data is not lost, but only temporarily unavailable. Clients that need to access the data of unavailable servers will wait for a configurable amount of time (usually several minutes). If the server is available again within this time, the client will continue transparently for the application. This mechanism also enables transparent rolling updates. If the server does not come back within this time frame, the client will report an error to the application. This is useful to prevent the system from hanging completely when a server is down.

However, there are of course systems where this small risk of data unavailability is not acceptable.

The classic approach to this is hardware shared storage, where pairs of two servers have shared access to the same external JBOD. While this approach is possible with BeeGFS, it complicates the system administration due to the shared

disks and it requires extra services like pacemaker to manage the failover. This approach also only moves the availability problem from the servers to the shared JBOD and does not reduce the risk of data loss if there is a problem with a RAID-6 volume.

Thus, BeeGFS comes with a different mechanism that is fully integrated (meaning no extra services are required) and which does not rely on special hardware like physically shared storage. This approach is called Buddy Mirroring, based on the concept of pairs of servers (the so-called buddies) that internally replicate each other and that help each other in case one of them has a problem.

Integrated Buddy Mirroring provides increased flexibility and protection compared to classic shared storage approaches.

Compared to the classic shared storage approach, Buddy Mirroring has the advantage that the two buddies of a group can be placed in different hardware failure domains, such as different racks or even different adjacent server rooms, so that your data even survives if there is a fire in one of the server rooms. Buddy Mirroring also increases the level of fault tolerance compared to the classic approach of shared RAID-6 volumes, because even if the complete RAID-6 volume of one buddy is lost, the other buddy still has all the data.

For high flexibility, Buddy Mirroring can be enabled for all data or only for subsets of the data, e.g. based on individual directory settings, which are automatically derived by newly created subdirectories. Buddy Mirroring can also be enabled anytime later while data is already stored on the system.

Buddy Mirroring is synchronous (which is important for transparent failovers), meaning the application receives the I/O completion information after both buddies of a group received the data. The replication happens over the normal network connection, so no extra network connection is required to use Buddy Mirroring.

Buddy Mirroring can be enabled individually for the metadata service and the storage service.

3.1 Storage Service Buddy Mirroring

A storage service buddy group is a pair of two targets that internally manages data replication between each other. In typical setups with an even number of servers and targets, the buddy group approach allows up to a half of all servers in a system to fail with all data still being accessible.

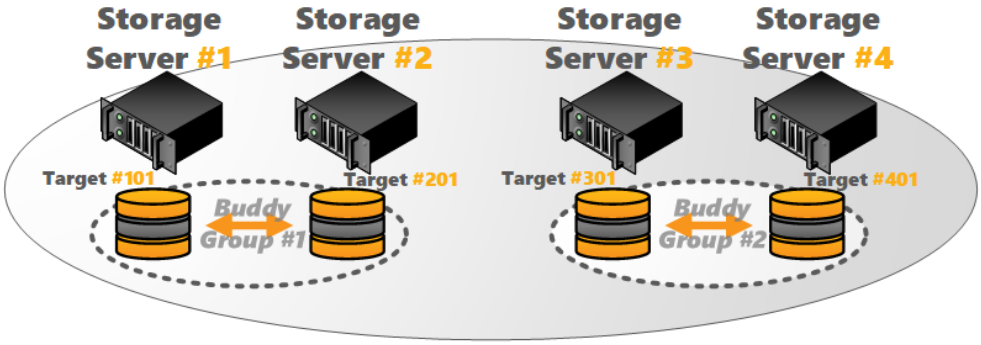

Figure 6: Storage Service Buddy Mirror Groups

Storage buddy mirroring can also be used with an odd number of storage servers. This works, because BeeGFS buddy groups are composed of individual storage targets, independent of their assignment to servers, as shown in the following example graphic with 3 servers and 2 storage targets per server. (In general, a storage buddy group could even be composed of two targets that are attached to the same server.)

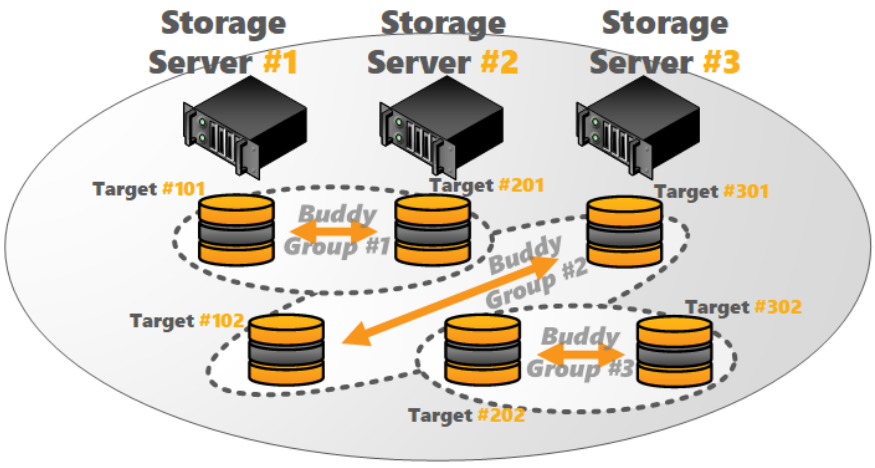

Figure 7: Storage Mirroring with odd Number of Servers

In normal operation, one of the storage targets (or metadata targets) in a buddy group is labeled as primary, whereas the other is labeled as secondary. These roles can switch dynamically when the primary fails. To prevent the client from sending the data twice (and thus reducing the maximum client network

throughput), modifying operations will always be sent to the primary by the client, which then takes care of the internal data forwarding to the secondary, taking advantage of the full-duplex capability of modern networks.

Buddy Mirroring takes advantage of the network's full-duplex capabilities for high throughput.

If the primary storage target of a buddy group is unreachable, it will get marked as offline and a failover to the secondary will be issued. In this case, the former secondary is going to be the new primary. Such a failover is transparent and happens without any loss of data for running applications. The failover will happen after a short delay to guarantee the consistency of the system while the change information is propagated to all nodes.

If a failed buddy comes back, it will automatically resynchronize with the primary. To reduce the time for synchronization, the storage buddies only synchronize files that have changed while the other machine was down.

The beegfs-ctl command line tool can be used to configure the buddy groups, enable or disable mirroring and to check the state of the buddies.

3.2 Metadata Service Buddy Mirroring

As described in 3.1 the same concept and methodology can be used for metadata service buddy mirroring.

4. Storage Pools

While all-flash systems usually are still too expensive for systems that require large capacity, a certain amount of flash drives is typically affordable in addition to the spinning disks for high capacity. The goal of a high-performance storage system should then be to take optimal advantage of the flash drives to provide optimum access speed for the projects on which the users are currently working.

Storage Pools were designed to provide full all-flash performance in systems with high capacity based on spinning disks.

One way to take advantage of flash drives in combination with spinning disks is to use the flash drives as a transparent cache. Hardware RAID controllers typically allow adding SSDs as transparent block level cache and also zfs supports a similar functionality through the L2ARC and ZIL. While this approach is also possible with BeeGFS, the effectiveness of the flash drives in this case is rather limited, as the transparent cache never knows which files will be accessed next by the user and how long the user will be working with those files. Thus, most of the time the applications will still be bound by the access to the spinning disks.

To enable users to get the full all-flash performance for the projects on which they are currently working, the BeeGFS storage pools feature makes the flash drives explicitly available to the users. This way, users can request from BeeGFS (through the beegfs-ctl command line tool) to move the current project to the flash drives and thus all access to the project files will be served directly and exclusively from the flash drives without any access to the spinning disks until the user decides to move the project back to the spinning disks.

The placement of the data is fully transparent to applications. Data stays inside the same directory when it is moved to a different pool and files can be accessed directly without any implicit movement, no matter which pool the data is currently assigned to. To prevent users from putting all their data on the flash pool, different quota levels exist for the different pools, based on which a sysadmin could also implement a time-limited reservation mechanism for the flash pool.

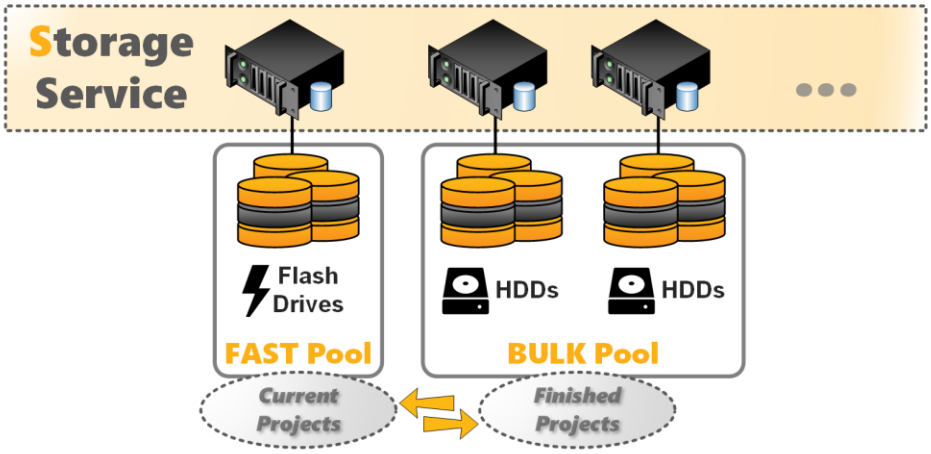

Figure 8: Storage Pools

Storage Pools are not limited to just two pools of flash drives and spinning disks.

However, while the concept of storage pools was originally developed to take optimum advantage of flash drives in a system with high capacity based on spinning disks, the concept is not limited to this particular use-case. The sysadmin can group arbitrary storage targets in such pools, no matter which storage media they use and there can also be more than just two pools.

A storage pool simply consists of one or multiple storage targets. If storage buddy mirroring is enabled, both targets of a mirror buddy group must be in the same pool, and the mirror buddy group itself must also be in the same storage pool.

5. BeeOND: BeeGFS On Demand

BeeOND elegantly unleashes the potential of the flash drives that you already have in the compute nodes.

Nowadays, compute nodes of a cluster typically are equipped with internal flash drives to store the operating system and to provide a local temporary data store for applications. But using a local temporary data store is often inconvenient or not useful for distributed applications at all, as they require shared access to the data from different compute nodes and thus the high bandwidth and high IOPS of the SSDs is wasted.

BeeOND (dubbed “beyond” and short for “BeeGFS On Demand”) was developed to solve this problem by enabling the creation of a shared parallel file system for compute jobs on such internal disks. The BeeOND instances exist temporary exactly for the runtime of the compute job exactly on the nodes that are allocated for the job. This provides a fast, shared all-flash file system for the jobs as a very elegant way of burst buffering or as the perfect place to store temporary data. This can also be used to remove a lot of nasty I/O accesses that would otherwise hit the spinning disks of your global file system.

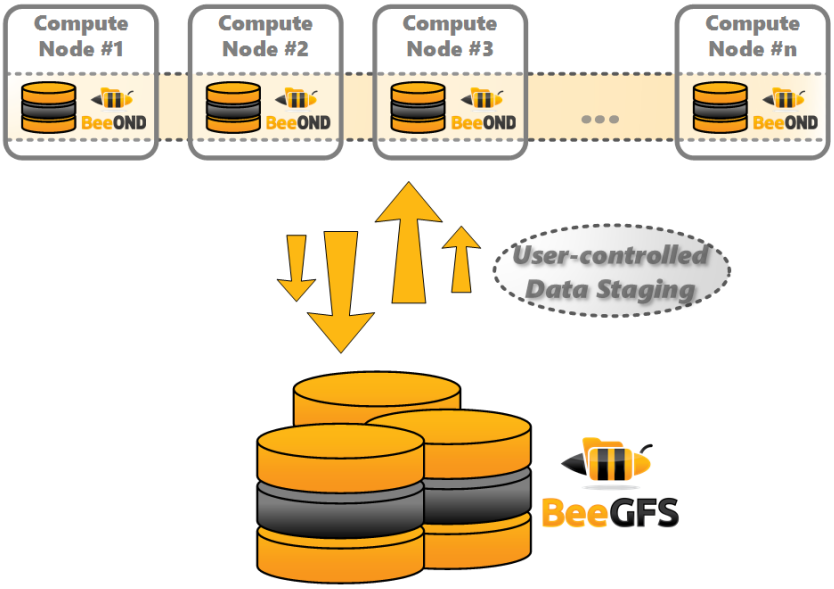

Figure 9: BeeOND: BeeGFS on demand

BeeOND is based on the normal BeeGFS services, meaning your compute nodes will also run the BeeGFS storage and metadata services to provide the shared

access to their internal drives. As BeeOND does not exclusively access the internal drives and instead only stores data in a subdirectory of the internal drives, the internal drives are also still available for direct local access by applications.

BeeOND can also be used when your global cluster file system is not based on BeeGFS.

While it is recommended to use BeeOND in combination with a global BeeGFS file system, it can be used independent of whether the global shared cluster file system is based on BeeGFS or on any other technology. BeeOND simply creates a new separate mount point for the compute job. Any of the standard tools (like cp or rsync) can be used to transfer data into and out of BeeOND, but the BeeOND package also contains a parallel copy tool to transfer data between BeeOND instances and another file system, such as your persistent global BeeGFS.

BeeOND instances can be created and destroyed with just a single simple command, which can easily be integrated into the prolog and epilog script of the cluster batch system, such as Torque, Slurm or Univa Grid Engine.

6. Cloud Integration

BeeGFS is available on the Amazon Web Services (AWS) as well as on Microsoft Azure.

BeeGFS can be used on-premise or in the cloud.

For the Amazon Web Services integration, it comes in two flavors: The community support edition is completely free of software charges, while the professional support edition adds a monthly fee. The BeeGFS instances can be launched directly from AWS Marketplace, where you can also find details on pricing model.

Launching BeeGFS on AWS will result in a fully working BeeGFS setup, which is readily available to store your large data sets for high performance access.

By default, the BeeGFS instances on AWS use Elastic Block Storage (EBS) volumes to preserve your data even when all virtual machines are shut down, so that you can come back later to resume working with your data sets and don’t need to keep your virtual machines running while you are not using them.

The Microsoft Azure implementation provides Resource Manager templates that will help deploying a BeeGFS instance based on CentOS 7.2.

Of course, if you are moving your datacenter into the cloud, it is also possible to perform your own BeeGFS installation for full flexibility instead of using the predefined templates and to get a normal BeeGFS professional support contract like on-premise setups.

7. Getting started and typical Configurations

BeeGFS can be tried out even with only a single machine for testing.

To get started with BeeGFS, all you need is a Linux machine to download the packages from www.beegfs.io and to install them on. If you want to, you can start with only a single machine and a single drive to do some initial evaluation. In this case, the single drive would be your management, metadata and storage target at the same time. The quickstart walk-through guide will show you all the necessary steps.

Typically, the smallest reasonable production system as dedicated storage is a single machine with two SSDs in RAID1 to store the metadata and a couple of spinning disks (or SSDs) in a hardware RAID6 or software zfs RAID-z2 to store file contents. A fast network (such as 10GbE, OmniPath or InfiniBand) is beneficial but not strictly necessary. With such a simple and basic setup, it will be possible to scale in terms of capacity and/or performance by just adding disks or more machines.

Often, BeeGFS configurations for a high number of clients in cluster or enterprise environments consist of fat servers with 24 to 72 disks per server in several RAID6 groups, usually 10 or 12 drives per RAID6 group. A network, usually InfiniBand, is used to provide high throughput and low latency to the clients.

In smaller setups (and with fast network interconnects), the BeeGFS metadata service can be running on the same machines as the BeeGFS storage service. For large systems, the BeeGFS metadata service is usually running on dedicated machines, which also allows independent scaling of metadata capacity and performance.

8. Contact Information

Questions?
info@thinkparq.com

If you want to keep up to date regarding news, events and features of BeeGFS, you should subscribe to the monthly newsletter at thinkparq.com/news or follow us on twitter. On the ThinkParQ news page, you will also find the release announcement list, a special low volume mailing list where you get a notification when a new BeeGFS release becomes available.

If you want to become part of the BeeGFS community, you can also join the BeeGFS user mailing list by subscribing at beegfs.io/support. On the public mailing list, users can ask other users for help or discuss interesting topics. Note that the

development team does not regularly check the posts on the public mailing list, as this is only part of the professional support.

If you are interested in more information, need help building your first BeeGFS storage system or want to become a BeeGFS partner for turn-key solutions, you can contact us at: info@thinkparq.com