BeeGFS unofficial documentation

Sep 24, 2019

Architecture

1 General Architecture **3**
- 1.1 The Management Service . . . 5
- 1.2 The Metadata Service . . . 5
- 1.3 The Storage Service . . . 5
- 1.4 The Client Service . . . 7
- 1.5 Admon: Administration and Monitoring System . . . 7

2 Built-in Replication: Buddy Mirroring **9**
- 2.1 Storage Service Buddy Mirroring . . . 10
- 2.2 Buddy Groups . . . 10
- 2.3 Metadata Service Buddy Mirroring . . . 11
- 2.4 Define Stripe Pattern . . . 11
- 2.5 Enabling and disabling Mirroring . . . 12
- 2.6 Restoring Metadata and Storage Target Data after Failures . . . 13
- 2.7 Caveats of Storage Mirroring . . . 14

3 Storage Pools **17**

4 Cloud Integration **19**

5 Striping **21**
- 5.1 Buddy Mirroring . . . 21
- 5.2 Impact on network communication . . . 22

6 Client Tuning **23**
- 6.1 Parallel Network Requests . . . 23
- 6.2 Remote fsync . . . 23
- 6.3 Disable locate/mlocate/updatedb . . . 24

7 Getting started and typical Configurations **25**

8 Installation and Setup **27**
- 8.1 General Notes . . . 27
- 8.2 GUI-based Installation and Service Management . . . 28

9 BeeOND: BeeGFS On Demand **29**

10 Admon **31**

10.1 Installation and basic Setup . . . 31
10.2 Admon GUI Start . . . 32
10.3 Admon Login . . . 32
10.4 Admon Main Menu . . . 32
10.5 Admon Menu Bar . . . 34

11 BeeGFS APIs Overview **35**

12 Contact Information **37**

13 Licensing **39**
13.1 Admon Licensing . . . 39

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 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.

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.

CHAPTER 1

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

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

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.

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.

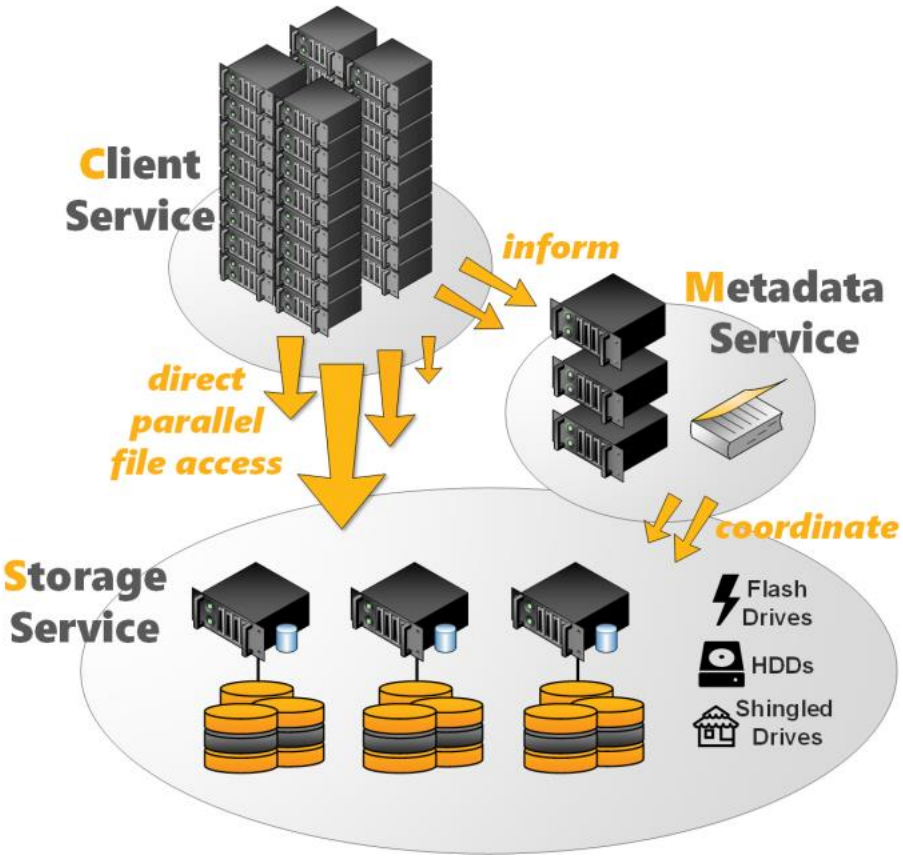

Fig. 1: BeeGFS Architecture Overview

Fig. 2: Management Service

1.1 The Management Service

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.

1.2 The Metadata Service

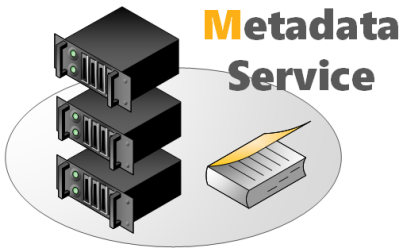

Fig. 3: Metadata Service

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.

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.

1.3 The Storage Service

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.

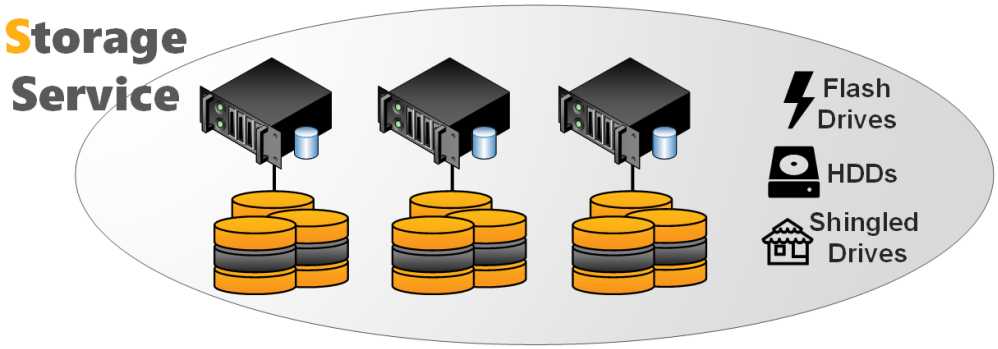

Fig. 4: Storage Service

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 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.

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.

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.

1.4 The Client Service

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.

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.

1.5 Admon: Administration and Monitoring System

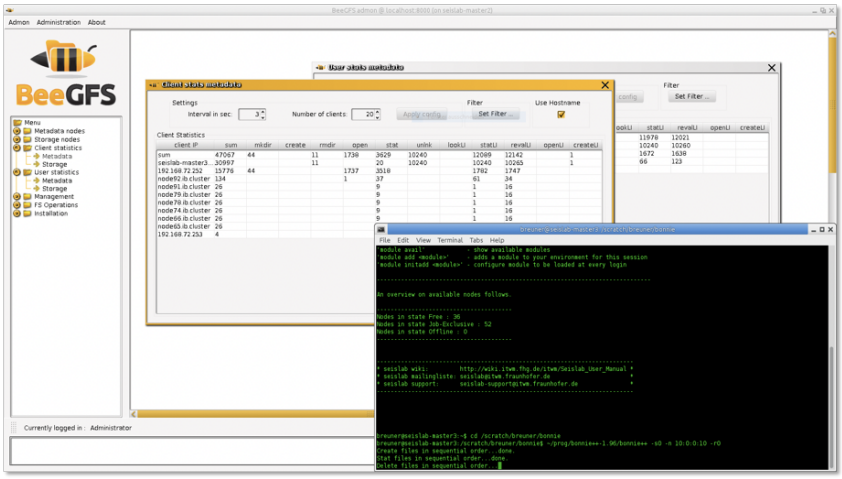

Fig. 5: Admon GUI - Client Operation Statistics

In addition to the beegfs-ctl command line tool, the optional BeeGFS Administration and Monitoring 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.

CHAPTER 2

Built-in Replication: Buddy Mirroring

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.

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.

2.1 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.

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.)

In normal operation, one of the storage targets (or metadata servers) 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.

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.

2.2 Buddy Groups

Mirror buddy groups are numeric IDs, just like the numeric IDs of the storage targets. Please note that buddy group IDs don't conflict with target IDs, i.e. they don't need to be distinct from storage target IDs.

There are basically two different ways to define buddy groups. They can be defined manually or you can tell BeeGFS to create them automatically.

Of course, defining groups manually gives you greater control and allows you to create more detailed configuration. For example, the automatic mode won't consider targets that are not equally sized. It also doesn't know about the topology of your system, so if you, for example, want to make sure that members of buddy groups are placed in different physical locations you have to define them manually.

2.2.1 Define Buddy Groups automatically

Automatic creation of buddy groups can be done beegfs-ctl, separately for metadata and for storage servers:

```
$ beegfs-ctl --addmirrorgroup --automatic --nodetype=meta
```

```
$ beegfs-ctl --addmirrorgroup --automatic --nodetype=storage
```

Please see the help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --addmirrorgroup --help
```

2.2.2 Define Buddy Groups manually

Manual definition of mirror buddy groups can be useful if you want to set custom group IDs or if you want to make sure that the buddies are in different failure domains (e.g. different racks). Manual definition of mirror buddy groups is done with the beegfs-ctl tool. By using the following command, you can create a buddy group with the ID 100, consisting of targets 1 and 2:

```
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=1 --secondary=2 --
↪groupid=100
```

Please see the help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --addmirrorgroup --help
```

When creating mirror buddy groups for metadata manually and one of them contains the root directory, it is necessary to set this one as primary.

2.2.3 List defined Mirror Buddy Groups

Configured mirror buddy groups can be listed with beegfs-ctl (don't forget to specify the node type):

```
$ beegfs-ctl --listmirrorgroups --nodetype=storage
```

```
$ beegfs-ctl --listmirrorgroups --nodetype=meta
```

It's also possible to list mirror buddy groups alongside other target information:

```
$ beegfs-ctl --listtargets --mirrorgroups
```

Please see the help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --listtargets --help
```

2.3 Metadata Service Buddy Mirroring

As described in *Storage Service Buddy Mirroring* the same concept and methodology can be used for metadata service buddy mirroring.

Note that odd numbers of storage this is not possible with metadata servers, since there are no metadata targets in BeeGFS. An even number of metadata server is needed so that every metadata server can belong to a buddy group.

2.4 Define Stripe Pattern

After defining storage buddy mirror groups in your system, you have to define a *data stripe* pattern that uses it.

2.5 Enabling and disabling Mirroring

By default, mirroring is disabled for a new file system instance. Both types of mirroring can be enabled with the beegfs-ctl command line tool. (The beegfs-ctl tool is contained in the beegfs-utils package and is usually run from a client node.)

Before metaadata or storage mirroring can be enabled, buddy groups need to be defined, as these are the basis for mirroring.

Storage mirroring can be enabled on a per-directory basis, so that some data in the file system can be mirrored while other data might not be mirrored. On the medatada side, it is also possible to activate or deactivate mirroring per directory, but certain logical restrictions apply. For example, for a directory to be mirrored effectively, the whole path to it must also be mirrored.

Mirroring settings of a directory will be applied to new file entries and will be derived by new subdirectories. For instance, if metadata mirroring is enabled for directory /mnt/beegfs/mydir1, then a new subdirectory /mnt/beegfs/mydir1/mydir2 will also automatically have metadata mirroring enabled.

After metadata mirroring is enabled for a file system using the beegfs-ctl –mirrormd command, the metadata of the root directory will be mirrored by default. Therefore, newly created directories under the root will also have metadata mirroring enabled. It is possible to exclude new folders from metadata mirroring by creating them using beegfs-ctl –createdir –nomirror.

To enable file contents mirroring for a certain directory, see the built-in help of the beegfs-ctl tool.

```
$ beegfs-ctl --setpattern --buddymirror --help
```

File contents mirroring can be disabled afterwards by using beegfs-ctl mode –setpattern without the –buddymirror option. However, files that were already created while mirroring was enabled will remain mirrored.

To check the metadata and file contents mirroring settings of a certain directory or file, use:

```
$ beegfs-ctl --getentryinfo /mnt/beegfs/mydir/myfile
```

To check target states of storage targets, use:

```
$ beegfs-ctl --listtargets --nodetype=storage --state
```

2.5.1 Activating Metadata Mirroring

After metadata server buddy groups have been defined as described under Management of Mirror Buddy Groups, metadata mirroring can be activated on the file system.

Note: When applying the beegfs-ctl –mirrormd command, no clients may be mounted. This can be achieved by stopping the beegfs-client service beforehand, and starting it again after beegfs-ctl –mirrormd was successfully performed. In addition, please restart the beegfs-meta service on all nodes afterwards.

To active metadata mirroring, use the beegfs-ctl tool:

```
$ beegfs-ctl --mirrormd
```

In order to synchronize all information across the various components of BeeGFS correctly, the clients can not be mounted during this process, and the metadata servers must be restarted afterwards.

The full series of steps in an already running system is: stop all clients, execute beegfs-ctl –mirrormd, restart all metadata servers, then restart all clients.

Please see the help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --mirrormd --help
```

Running this command will enable metadata mirroring for the root directory of the BeeGFS, as well as all the files contained in it. Note that existing directories except for the root directory will not be mirrored automatically. Please check *Migrating Existing Metadata* to find out how to mirror existing directories.

New files and directories created inside a directory with active metadata mirroring will also have metadata mirroring activated. There might be situations where it is desirable to deactivate metadata mirroring for part of the file systems. This gives a slight performance boost, but the data will not be preserved in the event of a server failure. A directory without metadata mirroring can be created using beegfs-ctl:

```
$ beegfs-ctl --createdir --nomirror <name>
```

For information about additional command line parameters, please see the beegfs-ctl help:

```
$ beegfs-ctl --createdir --help
```

Directories and files created inside a directory without metadata mirroring will have metadata mirroring disabled.

2.5.2 Migrating Existing Metadata

If a file is moved into a directory with active metadata mirroring, it will have its metadata mirrored. On the other hand, directories will not automatically be mirrored when moved. For a directory to be mirrored, it is therefore necessary to freshly create a directory inside a mirrored directory. The easiest way to enable mirroring for a whole directory tree is to do a recursive copy:

```
$ cp -a <directory> <mirrored-dir>
```

This will also copy the file contents, therefore it is possible to use it for enabling metadata and storage mirroring at the same time.

2.6 Restoring Metadata and Storage Target Data after Failures

If a storage target or metadata server is not reachable it will be marked as offline and won't get data updates. Usually, when the target or server re-registers it will automatically be synchronized from the remaining mirror in the buddy group (self-healing). However, in some cases it might be necessary that you manually start a synchronization process.

2.6.1 Automatic Resynchronization

In general, if a secondary target or server is considered to be out-of-sync, it is automatically set to the consistency state needs resync (see "Target and Node States" for more information on states) by the the management daemon.

The storage target resynchronization process for self-healing is coordinated by the primary target. The standard process tries to avoid unnecessary transfer of files. Therefore the primary target saves the time of the last successful communication with the secondary target. Only files which where modified after this timestamp will be resynchronized by default. To avoid losing cached data, a short safety threshold timespan will be added (defined by sysResyncSafetyThresholdMins in beegfs-storage.conf).

Since metadata are much smaller than storage contents, there is no timestamp-based mechanism in place, and instead the full mirrored metadata of the metadata server will be sent to its buddy during the resynchronization process.

2.6.2 Manual Resynchronization

In some cases it might be useful or even neccessary to manually trigger resynchronization of a storage target or metadata server. One case, for example, is a storage system on the secondary target that is damaged beyond repair. In this scenario all data of that target might be lost and a new target needs to be brought up with the old target ID. The automatic resync won't be sufficient then, because it would only consider files after the last successful communication of the targets. Another case for a manual resync override is when a fsck of the underlying local file system (e.g. xfs_repair) has removed old files.

The beegfs-ctl tool can be used to manually set a storage target or metadata server to the needs resync state. Please note that this does not trigger a resync immediately, but does only inform the management daemon about the new state. The resync process then will be started by the primary of that buddy group a few moments later.

As said before, the primary target saves the time of the last successful communication with the secondary target. Without additional parameters, this timestamp will be used to shorten resynchronization times as much as possible. But it is also possible to override this timestamp to resynchronize a longer timespan or to resynchronize everything in the case described previously.

Please see the help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --startresync --help
```

If a resynchronization is already running and you want to abort it and start anew, you can do so by passing the –restart parameter to beegfs-ctl. If you don't, the current process keeps running and your request will be ignored. This is particularly useful if the system started an automatic resynchronization after a secondary target became reachable again, but you know that the timestamp-based approach is not sufficient. For example, this might be the case if your complete underlying filesystem broke before the secondary target was started, i.e. the target is completely empty and needs a full synchronization. Note that restarting a running resync is only possible for storage targets because metadata servers never do a partial resynchronization.

The following command could be used to stop the automatic resynchronization and start a full resynchronization instead.

```
$ beegfs-ctl --startresync --nodetype=storage --targetid=X --timestamp=0 --restart
```

2.6.3 Display Resynchronization Information

The beegfs-ctl command line tool can be used to display information on an ongoing resynchronization process.

Please see the built-in help of beegfs-ctl for more information on available parameters:

```
$ beegfs-ctl --resyncstats --help
```

2.7 Caveats of Storage Mirroring

Storage buddy mirroring provides protection against many failure modes of a distributed system, such as drives failing, servers failing, networks being unstable or failing, and a number of other modes. It does not provide perfect protection if a system is degraded, mostly only for the degraded part of the system. If any storage buddy group is in degraded state, another failure may cause data loss. Administrative actions can also cause data loss or corruption if the system is in an unstable or degraded state. These actions should be avoided if at all possible, for example by ensuring that no access to the system is possible while the actions are performed.

2.7.1 Setting states of active storage targets

When manually changing the state of a storage target from GOOD to NEEDS_RESYNC, clients accessing files during a period of propagation "see" different versions of the global state. This influences data and file locks. Propagation happens every 30 seconds, so the period will not take longer than a minute. This may happen because the state is not synchronously propagated to all clients, which makes the following sequence of events possible:

1. An administrator sets the state of an active storage target which is the secondary of a buddy group to NEEDS_RESYNC with beegfs-ctl –startresync.
2. The state is propagated to the primary of the buddy group. The primary will no longer forward written data to the secondary.
3. A client writes data to a file residing on the buddy group. The data is not forwarded to the secondary.
4. A different client reads data from the file. If the client attempts to read from the primary, no data loss occurs. If the client attempts to read from the secondary, which is possible without problems in a stable system, the client will receive stale data.

If the two clients in this example used the file system to communicate, eg by calling flock for the file they share, the second client will not see the expected data. Accesses to the file will only stop considering the secondary as a source once all clients have received the updated state information, which may take up to 30 seconds.

Setting the state of a primary storage target may exhibit the same effects. Setting states for targets that are currently GOOD, and by that triggering a switchover, must be avoided while clients are still able to access data on the target. Propagation of the switchover takes some time during which clients may attempt to access data on the target that was set to non-GOOD. If the access was a write, that write may be lost.

2.7.2 Fsync may fail without setting targets to NEEDS_RESYNC

When fsync is configured to propagate to the storage servers and trigger an fsync on the storage servers, an error during fsync may leave the system in an unpredictable state if the error occurred on the secondary of a buddy group. If the fsync operation failed on the secondary due to a disk error the error may be detected only during the next operation of the secondary. If a failover happens before the error is detected the automatic resync from the new primary (old secondary, which has failed) to the new secondary (old primary) may cause data loss.

CHAPTER 3

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.

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.

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.

CHAPTER 4

Cloud Integration

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

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.

CHAPTER 5

Striping

Striping in BeeGFS can be configured on a per-directory and per-file basis. Each directory has a specific stripe pattern configuration, which will be derived to new subdirectories and applied to any file created inside a directory. There are currently two basic parameters that can be configured for stripe patterns: the desired number of storage targets for each file and the chunk size (or block size) for each file stripe.

The stripe pattern parameters of BeeGFS can be configured with the Admon GUI or the command-line control tool. The command-line tool allows you to view or change the stripe pattern details of each file or directory in the file system at runtime.

The following command will show you the current stripe settings of your BeeGFS mount root directory (in this case "/mnt/beegfs"):

```
$ beegfs-ctl --getentryinfo /mnt/beegfs
```

Use mode setpattern to apply new striping settings to a directory (in this case "stripe files across 4 storage targets with a chunksize of 1 MB"):

```
$ beegfs-ctl --setpattern --numtargets=4 --chunksize=1m /mnt/beegfs
```

Stripe settings will be applied to new files, not to existing files in the directory. With time, as files are continuously overwritten, moved, copied, removed, and recreated, the new stripe pattern will gradually be applied to all files in the directory.

5.1 Buddy Mirroring

If you have buddy mirror groups defined in your system, you can set the stripe pattern to use buddy groups as stripe targets, instead of individual storage targets. In order to do that, add the option –buddymirror to the command, as follows. In this particular example, the data will be striped across 4 buddy groups with a chunk size of 1 MB.

```
$ beegfs-ctl --setpattern --numtargets=4 --chunksize=1m --buddymirror /mnt/beegfs
```

In BeeGFS version 7, this option has been replaced with –pattern=buddymirror.

5.2 Impact on network communication

The data chunk size has an impact on the communication between client and storage servers in several ways, as follows.

- When a process writes data on a file located on BeeGFS, the client identifies the storage targets that contain the data chunks that will be modified (by querying the metadata servers) and send modification messages to the storage servers containing the modified data. The maximum size of such messages is determined by the data chunk size of the file.

  If you define chunksize=1m, 1 MB will be the maximum size of each message. If the amount of data written to the file is larger than the maximum message size, more messages will have to be sent to the servers and this may cause performance loss. So, slightly increasing the chunk size to a few MB has the effect of reducing the amount of messages and this can have a positive performance impact, even in a system with a single target.

- In addition, it is important to make sure that a *data chunk fits the RDMA buffers* available on the client, in order to prevent the messages from being split, in order to be transmitted over RDMA.

- You also have to consider the file cache settings. When the client is using the buffered cache (tuneFileCacheType = buffered), it uses a file cache buffer of 512 KB to accumulate changes on the same data. This data is sent to the servers only when data from outside the boundaries of that buffer is needed by the client. So, the larger this buffer, the less communication will be needed between the client and the servers. You should set this buffer size to a multiple of the data chunk size. For example, adding tuneFileCacheBufSize = 2097152 to the BeeGFS client configuration file will raise the file cache buffer size to 2 MB.

CHAPTER 6

Client Tuning

6.1 Parallel Network Requests

- Each BeeGFS client establishes multiple network connections to the same server, which allows the client to have multiple network requests in flight to this server
  - The number of connections from a particular client to the same server can be configured by setting the value of connMaxInternodeNum in /etc/beegfs/beegfs-client.conf
- Increasing the number of connections may improve performance and responsiveness for certain workloads.
  - When increasing the value, it is extremely important keep the resulting RAM usage for network buffers on the servers in mind, especially for Infiniband and larger cluster setups. Make sure to read the comments for connMaxInternodeNum and connRDMABufSize in beegfs-client.conf to learn more about the server-side RAM usage.
  - On a compute node, it usually doesn't make sense to set this number higher than the number of CPU cores.
  - On a cluster login node, setting this value higher than the number of CPU cores may help to improve responsiveness when multiple users are active.
- BeeGFS clients establish connections only when they are needed (and drop them after some idle time). Use the command beegfs-net on a client to see the number of currently established connections to each of the servers.
  - beegfs-net is contained in the beegfs-utils package.
- The total space used by the buffers (connRDMABufSize x connRDMABufNum) should be larger or equal to the data chunk size, so that the messages exchanged between client and storage servers do not need to be split to fit into the buffers available. The default RDMA settings (connRDMABufSize = 64 KB, connRDMABufNum = 12) are OK for the default chunk size of 512 KB. If you set a chunk size of 1 MB and a buffer size of 64 KB, the number of buffers should be at least 1 MB / 64 KB + 4 additional buffers for protocol, so 20 in this example.

6.2 Remote fsync

- BeeGFS clients have a configuration option to control behavior when a user application calls fsync()

  - The option is called tuneRemoteFSync in /etc/beegfs/beegfs-client.conf
  - The client can either enforce that data is committed to the server disks on fsync() (=> tuneRemoteFSync=true) or only make sure that data is transferred to the server-side cache (=> tuneRemoteFSync=false).
- Disabling remote fsync can significantly reduce disk seeks and thus improves performance for applications that use a lot of fsync() calls.

6.3 Disable locate/mlocate/updatedb

- Some Linux distributions install a locate tool by default, which scans all file systems once per day to build a database of existing files.
- In a cluster, you certainly would not want to have all of your compute nodes scan the entire BeeGFS file system each day.
- Either deactivate this service if you don't need it or edit the file /etc/updatedb.conf to make sure that the "beegfs" file system type is contained in the "PRUNEFS" list and you BeeGFS mountpoint is contained in the PRUNEPATHS list.

CHAPTER 7

Getting started and typical Configurations

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.

CHAPTER 8

Installation and Setup

There are two ways to install BeeGFS: GUI-based (using a graphical Java interface) or manually (using shell commands).

The graphical installation is based on the BeeGFS Admon ("Administration and Monitoring") service, to which the graphical Java interface connects. In general, the GUI-based installation is recommended only for inexperienced users, because it does not provide the full flexibility of a manual installation (e.g. several configuration settings are not available and installation into an image is not supported by the GUI-based installation procedure).

If you are familiar with the Linux command line, it is recommended to perform the manual installation. Of course, you can still use the graphical interface to view usage statistics after a manual installation, if you want to.

8.1 General Notes

Installation Paths

: BeeGFS binaries and libraries will be installed to /opt/beegfs. The configuration files are located in the directory /etc/beegfs. Each service (including the client) comes with an init script in /etc/init.d.

Log Files

: BeeGFS services create log files in /var/log.

Runtime Compatibility of different Versions

: Different versions of BeeGFS clients and servers are compatible if they are part of the same major release series (e.g. all v6.x versions are part of the same major release v6).

Storage Format Compatibility

: Some new major releases of BeeGFS may introduce new storage format features, which might require upgrades of on-disk data structures. Upgrade tools will be provided in these cases to convert existing data to the new format in place. See release changelog for compatibility notes.

Compiler

The GNU C compiler (gcc) must be installed on client machines to build the BeeGFS client kernel modules.

Infiniband Libraries

For native Infiniband support, you need to have the ibverbs and rdmacm libraries of OFED 1.2 or higher installed. (Most Linux distributions also ship with sufficiently recent versions of these libraries.)

8.2 GUI-based Installation and Service Management

Graphical installation is performed through the BeeGFS Admon ("Administration and Monitoring"). It consists of a Java GUI and a beegfs-admon service, to which the graphical interface connects. The beegfs-admon service uses ssh to run installation commands on the other BeeGFS nodes. Thus, passwordless ssh login from the host running the beegfs-admon service to all BeeGFS nodes for user root is required (including the node where the beegfs-admon service is running).

Note: If you don't know how to configure ssh without password, you might want to have a look at the tool ssh-keygen to create a public/private key pair (without a passphrase) and then use ssh-copy-id to copy the new key to the other nodes, or use your favorite search engine to search for "passwordless ssh".

8.2.1 Download and Installation of the Admon service

Download the appropriate BeeGFS repository file for your Linux distribution from the table below.

- Admon is the administration and monitoring service of BeeGFS.
- The Admon service typically runs on a cluster master node.

Linux Base Distribution	Version	Package Manager	Repository File (Save to...)
Red Hat Linux (and derivatives, e.g. Fedora)	6.x	yum	Download (Save to: /etc/yum.repos.d/)
	7.x		

CHAPTER 9

BeeOND: BeeGFS On Demand

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.

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.

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.

CHAPTER 10

Admon

The BeeGFS Administration and Monitoring System (short: Admon) provides a graphical interface to perform administrative management tasks and to monitor the state of the file system and its components.

The BeeGFS Administration and Monitoring System consists of two parts:

- The Admon daemon, which can run on any machine with network access to the metadata and storage servers. This daemon gathers the status information of the other BeeGFS services and stores it in a database.
- The graphical Java-based client, which can run on your workstation. It connects to the remote Admon daemon via http.

Note: It is recommend to use Oracle Java Runtime Environment 7 (formerly known as Sun JRE 7) or higher to run the BeeGFS Admon GUI. Other Java runtime environments may work, but are not fully tested.

10.1 Installation and basic Setup

The Administration and Monitoring System for BeeGFS is contained in the optional beegfs-admon package.

The package is available either from the general BeeGFS repository or via direct download.

The package provides an init script to start the Admon daemon (/etc/init.d/beegfs-admon) and a configuration file (/etc/beegfs/beegfs-admon.conf).

Note: If you installed BeeGFS manually (i.e. not via the Admon GUI), you need to edit the beegfs-admon.conf file and set the parameter sysMgmtdHost to the hostname of your management server.

After installation, start the daemon:

```
$ /etc/init.d/beegfs-admon start
```

The graphical user interface for BeeGFS Admon comes packaged together with the Admon daemon and is located in /opt/beegfs/beegfs-admon-gui.

10.2 Admon GUI Start

If your BeegFS Admon daemon is not running yet, start it as described here: Admon Installation and basic setup

By default, TCP port 8000 will be opened by the daemon for HTTP connections.

To get the GUI, point your browser to http://Host_Where_The_Admon_Runs:8000 and download the jar-file from there.

The GUI can be started by double clicking from a file browser or by using the java -jar command (depending on your operating system and configuration). If you want to run the GUI from its default location on the Admon host, use the following command:

```
$ java -jar /opt/beegfs/beegfs-admon-gui/beegfs-admon-gui.jar
```

At first start, you will be prompted to provide the hostname and port (default: 8000) of the host on which the Admon daemon is running. It is also possible to change the resolution of the internal desktop of the GUI and the default log level of the GUI.

Note: It is recommend to use Oracle Java Runtime Environment 7 (formerly known as Sun JRE 7) or higher to run the BeeGFS Admon GUI. Other Java runtime environments may work, but are not fully tested.

10.3 Admon Login

The login mechanism is based on two predefined users.

The user “Information” (which has the initial password “information”) is only able to view statistics, whereas the user “Administrator” (which has the initial password “admin”) is also able to perform administrative tasks.

It is highly recommended that the first thing you do is to log in using the administrative account and change the predefined passwords.

A user with administrative privileges can also turn off the need for authentication for the informational user.

10.4 Admon Main Menu

The menu is a tree-like view on the left hand side and the associated windows will open on double-click. The following sections will give a short overview of the different items.

The menu is a tree-like view on the left hand side and the associated windows will open on double-click. The following sections will give a short overview of the different items.

10.4.1 Metadata Nodes

The menu item “Metadata Nodes” contains an overview page, as well as a dedicated page for each metadata node in the system.

The overview shows basic information, the status of all nodes and the total number of metadata requests in the system.

The page for a specific metadata node shows some general information on the node itself, the status of the node and the number of work requests to this node.

10.4.2 Storage Nodes

Like the meta nodes menu, the menu item "Storage Nodes" also consists of an overview page, as well as a dedicated page for each storage node in the system.

Values which can be retrieved on these pages include the general status information, as well as disk space usage and data throughput.

For disk performance, four values are displayed. While the read and write graphs are very exact (measured every second), they are also very erratic. The averaged graphs are better suited to identify a tendency. These graphs are always an average of the last 30 values. You can easily hide each throughput line by disabling the appropriate checkbox under the graphic.

Note that only the 10 min history view is based on the exact one second interval. The other history views are based on more coarse-grained (averaged) values.

10.4.3 Client statistics

The pages contains the client statistics for metadata operations (create, stat, ...) or the clients statistics for the storage operations (read, write, ...).

10.4.4 User statistics

The pages contains the user statistics for metadata operations (create, stat, ...) or the user statistics for the storage operations (read, write, ...).

10.4.5 Management

The management pages contain elements for administrative tasks. The page "Known Problems" is designed as a quick overview of the system's health. All problems related to the status of the nodes and their interconnection are listed here. The "Start/Stop Daemon" page allows start or stop all daemons and clients. The item "Log Files" opens a window which shows the log files of all daemons and clients.

10.4.6 FS Operations

The menu item "FS Operations"->"Stripe Settings" allows you to view and change the striping information in your file system. In BeeGFS, it is possible to define the chunk-size of data that will be written, as well as the number of storage targets, over which one file will typically be distributed. The corresponding information can be retrieved on this page. Furthermore, if you logged in with administrative privileges, the system will allow you to change these settings for each directory in the file system.

With the file browser you can browse through the global BeeGFS and retrieve information on the stored files. Please note, that although you are able to see directories and files, you will not be able to view the content.

10.4.7 Installation

The management pages contain elements for automation and simplification of the installation/uninstallation tasks. Please refer to the BeeGFS installation guide for a detailed description. Also the installation log file is available in this menu.

10.5 Admon Menu Bar

The menu bar contains options which are not required for the installation and the day by day administration.

10.5.1 Admon

The menu item "Change Settings" contains the configuration options of the GUI. Also the logout option and the close option for the GUI.

10.5.2 Administration

The options inside the menu item "User Settings" allow you to change the login passwords and to disable the password for the Information user. The effect of the latter is that users can view the web-frontend without being asked for a password. (The administrative account is not affected by this setting).

The menu item "Mail Settings" lets you define some values for e-Mail notifications by the software. If configured accordingly, an administrator can receive an e-Mail whenever a node in the system appears to be down. These pages are only accessible by the user "Administrator".

CHAPTER 11

BeeGFS APIs Overview

Besides the POSIX interface, BeeGFS provides other APIs to take more control of data placement, query additional information or access data through other interfaces.

1. The **'Striping API'_** allows the application developers to create files with individual stripe patterns, which are adjusted for the access pattern of a particular file. It also allows querying of stripe pattern details of files.
2. The **'Cache API'_** provides functions to copy data between a fast BeeGFS cache file system (typically a BeeOND instance) and a global BeeGFS. Prefetching and flushing of data to and from the cache file system can be done synchronously and asynchronously. The asynchronous prefetch/flush offloads the copy task to a cache daemon, which copies the data with multiple threads to speed up the data transfer.
3. With the **'Hadoop BeeGFS Connector'_**, BeeGFS can be used as an alternative to the Hadoop file system (HDFS). Hadoop applications can use the general Hadoop file system API to access BeeGFS.

CHAPTER 12

Contact Information

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

CHAPTER 13

Licensing

The BeeGFS client module is licensed under the *GPLv2 <http://www.gnu.org/licenses/gpl-2.0.html>*.

All other BeeGFS components are licensed under the *BeeGFS EULA <https://www.beegfs.io/docs/BeeGFS_EULA.txt>*.

13.1 Admon Licensing

The BeeGFS Administration and Monitoring System is licensed under the terms of the BeeGFS End User License Agreement.

The software makes use of the following third party libraries, which are all used unmodified:

- OpenSSL (http://www.openssl.org), which is licensed under the OpenSSL License (http://www.openssl.org/source/license.html).
- Sqlite (http://www.sqlite.org), which was given to the public domain by the authors.
- Mongoose (http://code.google.com/p/mongoose), which is licensed under MIT license (http://www.opensource.org/licenses/mit-license.php)
- TinyXML++ (http://code.google.com/p/ticpp), which is licensed under MIT license (http://www.opensource.org/licenses/mit-license.php)
- JChart2D (http://jchart2d.sourceforge.net), which is licensed under the GNU Lesser General Public License (http://www.gnu.org/copyleft/lesser.txt)