Evaluating the MetaData Performance of BeeGFS®

Jan Heichler

May 2015 – v1.2

Abstract

Scope of this paper is to evaluate the MetaData performance of BeeGFS. We work with three different scenarios. In the first one we compare the performance of different backend filesystems to store the BeeGFS MetaData in, using the following performance metrics: file create rate, file read rate and file remove rate. The filesystems we are using are ext4, xfs and btrfs and we are running the tests using one singleHDD and one single SSD as backend.

Second, we investigate how BeeGFS scales when we scale the number of MetaData servers in a BeeGFS storage cluster. Since BeeGFS has a fully distributed approach for Object- and MetaData there is the frequent question of how this scales and what needs to be done to get a certain performance. Our goal is not to evaluate the hardware itself but to evaluate the architecture of the filesystem. We are focusing on two key performance indicators: file create rate and file stat rate. There will be two different sets of tests. The first set of tests evaluates how different MetaData targets behave while the second set of tests shows how performance can be scaled by adding more MetaDataServers.

Third, we are going to evaluate how a system of a fixed size will behave with different numbers of clients and different file sizes.

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. The hardware platforms to be used

The first scenario was benchmarked on a system consisting of a single storage server with:

- two Intel® Xeon® CPU E5-2650v2 CPUs with a clockspeed of 2.60GHz and 8 Cores each (so 16 in total) – Hyperthreading was enabled,
- 64 GB of DDRIII-memory,
- 1 x SATA II SSD "SSDSC2MH250A2" by Intel®,
- 1 x SAS 15k HDD drive, and
- Mellanox ConnectX III InfiniBand Card with a single FDR x4 connection to the InfiniBand switch (56 Gbps).

As clients we use four nodes with each:

- one Intel® Xeon® CPU E3-1230 v3 CPU with a clockspeed of 3.30GHz,
- 32 GB of DDRIII-memory, and
- Mellanox ConnectX III InfiniBand Card with a single FDR x4 connection to the InfiniBand switch (56 Gbps)

All benchmarks for the second and third scenario were taken on a system consisting of 40 servers. Each server was equipped with:

- two Intel® Xeon® E5-2680v2 CPUs with a clockspeed of 2.8 GHz and 10 Cores each (so 20 in the server) – HyperThreading was enabled,
- 64 GB of DDRIII-memory,
- 4 x SATA III SSD (10 servers with 1TB per SSD – 30 with 256GB per SSD) connected to a SAS2 backplane and controlled individually by a Dell PERC H310 controller, and
- Mellanox ConnectX III InfiniBand Card with a single FDR x4 connection to the InfiniBand switch (56 Gbps).

For the tests we are going to use (up to 20 of) these servers as MetaData- or ObjectDataServers and the remaining nodes as clients. The exact setup is described in the individual sections.

The hardware used in these tests differs due to varying availability of hardware in the test lab and the quantity of it. This is important when comparing the absolute results of the individual benchmarks. Additionally the tests were executed independently of each other and therefor the used benchmarks differ in a few details.

2. How to measure MetaData performance

To evaluate the MetaData performance we are going to use an open source tool called mdtest. Mdtest runs a (configurable) set of tests on files and directories. The tool is programmed using MPI and can be run with a large number of processes on a large number of nodes. Therefor it is a perfect fit for our use case, as it can simulate a large number of clients accessing the parallel filesystem.

Measuring MetaData performance is generally not trivial. There are many more parameters that might influence performance tremendously then when measuring sequential bandwidth or even random IOPS.

In the individual tests we are going to work with different number of processes and different number of files and directories. The settings are described at the beginning of each section in detail and we explain why we made certain choices.

3. 1st scenario: Comparing different backend filesystems

The BeeGFS MetaDataServer is using an existing local Linux filesystem to store its own data. This allows users to pick whatever filesystem they have best knowledge about. Still, it leaves the question what performance impact this has.

We are going to compare ext4, xfs and btrfs as underlying filesystem on both, a single fast spinning HDD and a single SSD. We picked these as they are the default choices of current Linux distributions and are widely used. This sums up to a total of six individual tests that we will compare in the following section.

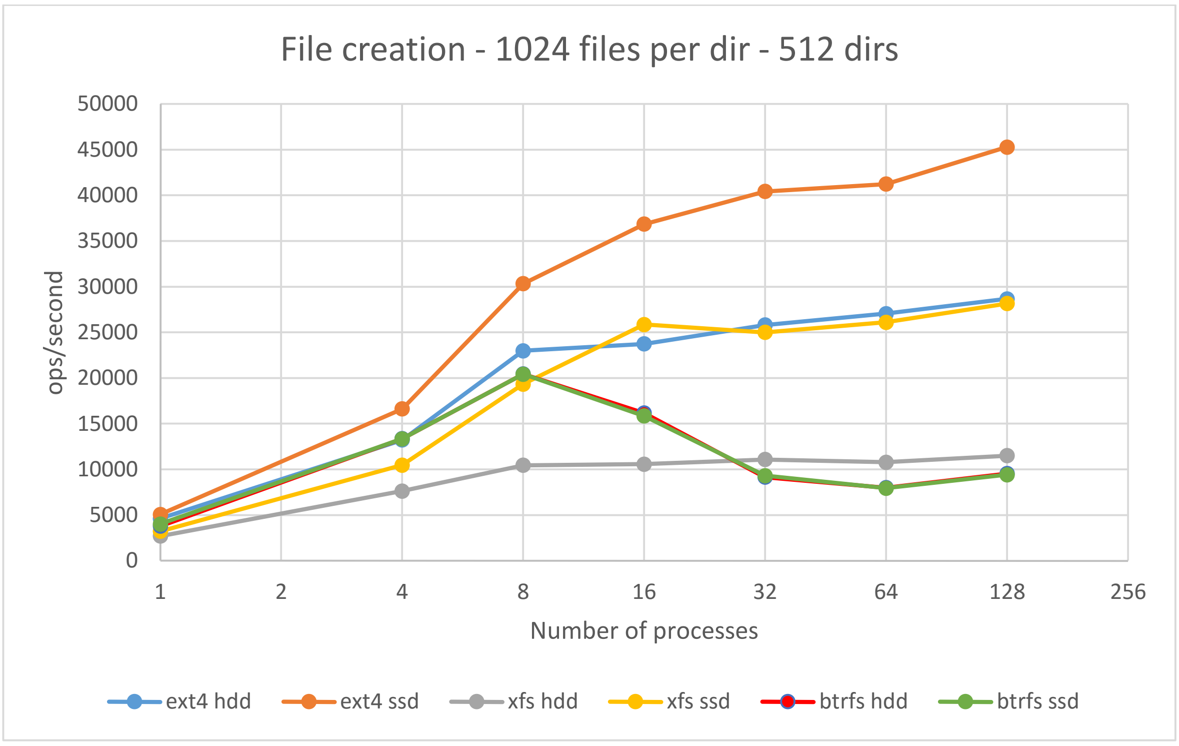
As a test we will use mdtest with the following commandline

```
mpirun -np <no_of_clients> mdtest -d /beegfs/ -i 2 -I <no_of_files>

-z 1 -b <no_of_dirs> -L -u -F
```

Where `no_of_clients` is scaled from 1 to 128 and `no_of_dirs` is calculated by dividing 512 through the *no_of_clients* to create a constant number of directories/files in the tests. The `no_of_files` was fixed to 1024 in the individual tests. For a full MPI startup you will have to add your hostnames or a machinefile in an appropriate way for your MPI version used.

`-d`	the directory to create files and directories in
`-i 2`	run the whole test 2 times – we will use the “mean” result of the output – which is the average over the two 2 runs – since the standard deviation was relatively small
`-I <no>`	defines the number of files to be created by each process.
`-z 1`	each process will create a directory structure 1 levels deep
`-b <no>`	each process will create a directory structure.
`-u`	each process will work in its own subdirectory
`-L`	files are just created at the leaf-level of the directory tree each process creates
`-F`	this just enables the file orientated tests of mdtest to be performed



First, let's have a look at the results of the three filesystems when running on HDD. For lower process counts (1, 4 and 8) ext4 and btrfs don't perform much differently. Both scale up to a create rate of 20k to 22k creates/s. xfs on the other hand shows that it scales much less than the other two – and also has a smaller absolute create rate (~10k creates/s).

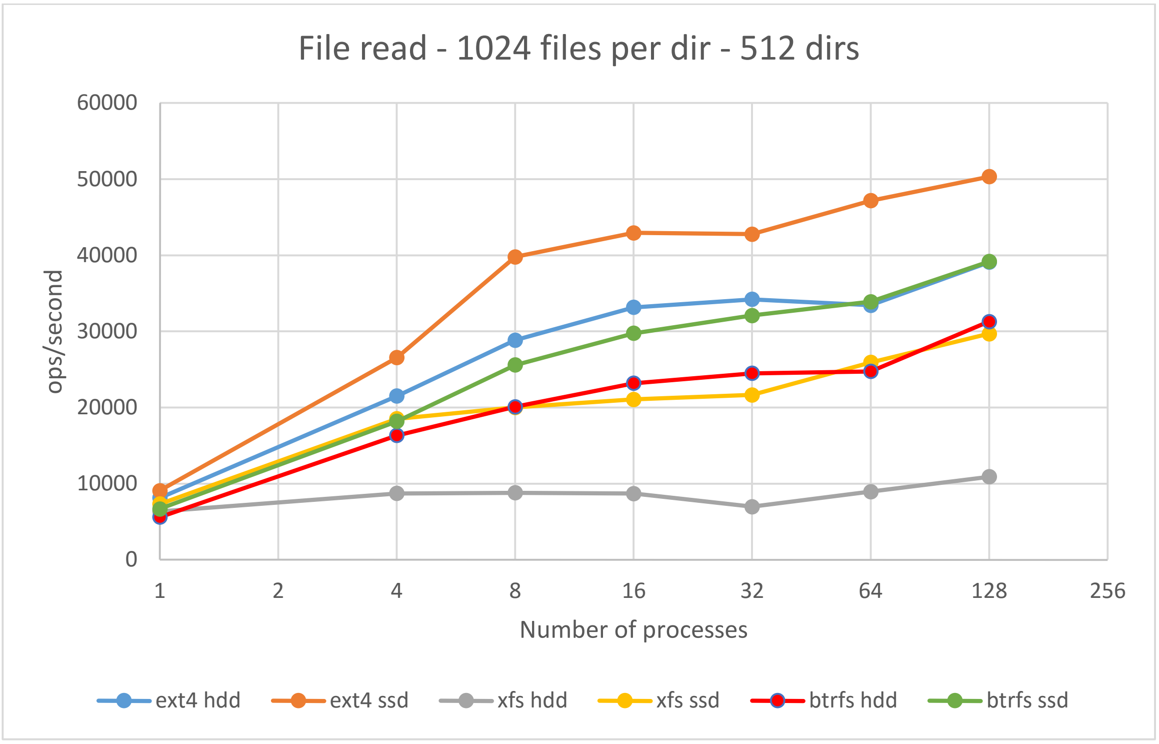
Scaling the number of processes further creates a larger gap between btrfs and ext4. While ext4 can be pushed up to a create rate of close to 30k creates/s, btrfs breaks down in performance when more and more clients create files concurrently – independent of HDD or SSD being used. Please note that the dark-blue line for "btrfs hdd" is barely visible behind the green line for "btrfs ssd" – the light-blue line represents "ext4 hdd".

xfs on the other hand stays at the performance level that it has shown already for 8 clients and delivers a rather constant 10.5 to 11k creates/s

Looking at the scenario with SSDs we notice two important things:

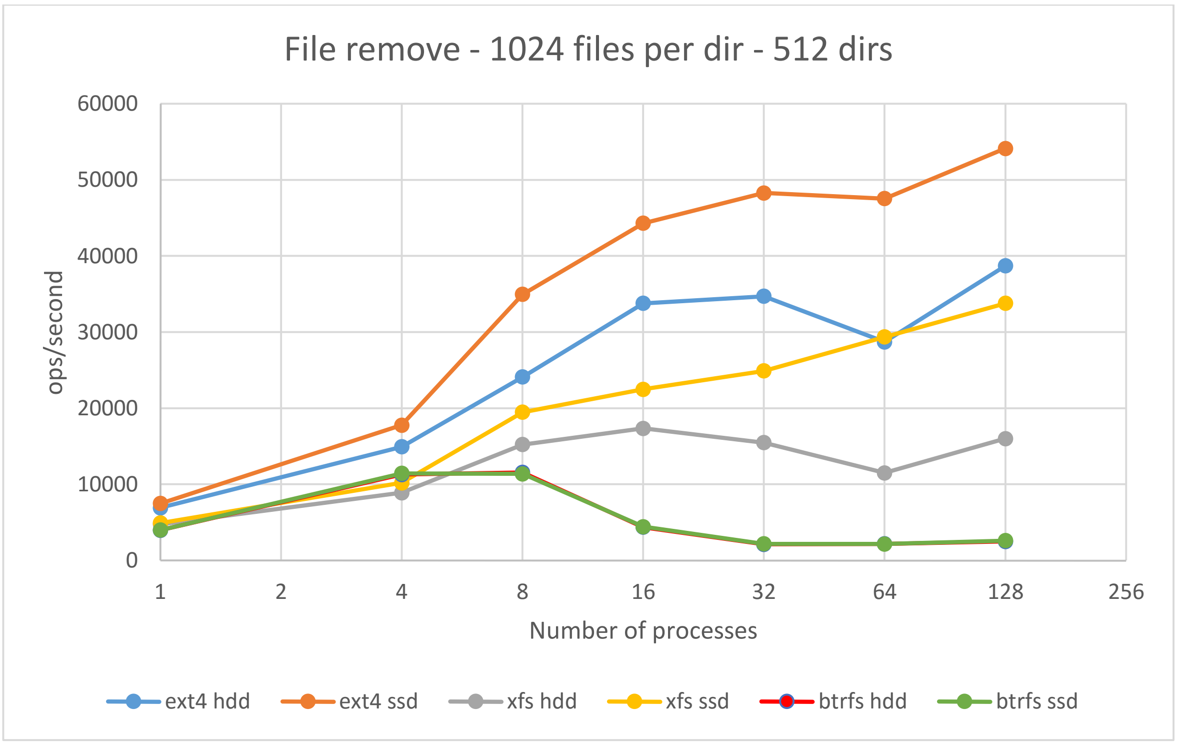
btrfs doesn't seem to deliver the advantage in performance of the SSD to the test results – it basically behaves the same running from SSD and running from HDD. The reason for this is unclear.

The second thing that can be seen is that both ext4 and xfs performance gets boosted while running the filesystem on SSD. xfs can be pushed to close to 30k creates/s, which is nearly triple of what was possible on HDD while ext4 scales up to 45k creates/s which is 60% more than before.



For file read operations the result is quite similar to file create for ext4 and xfs. xfs stays on a relatively stable level of performance when running from HDD and gets boosted the most by using a SSD as storage. ext4 shows a solid performance from HDD and can be pushed further when running from a SSD.

Btrfs behaves differently than for the create test. Its performance doesn't lower with an increasing number of processes and we see a clear difference between HDD and SSD as backend storage device.



Our last test is deleting the created files again and behaves quite similar to the create test. Btrfs shows a bad performance for an increasing number of processes and there is no added performance running from SSD. xfs and ext4 behave roughly as before.

The conclusion of the measurements is that ext4 is the best choice if one aims for best MetaData performance. Additionally, SSDs can boost the performance of MetaData operations by a margin. Especially for mixed workloads in real world scenarios SSDs have shown a far greater advantage than can be shown in the microbenchmarks done for this paper. With this findings the use of SSDs for MetaData is strongly recommended.

4. 2nd scenario: Scaling of MetaData performance

In this scenario we are going to check how far we can push a single server by adding more SSDs for storing MetaData and we are going to scale the number of MetaDataServers up to 20 machines. The goal is to show how BeeGFS scales and what absolute performance can be achieved.

4.1 The MetaData performance of a single server

The first set of tests is to evaluate the performance of a single server. We are starting with using the 4 SSDs in the server in a single RAID0 configuration to build a very fast MetaDataTarget (MDT). A single MetaDataServer (MDS) is running on the machine and it will use 160 worker threads spread across the 20 physical cores. The number of files will be kept constant and the number of client processes will be scaled.

The commandline used is

```
mpirun -np <no_of_clients> mdtest -d /mnt/beegfs/ -i 3 -I <no_of_files> -z 2
 -b 8 -L -u -F
```

Where *no_of_files* is calculated to generate a total of 840960 files. For a full MPI startup you will have to add your hostnames or a machinefile in an appropriate way for your MPI version used.

In detail the options used are:

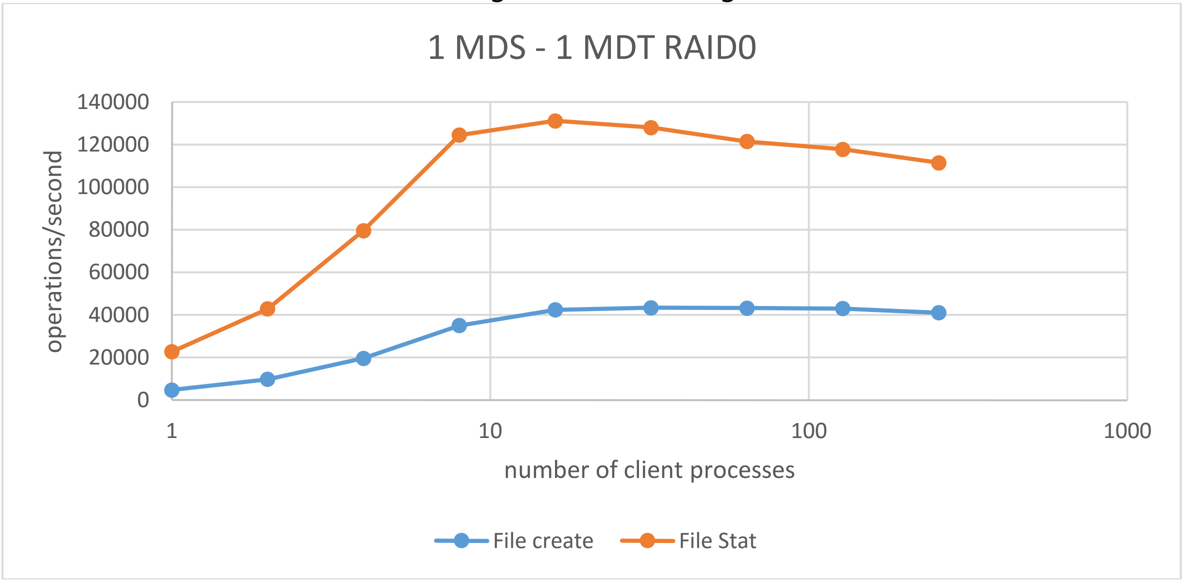
`-d`	the directory to create files and directories in
`-i 3`	run the whole test 3 times – we will use the “mean” result of the output – which is the average over the 3 runs
`-I <no>`	defines the number of files to be created by each process. The number is set to a value that the total number of files across all processes reaches 840960.
`-z`	each process will create a directory structure 2 levels deep

`-b`	each process will create a directory structure 8 directories wide
`-u`	each process will work in its own subdirectory
`-L`	files are just created at the leaf-level of the directory tree each process creates
`-F`	this just enables the file orientated tests of mdtest to be performed

The above is creating a lot of files in a lot of directories. Each process is creating 16 directories and is working in its own subdirectory. Since we scale from 1 to 256 Clients we will have 256 x 16 = 4096 subdirectories for the largest test. Since BeeGFS MetaData is distributed across different MDS by directory, performance will scale with the number of directories that clients use in parallel.

The files that we create are empty. We did want to test the pure performance of the MDS and any other file size than 0 bytes would involve the ObjectStorageTargets (OSTs) in the operation of creation. We will show in the 3rd scenario how the system behaves with different file sizes created. Also we don't evaluate file remove performance as this is also involving the ObjectStorageServer (OSS) and means one is moving from evaluating the pure MDS performance to evaluating a certain system.

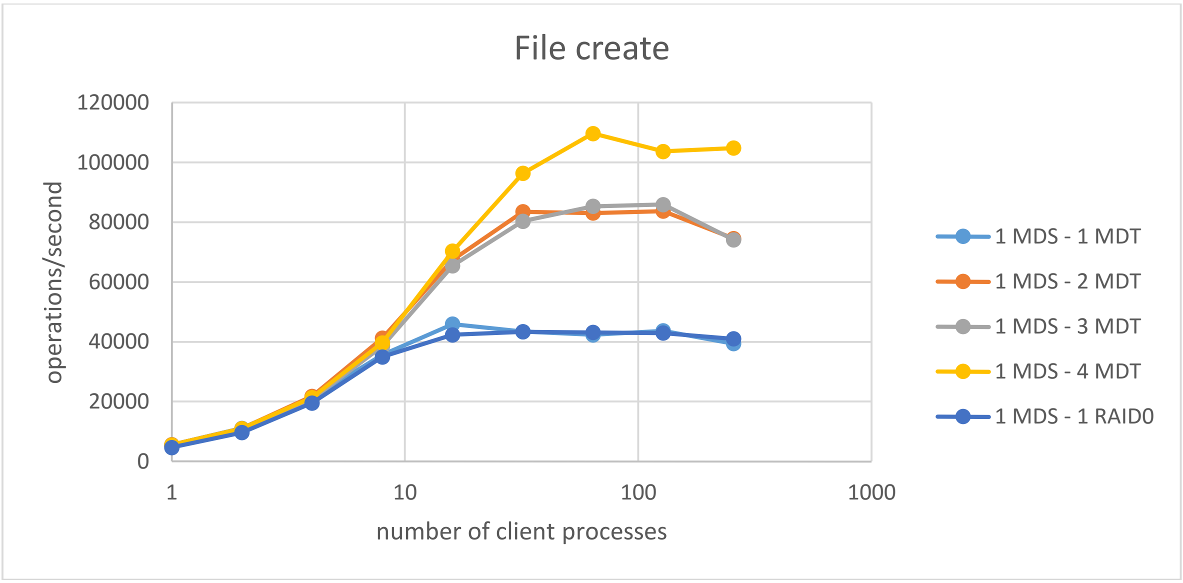
For the first test the four SSDs in a single MDS are configured as a (software) RAID0.

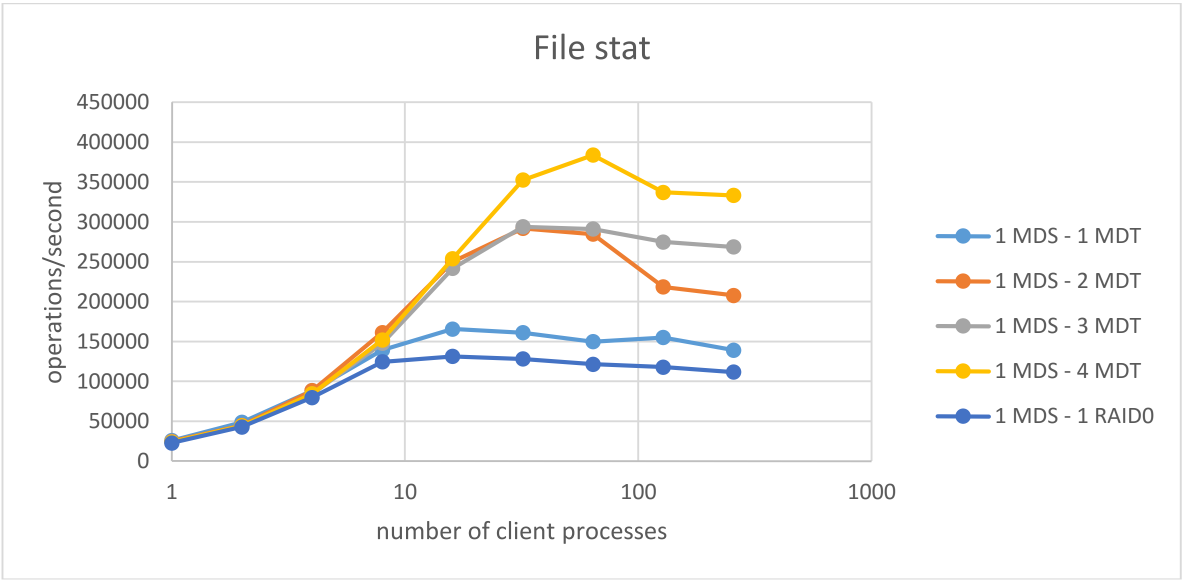


The performance figures show that a single client process can't max out the capability of the single MDS with a single (fast) MDT. When scaling the number of processes the maximum performance is reached with 16 client processes and a single server can deliver around 130,000 file stat operations and around 42,000 file create operations.

During the tests the performance of the local RAID0 built from 4 SSDs was monitored and the observation was that the SSDs are not performing as fast as they could: their utilization was not at 100%. To further drive utilization, different chunksizes on the RAID were tested but didn't improve the situation. Additionally a different number on WorkerThreads for the MDS were tried but also didn't increase the performance significantly.

In the following test we tried to use the capability of BeeGFS to run several servers on the same machine. Since the MDS can just have exactly one MDT we will start up to four MDS concurrently on the same physical machine which each MDS using a single SSD as MDT. When for the single RAID0-target we chose 160 worker threads we will now use 40 worker threads per MDS-instance. Additionally, we will bind the threads of each instance to a one of our two available CPU-sockets. So server instance #1 and #3 are going to be pinned to socket 0 – and #2 and #4 are pinned to socket 1. That way we prevent threads moving around between different NUMA zones in the server





The results show basically the same performance when using a single MDS with a single SSD as MDT as it did with the RAID0 as MDT. For file stat the single SSD as MDT performs even better than the RAID0.

When we scale towards two MDS instances we see nearly double the performance of a single instance but adding the third instance doesn't provide more performance. With adding the fourth MDS instance we max out performance to nearly 120,000 file creates and 400,000 file stat operations.

The reason for the bad performance of the RAID0 setup couldn't be determined fully. It is suspected either in the Linux RAID-layer that makes sub-optimal calls to the SSDs or the single instance of ext4 doesn't scale for the high amount of IOPS that is needed. For daily operation the conclusion from this is that one has to watch the performance characteristics closely and also that a single MDS instance can't be sped up by scaling the hardware up. It is another good reason for a scale out concept.

For the next tests we will stick with the 4 MDS instances on a single physical machine. We will refer to this setup as "a MDS with four MDT" and start to scale out the number of machines. We will increase the number of worker threads per MDS to 80 up to a total of 320 threads. So we will overload the 20 cores/40 hardware threads of the physical machine by a factor of 16/8. This is not unusual as there are typically way more requests made to the MDS than there are physical cores. Since each request has to wait for the underlying MDT to fulfil it there is enough compute power available to increase the throughput.

4.2 Scaling the number of MetaDataServers

In the next test we will scale the number of MetaDataServers from 1 to 20. Each physical server will run 4 MDS instances, so actually we will scale from 4 to 80 MDS. The reason for that has been explained in the paragraph before. We will refer to a "MDS with four MDT" in the following meaning the physical machine with four BeeGFS MDS instances using a single SSD each as MDT.

Since we have shown that the performance that can be achieved depends on the number of client processes, we will scale the number of client processes with the number of MetaDataServers. So for each MDS we will use 32 client processes to max out performance. Additionally, we will create a larger number of files. We will ensure that per MDT at least one million files are created.

The commandline used is

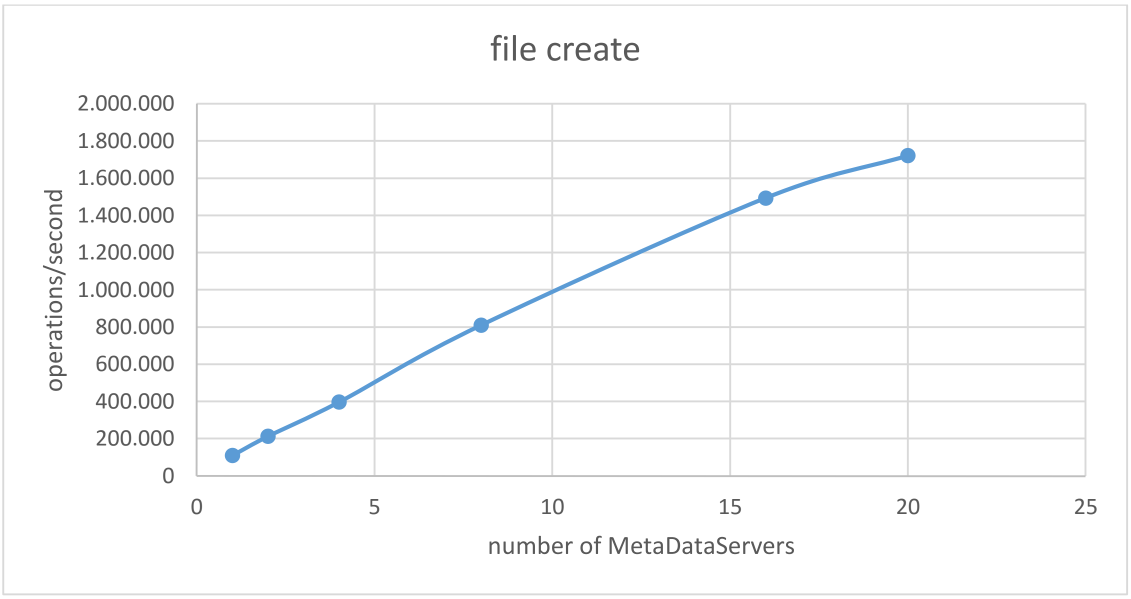
```
mpirun –np <no_of_clients> mdtest -d /mnt/beegfs/ -i 3 -I 6000 -z 2 -b 4 -L -u -F
```

For a full MPI startup you will have to add your hostnames or a machinefile in a appropriate way for your MPI version used. The number of files per 32 clients is 4032000 which means that each MDS with four MDT gets that amount of files.

In detail the options used are:

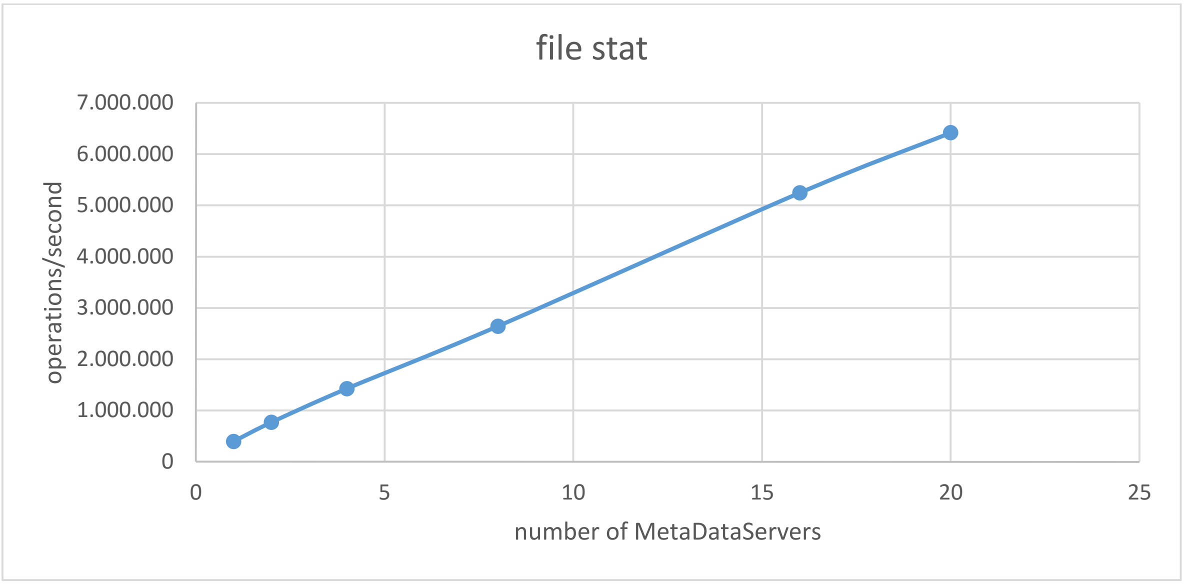
-d	the directory to create files and directories in
-i 3	run the whole test 3 times – we will use the "mean" result of the output – which is the average over the 3 runs
-I 6000	defines the number of files to be created by each process. So in each directory there will be 6000 files to be created.
-z	each process will create a directory structure 2 levels deep
-b	each process will create a directory structure 4 directories wide
-u	each process will work in its own subdirectory
-L	files are just created at the leaf-level of the directory tree each process creates
-F	this just enables the file orientated tests of mdtest to be performed

We reduced the total number of directories created by a single client process from 64 to 16 here (2 directories deep – 4 directories wide). So with 32 clients per MDS we will create 512 directories. Scaling it up to 20 MDS and 640 clients that equals to a total of 10240 directories – this seems to be an awful lot but considering the fact that a system with 20 MDS and the performance shown next means that there is a really large number of clients being present and running concurrent workloads.



Starting with a single MDS with four MDT delivering ~110,000 file creates, we scale to 20 MDS with 80 MDT and a total of 1,720,799 creates. This shows that the performance scales by a factor 15.7 when scaling up by a factor of 20. The efficiency therefor is 78.5%.

Looking at the file stats, the picture is quite similar. Starting with a single MDS with four MDT doing 395,000 stat operations and getting to 20 MDS with 80 MDT and a total of 6,419,445 stat operations. This equals to a speedup of 16.3 and an efficiency of 81.3%



Both tests show that the performance for MetaData operations increases tremendously when adding more MDS (and more MDT) to the system. It proves that the distributed MetaData approach of BeeGFS works and is able to sustain a performance that wasn't possible a couple of years back. Additionally, we have seen that BeeGFS can make effective use of SSDs and deliver their performance to the clients.

5. 3rd scenario: Working on a filesystem with fixed size

In the paragraphs before we have evaluated the performance of a single MDS and found that it is more efficient to use four individual MDTs than a single large one. We have also seen that scaling the number of MDS adds up performance to your parallel storage.

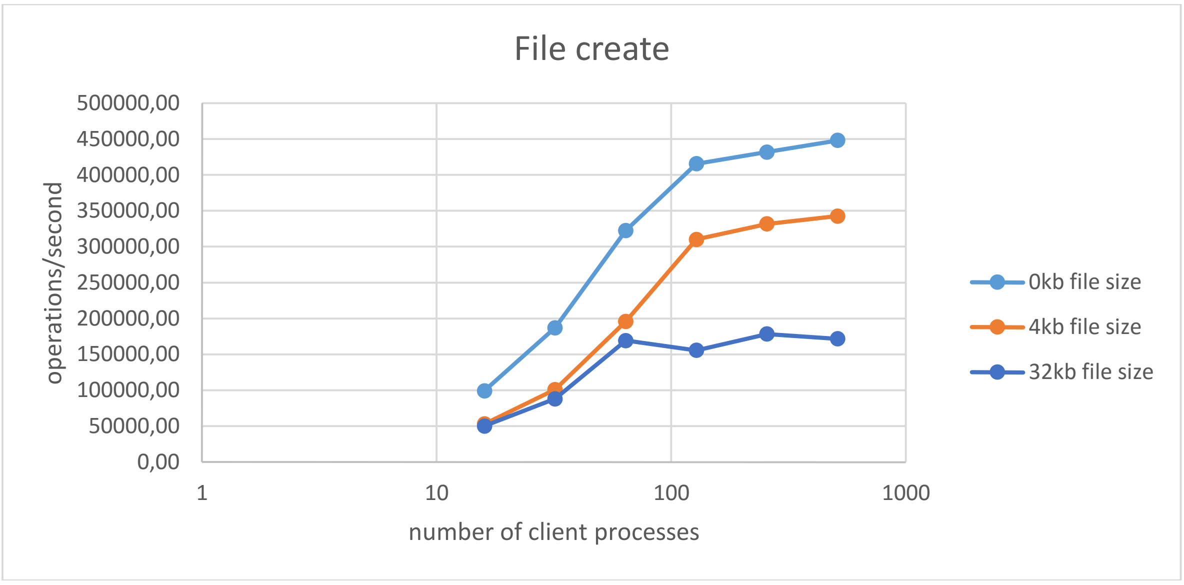
In this paragraph we want to build a system that is more "real world" and evaluate how it behaves. We will use four MDS (with 16 MDT) and 16 OSS with 16 OST. The OSTs will consist of the four SSDs in a RAID0 configuration. Please note that our storage targets have a performance characteristic that differs from normal hard drives. On the other hand we are running from a total of just 64 SSDs and do not make use of any sophisticated RAID controller with large DRAM cache.

For this setup we will

- run file creates with 0kb, 4kb and 32kb file size
- test file stat performance & file remove performance
- scale number of clients creating traffic
- scale number of files

With the evaluation done so far we can set our expectations for the test system in this paragraph. We have shown that a server can deliver about 100,000 creates and about 400,000 stat operations according to our results. We could therefore expect that our test system does (close to) 400,000 creates and (close to) 1,600,000 stats.

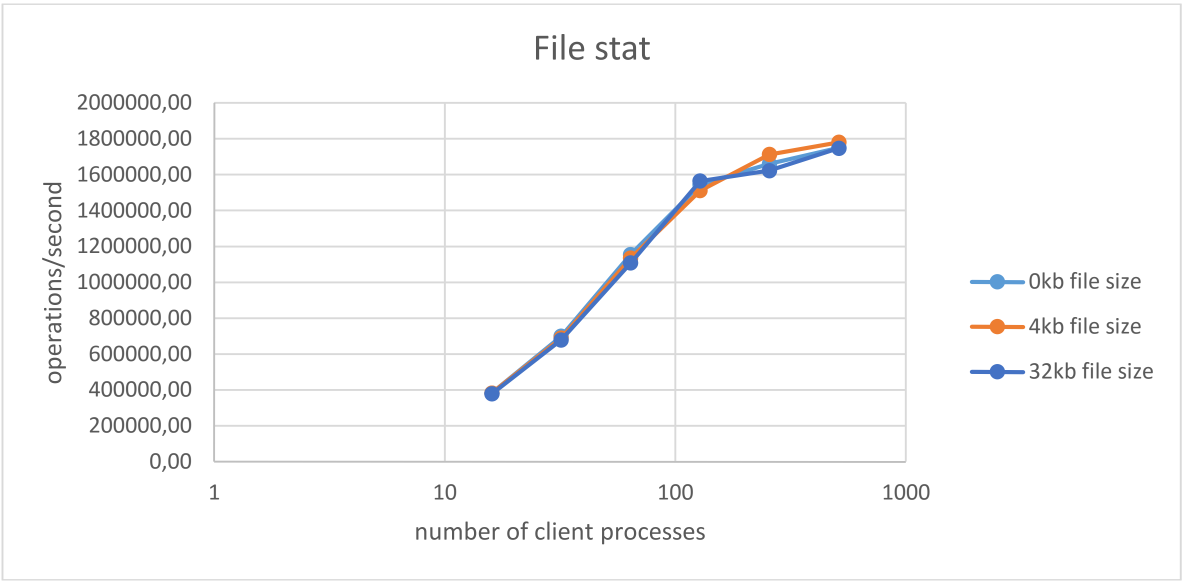
The first aspect we will evaluate is file create performance when doing 0k, 4kb and 32kb files. We will do this in combination with the scaling of the number of client processes and we will keep the number of files per client process constant (meaning that we will create an increasing number of files when increasing the number of clients).



The first obvious result is that our expectation of about 400,000 file creates for 0k filesize is exceeded when we increase the number of clients. With 512 client processes the system was able to do about 10% better.

The second (and not unexpected) observation is that when one is creating a 4kb file the OSTs are involved and more messages get send around. This creates a gap in performance that is (roughly) 50% when working with 16 client processes and closes to ~22% for 512 client processes. So when working with a large number of clients the system still delivers 78% of the "0kb file creates".

For 32kb file size the create rate doesn't increase anymore when getting above 64 clients. The reason for that is that the performance of the OSTs is limiting performance here. This is the point where a larger storage system would be able to scale further. With ~160,000 files created – each 32k in size – one is transferring data at about 5 GB/s, not considering the MetaData that is also transferred and the roundtrip times for the messages that have to be sent. To get to an even higher 32kb file create rate one would have to add more or faster OSTs at this point.



For file stat the picture changes quite a bit: The performance does scale again above the expected point when running with more than 128 processes. The four servers deliver close to 1,800,000 stat operations when using 512 client processes.

The most important conclusion is that file stat operations are completely independent of the file size. This shows that the design of BeeGFS is really efficiently differentiating between MetaData and actual ObjectData. Both can be treated completely separate, which is beneficial for real-world-scenarios since a user doing a "ls –l" in a directory doesn't interfere with a compute job pulling many GBytes of data from the storage servers at the same time.

6. Conclusion

In this document we have compared different Linux file systems to find the best choice for storing BeeGFS MetaData. The test results clearly show that using ext4 has significant advantages for operations such as file create. Additionally, we have shown that using a SSD instead of a mechanical hard disk improves performance further.

In the second scenario we have shown that scaling the number of MetaDataServers improves performance significantly. While going from one to 20 servers we scaled the performance from 110k ops/s to 1,72 mio ops/s when doing file creates and from 395k ops/s to 6,4 mio ops/s for stat operations. This proves that the method of distributing MetaData that is implemented in BeeGFS works and allows to scale the MetaData component of a parallel storage system to the level that is needed to sustain the performance for the users.

While the first and second scenario tried to measure the performance of the MetaData component isolated from the rest of the storage system by working with 0-byte files, the third scenario evaluates a system of a fixed size for 3 different file sizes. It shows that also for this scenario BeeGFS can scale MetaData performance to a high level. The most important result of the measurements is that the split from Meta- and ObjectData in BeeGFS works very well and allows to execute stat operations without involving the ObjectStorageServers at all. This aspect is very important when thinking about real-world usage of parallel filesystems where interactive users execute commands like “ls –l” and (highly) parallel compute jobs are pulling large files at several GiB/s sequential transfer speed.

7. Contact information

If you want to be updated about news and updates around BeeGFS in general then you can follow BeeGFS on twitter (www.twitter.com/BeeGFS). 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