An introduction to BeeGFS®

Jan Heichler

November 2014 – v1.1

Abstract

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

About Fraunhofer

Fraunhofer is Europe’s largest application-oriented research organization. Fraunhofer is operating 66 institutes in Germany and offices in Europe, the USA and Asia. The institutes work in different fields such as health, energy, environment, communication, and information technology. Their research is focused on applied science, making technology easily applicable, and creating new products. This is done in close co-operation with universities and other research institutes, and typically as contract research for industry partners.

The Fraunhofer Institute for Industrial Mathematics (ITWM) in Kaiserslautern (Germany) has a focus on High Performance Computing (HPC) and provides the Fraunhofer Competence Center for HPC - also for other institutes.

About ThinkParQ

ThinkParQ is a Fraunhofer HPC spin off company, founded in late 2013 by some of the key people behind BeeGFS to provide professional support, services and consulting for BeeGFS customers. ThinkParQ is closely connected to Fraunhofer by an exclusive contract. The company is taking all efforts to enable the file system adoption in different markets. ThinkParQ is the first contact address for BeeGFS and connects end users to turn-key solution provides, organizes events, attends exhibitions & trade shows, and works very closely with system integrators.

All requests concerning BeeGFS should initially be addressed to ThinkParQ GmbH.

Content

1. History

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

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

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

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

2. Overview

In general, BeeGFS is a network file system. It allows clients to communicate with storage servers via network (anything with TCP/IP on it or RDMA-capable interconnects like InfiniBand, RoCE or Omni-Path with native verbs support). A BeeGFS solution could therefore be called a Network Attached Storage (NAS).

Furthermore, BeeGFS is a parallel file system – by adding more servers, the capacity and performance of them is aggregated in a single namespace. That way the filesystem performance and capacity can be scaled to the level which is required for the specific application. Furthermore, BeeGFS is splitting MetaData from ObjectData. The ObjectData is the data users want to store, whereas the MetaData is the "data about data", such as access rights and file size – but most important in the MetaData is the information, on which of the numerous storage servers the content of a file can be found. The moment a client has got the MetaData for a specific file or directory, it can talk directly to the ObjectDataServers to retrieve the information. There is no further involvement of the MetaDataServer (unless MetaData has to be changed).

The number of ObjectStorageServers as well as the number of MetaDataServers can be scaled. Therefor all aspects of performance requirements can be satisfied by scaling to the appropriate number of servers.

BeeGFS is targeted at everyone, who has a need for large and/or fast storage. Traditional fields are HighPerformance- and HighThroughputComputing as well as for storage of (large and growing) research data. The concept of scalability additionally allows users with a fast (but perhaps irregular) growth to adapt easily to the situation they are facing over time.

A very important differentiator for BeeGFS is the ease-of-use. This was one main design goal from the beginning. The philosophy behind BeeGFS is to make the hurdles as low as possible and allow the use of a parallel file system to as many people as possible for their work. We will explain how this is achieved in the following paragraphs.

The BeeGFS file system comes without license fee: It is a “free to use” product for end users – so whoever wants to try it for his own use, can download it from www.beegfs.com and use it. The client is published under GPL, and the server is covered by the BeeGFS EULA.

There is professional support available through ThinkParQ to enable customers to use the product in productive environments with defined response times for support. Also professional support allows system integrators to build solutions using BeeGFS for their customers.

BeeGFS is a Linux based file system – so all parts (clients, servers) run under Linux operating systems. There might be native support for other architectures in the future, but there hasn’t been an implementation so far. Since the client is GPL, everyone should feel encouraged to port it to a different operating system, if he sees any need for it.

3. General Architecture

BeeGFS needs four main components to work

- the ManagementServer
- the ObjectStorageServer
- the MetaDataServer
- the file system client

There are two supporting daemons

- The file system client needs a “helper-daemon” to run on the client
- The Admon daemon might run in the storage cluster and give system administrators a better view about what is going on. It is not a necessary component and a BeeGFS system is fully operable without it.

In the following we will describe the individual components in greater detail and explain, how the components work.

The term “servers” is used to refer to a Linux process running on a specific machine – not the machine itself. The difference is very important since many servers (meaning services) can run on the same physical machine.

An important design decision with regards to “ease of use” was the choice for the file systems used locally on the disks to store data. Other implementations of parallel file systems bring their own block layer file system – and this has shown to be a major hurdle when starting new into the world of parallel storage and when managing productive installations. One would have to learn a new way of managing block storage, of creating file systems and doing file system checks.

BeeGFS was designed to work with any local file system (such as e.g. ext4, xfs or zfs) for data storage, as long as it is POSIX compliant. That way system administrators can chose the local file system that they prefer and are experienced with and therefore do not need to learn the use of new tools. This is making it much easier to adopt the technology of parallel storage – especially for sites with limited resources and manpower.

BeeGFS can run on different hardware platforms (like x86, x86_64, ARM, OpenPower and more) and different flavors of Linux:

- RHEL/Scientific Linux/CentOS
- SuSE Linux Enterprise Server/SuSE Linux Enterprise Desktop/OpenSuSE
- Debian/Ubuntu

This is also making it easier for system administrators to use the software, as they can use the Linux distribution of their choice and do not need to adapt.

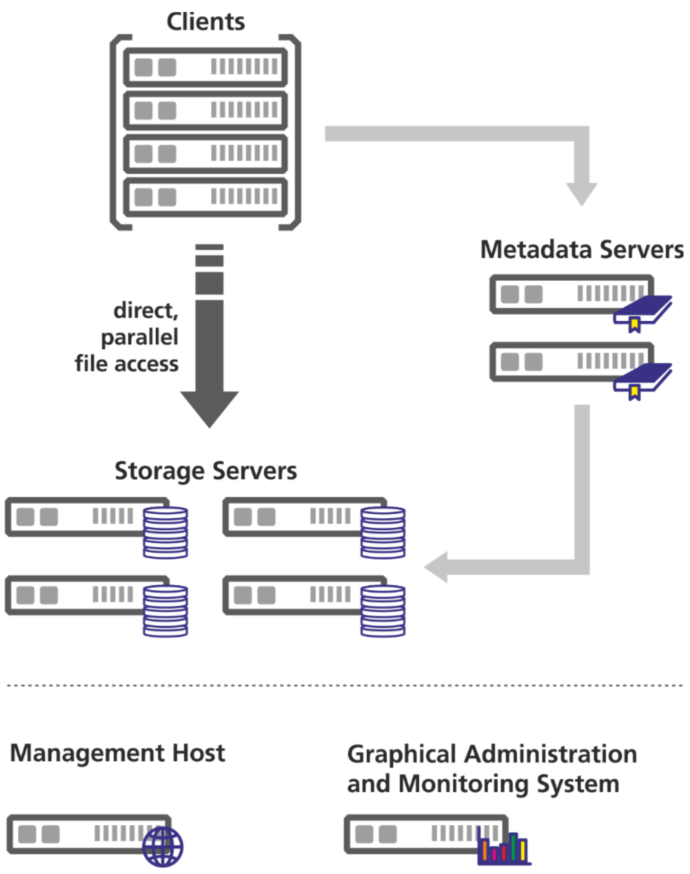
Clients
Metadata Servers
direct, parallel file access
Storage Servers
Management Host
Graphical Administration and Monitoring System

Figure 1: BeeGFS Architecture

4. The ManagementServer (MS)

The MS is the component of the file system making sure that all other processes can find each other. It is the first daemon that has to be set up – and all configuration files of a BeeGFS installation have to point to the same MS. There is always exactly one MS in a system – and the service itself is not critical since, if it fails, the system will continue to operate - but one would not be able to add new clients or servers until it is restored. Additionally, the MS is not performance critical as it is not involved in operations.

The MS maintains a list of all file system components – this includes clients, MetaDataServers, MetaDataTargets, StorageServers and StorageTargets. Additionally, the MS tags StorageTargets and MetaDataTargets with one of three labels – normal, low and critical. These tags influence the way targets are picked, when creating new files.

The labels are assigned based on the available free space on the target. The target chooser running on the MetaDataServer will prefer targets tagged normal as long as such targets are available – and it will not pick any target tagged critical before all targets entered that state.

By setting the values for the tagging in the MS config file one can influence the behavior of target picking.

5. The MetaDataServer (MDS)

The MDS contains the information about MetaData in the system. BeeGFS implements a fully scalable architecture for MetaData and allows a practically unlimited number of MDS. Each MDS has exactly one MetaDataTarget (MDT). The term MDT defines the specific storage device with the appropriate local file system to store MetaData of this MDS. The recommended choice for the file system on MDT is ext4 as it performs well with small files and small file operations. Additionally, BeeGFS makes use of storing information in extended attributes and directly in the inodes of the file system to optimize performance – both works well with ext4.

MDT are typically built from SSDs (or fast spinning disks). If there is a need for RAID protection, then RAID1 and RAID10 are appropriate RAID-Levels. It has been seen that RAID levels such as 5 and 6 are not a good choice, since the access pattern for MetaData is highly random and consists of small read and write operations. Picking a RAID-level, that is not performing well with these access patterns, will lead to reduced MetaData performance.

Each directory in the global file system is attached to exactly one MDS handling its content. So if the MetaData for directory A is handled by MDS #1 then MetaData for files residing in A is stored on #1. If a subdirectory B is created in A then a new MDS is picked randomly from the number of available MDS to store B (and the files to be created in bar).

Since the assignment of directories to MDS is random BeeGFS can make efficient use of a (very) large number of MDS. As long as the number of directories is significantly larger than the number of MDS (which is normally the case), there will be a roughly equal share of load on each of the MDS.

There is one exception in assignment: The root level directory is always to be found on MDS #1.

That way there is a defined entry point into the directory tree. The top level directory has links to the MDS holding the information about subdirectories. With this information a client can walk the directory tree and find the MDS responsible for a specific directory.

The moment this information is found all MetaData operations (such as file create or file stat) can be executed without involving the other MDS. That allows scaling of performance for MetaData operations.

The MDS is a fully multithreaded daemon. The number of threads to start has to be set to appropriate values for the used hardware. Factors to consider here are

- *number of CPU cores in the MDS* – handling MetaData requests is creating some load on the CPU-cores – therefore having a number of them in the MDS is a good choice. However: Serving these requests is also IO-intense, which creates delays while waiting for the disks to supply the data. This makes it logical to start more threads then CPU-cores are available. The number of MDS-threads might exceed the number of CPU cores by a factor of 4 or 8.

  From a performance perspective the clock speed of the cores is influencing the performance characteristics of the MDS. To serve a small number of requests a high clock is more important than a large number of cores as it decreases the latency. For systems that have to deal with a very large number of requests the number of cores becomes more important as it increases the throughput.

- *number of clients* – the larger the number of clients in the system the more requests can be made to the MDS(s) – and the more threads can be efficiently used to serve them.

- *workload* – depending on how (and how often) a workload makes requests to a MetaDataServer, a different number of threads will perform better (or worse).

- *number and type of disks forming the MDT* – The most important factor is the performance and performance behavior of the MDT. With the scaling of the number of MDS-threads one is scaling the possible IO-requests being made to the MDT. Since these requests are random, the performance is correlating with the random-IOPS the MDT can sustain.

  SSDs can sustain a very high number of random IOPS compared to mechanical drives and are highly recommended for MetaData. The difference in comparison to a mechanical disk (even with 15k RPM) can be a factor of 100 and more.

  Additionally, SSDs can lower the latency for individual requests significantly which helps interactive use.

So in general serving MetaData needs some CPU power – and especially, if one wants to make use of fast underlying storage like SSDs, the CPUs should not be too light to avoid becoming the bottleneck.

There is no perfect choice for the number of threads that works for everyone equally well, but having too many threads is a waste of resources (RAM in particular) and might impact performance negatively because of many task switches. Having too little threads on the other hand limits your performance for sure. So in case of doubt go for the larger number of threads.

6. The ObjectStorageServer (OSS)

The OSS is the main service to store the file contents. Each OSS might have one or many ObjectStorageTargets (OST) – where an OST is a RAID-Set (or LUN) with a local file system (such as xfs, ext4 or zfs) on top.

A typical OST is between 6 and 12 hard drives in RAID6 – so an OSS with 36 drives might be organized with 3 OSTs each being a RAID6 with 12 drives.

The OSS is a user space daemon that is started on the appropriate machine. It will work with any local file system that is POSIX compliant and supported by the Linux distribution. The underlying file system may be picked according to the workload – or personal preference and experience. Like the MDS, the OSS is a fully multithreaded daemon as well – and so as with the MDS, the choice of the number of threads has to be made with the underlying hardware in mind.

The most important factor for the number of OSS-threads is the performance and number of the OSTs that OSS is serving. In contradiction to the MDS, the traffic on the OSTs should be (and normally is) to a large extend sequential. Since the number of OSS-threads influences the number of requests that can be put on the disks, increasing the number of threads includes the risk to randomize traffic, which is seen by the disks unnecessarily.

One of the key functionalities of BeeGFS is called striping. In a BeeGFS file system each directory has (amongst others) two very important properties defining how files in these directories are handled.

numtargets defines the number of targets a file is created across. So if “4” is selected each file gets four OSTs assigned to where the data of that file is stored to.

chunksize on the other hand specifies, how much data is stored on one assigned target before the client moves over to the next target.

Let’s assume that the setting of “numtargets” is 4 and “chunksize” set to 1MB. When a file is now created (e. g.) ost01, ost02, ost03 and ost04 are selected (in that order). When the user starts filling the file with data the first 1MB gets stored on ost01 – after that the 2nd MByte is stored on ost02 – the 3rd on ost03 and the 4th on ost04. The 5th MByte then gets stored on ost01 again – and so on.

The goal of this striping of files is to improve the performance for the single file as well as the available capacity. Assuming that a target is 30 TB in size and can move data with 500 MB/s, then a file across four targets can grow until 120TB and be accessed with 2 GB/s. Striping will allow you to store more and store it faster!

7. The file system client

BeeGFS comes with a native client that runs in Linux – it is a kernel module that has to be compiled to match the used kernel. The client is an open-source product made available under GPL. The client has to be installed on all hosts that should access BeeGFS with maximum performance.

The client needs two services to start:

beegfs-helperd: This daemon provides certain auxiliary functionalities for the beegfs-client. The helperd does not require any additional configuration – it is just accessed by the beegfs-client running on the same host. The helperd is mainly doing DNS for the client kernel module and provides the capability to write the logfile.

beegfs-client: This service loads the client kernel module – if necessary it will (re-)compile the kernel module. The recompile is done using an automatic build process that is started when (for example) the kernel version changes. This helps to make the file system easy to use since, after applying a (minor or major) kernel update, the BeeGFS client will still start up after a reboot, re-compile and the file system will be available.

Compared to other products available on the market this eliminates certain challenges for daily operation. One of the biggest is that system administrators have to check, if there is the right version of the file system client available matching their new kernel and at the same time works with the storage servers. This has to be done before applying patches and increase the administrative overhead.

When the client is loaded, it will mount the file systems defined in beegfs-mounts.conf. Please note that BeeGFS mountpoints are usually not defined in */etc/fstab*. If necessary mounts can be defined *in /etc/fstab,* but it will disable the autobuild mechanism.

8. Getting started and typical configurations

To get started with BeeGFS, all it needs is to download the packages from www.beegfs.com and a Linux machine to install them on. Basically you can start doing it with a single machine and a single drive to try it.

(Probably) the smallest reasonable production system is a single machine with two SSDs (or fast spinning disks) in RAID1 to store the MetaData and a couple of hard drives (or SSDs) in RAID5 or RAID6 to store ObjectData. Fast network (such as 10GE or InfiniBand) is beneficial but not necessary. From this building block one can grow capacity and performance by adding disks and/or more machines.

Typical configurations work with full blown enterprise servers containing between 12 and 72 hard disk drives in several RAID6-Sets. As network usually InfiniBand or 40GE is used to bring the performance to the clients. MetaData is either stored on the same machines as ObjectData or on special machines.

9. Contact information

If you want to be updated about news and updates around BeeGFS in general, then you can follow @BeeGFS on twitter or see www.beegfs.com/news. If you want to become part of the BeeGFS community, you can also join the BeeGFS user mailing list by subscribing at http://www.beegfs.com/support. On the mailing list, users can ask other users for help or discuss interesting topics.

If you are interested in more information, help building your first BeeGFS storage system or becoming a system integrator who can offer turn-key solutions using BeeGFS, you can contact us at:

info@thinkparq.com