



# Picking the right number of targets per server for BeeGFS®

Jan Heichler

March 2015 – v1.3

## Abstract

In this paper we will show the performance of two different configurations using the same hardware. Since the question is often asked how RAID-sets on a given hardware should be configured in order to achieve maximum performance we will compare a server with 48 disks divided into four targets of 12 disks in RAID6 each to a server running 48 disks in a single RAID60.

We will scale the number of client processes accessing the storage and reading/writing sequentially on the BeeGFS file system from 1 to 128.

## 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.

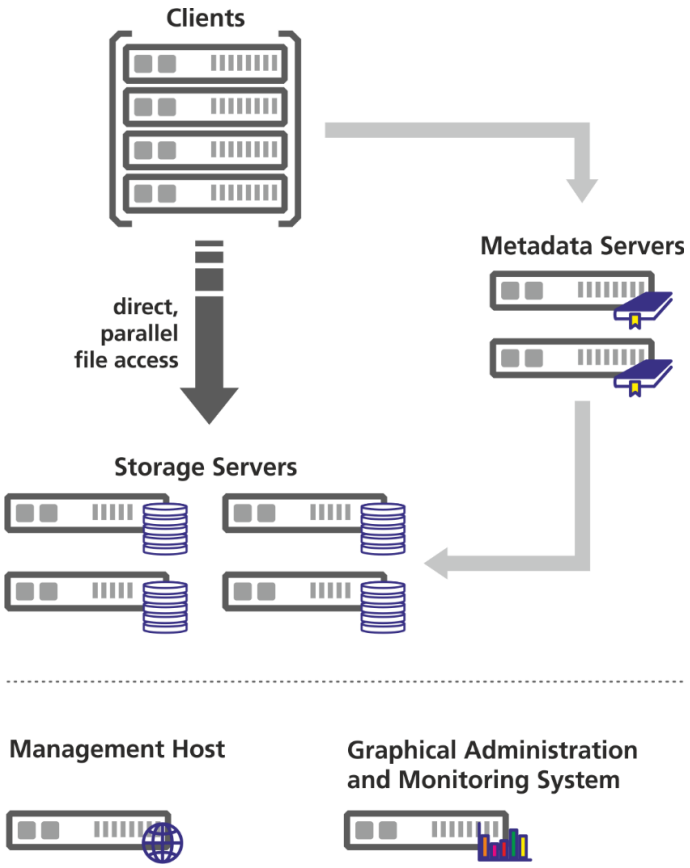
# Content

# 1. Introduction

BeeGFS is a distributed file system that can aggregate the capacity and performance of many storage servers into a single namespace. It is optimized for clients accessing this name space in a concurrent matter to achieve excellent throughput.

A single BeeGFS storage server can export from one to many so called “targets” to store files from clients. Files can be striped across multiple targets to improve the performance when accessing that particular file.

For a more detailed introduction on how BeeGFS works we recommend reading our paper “An introduction to BeeGFS”.




A central question when designing a BeeGFS storage solution is how many targets a server should export and how these targets should be organized.

In general, BeeGFS expects reliable targets – meaning targets that are RAID-protected. So exporting single disks as targets is not recommended. When building RAID-sets, the most common choice is using RAID6 or RAID60 because it combines the protection against double drive failures with a moderate overhead in storage capacity.

We want to answer the question whether it is better to use a larger number of smaller individual targets or if it is preferable to create large RAID60-arrays and have therefor fewer (but possibly faster) targets.

In this document we will use the following terms:

| | |
|---|---|
| chunksize: | the amount of data stored on a single disk in a RAID-array before the next disk is used. |
| full-stripe: | the amount of data stored on all data-disks in a RAID-array stores – this explicitly excludes data needed for parity – before the next stripe is written. |
| RAID6 N+M: | this refers to a RAID-array using the equivalent of N drives for data – and the equivalent of M drives for parity information |
| RAID60 4x(8+2) | a RAID60 configuration that stripes data across 4 RAID6 with each RAID6 having a 8+2 configuration |
| numtargets: | defines the number of targets in BeeGFS that are used to store a single file |
| BeeGFS chunksize: | the chunksize used by BeeGFS when striping across two or more targets |

Additionally we will use the proper Si-units for giving capacities and speeds – meaning: MiB, GiB, kiB etc.

## 2. The hardware platform to be used

We will use a single storage server with the following hardware configuration:

- Two Intel® Xeon® CPU E5-2650v2 CPUs with a clock speed of 2.60GHz and 8 Cores each (so 16 in total), Hyperthreading is enabled
- 64 GB of DDRIII-memory
- 1 x SAS 15k HDD drive for MetaData
- Mellanox ConnectX III InfiniBand Card with a single FDR x4 connection to the InfiniBand switch (56 Gbps)
- LSI RAID-Card MegaRAID 9380-8e
- 48 disks (2TB, 7200RPM, SAS2) in 4 external 12-bay JBODs connected to LSI RAID-card

The LSI MegaRAID controller has two external SAS3-ports with 4 lanes each. Each of the ports connects to 2 JBODs (daisy chained), since the JBODs did just support SAS2 the link to the JBOD ran in SAS2 mode (4 lanes per link)

Each JBOD contains 12 drives connected to a SAS2 backplane. The drives itself were spinning with 7200 RPM and had a SAS2 interface.

As clients we use four nodes with each

- One Intel® Xeon® CPU E3-1230 v3 CPU with a clock speed of 3.30GHz
- 32 GB of DDRIII-memory

- Mellanox ConnectX III InfiniBand Card with a single FDR x4 connection to the InfiniBand switch (56 Gbps)

## 3. Hardware and software configuration

All tests ran using a Scientific Linux 6.5 with a kernel version 2.6.32-431.17.1.el6.x86_64.

The 48 disks in the storage server will be configured in three ways for testing

1) Four arrays with 12 disks each – so all disks in one JBOD are part of the same array. We will pick RAID6 as a RAID-level since this is the most used RAID-level in production systems. As a chunksize per disk 256kiB will be set – this provides a balance between good sequential throughput and moderate read-modify-write overhead. It is a setting many BeeGFS setups in production use. The full-stripe of a array in this setup is 2560kiB

2) Eight arrays with 6 disks each (4+2) – so all disks in one JBOD are part of the two arrays. Using the same chunksize of 256kiB that will lead to a full stripe of 1024kiB.

3) We will build a single RAID60 array from all 48 drives – the structure that is used puts 12 drives in a RAID6 and let the RAID-controller stripe across all four of them. Again we will use a chunksize of 256kiB per disk. The full-stripe of a RAID6 is in this case 2560kiB, the full-stripe of the whole RAID60 is 10240 kiB.

For all the block devices in Linux we will apply the following settings:

*echo deadline > /sys/block/<devicename>/queue/scheduler*
*echo 2048 > /sys/block/<devicename>/queue/nr_requests*
*echo 32768 > /sys/block/<devicename>/queue/read_ahead_kb*

As the underlying local Linux file system we will use a properly aligned xfs on the block devices.

Configuration 1) *mkfs.xfs –d su=256kb,sw=10 /dev/<devicename>*
Configuration 2) *mkfs.xfs –d su=256kb,sw=4 /dev/<devicename>*
Configuration 3) *mkfs.xfs –d su=256kb,sw=40 /dev/<devicename>*

As mount options we used consistently

*noatime,logbufs=8,logbsize=256k,largeio,inode64,swalloc,allocsize=131072k,nobarrier*

From a BeeGFS perspective we will set the following things for all tests:

In fhgfs-client.conf

*tuneNumWorkers* = *32*

In fhgfs-storage.conf

*tuneWorkerBufSize* = *4m*
*tuneFileReadSize* = *256k*
*tuneFileReadAheadTriggerSize* = *4m*
*tuneFileReadAheadSize* = *16m*
*tuneFileWriteSize* = *2048k*
*tuneFileWriteSyncSize* = *0m*

In fhgfs-meta.conf

*tuneNumWorkers* = *64*
*tuneTargetChooser* = *roundrobin*

If not explicitly stated otherwise, we will work with default settings subsequently.

## 4. How to measure performance

We will focus on sequential performance in our tests and execute read and write. To scale the number of processes we will use a MPI coordinated storage benchmark named IOR. MPI is used to start the processes that read and write the data across the four available client nodes.

For IOR we will use the following command line:

*mpirun /cm/shared/apps/ior/current/src/ior -r -w -t2m -F -g –b<no>g -i 2 -o /beegfs/test*

*-r* run read tests

*-w* run write tests

*-t2m* transfer size of the application is set to 2 MiB

*-F* each process reads and writes his own file

*-g* use barriers between open, write/read and close

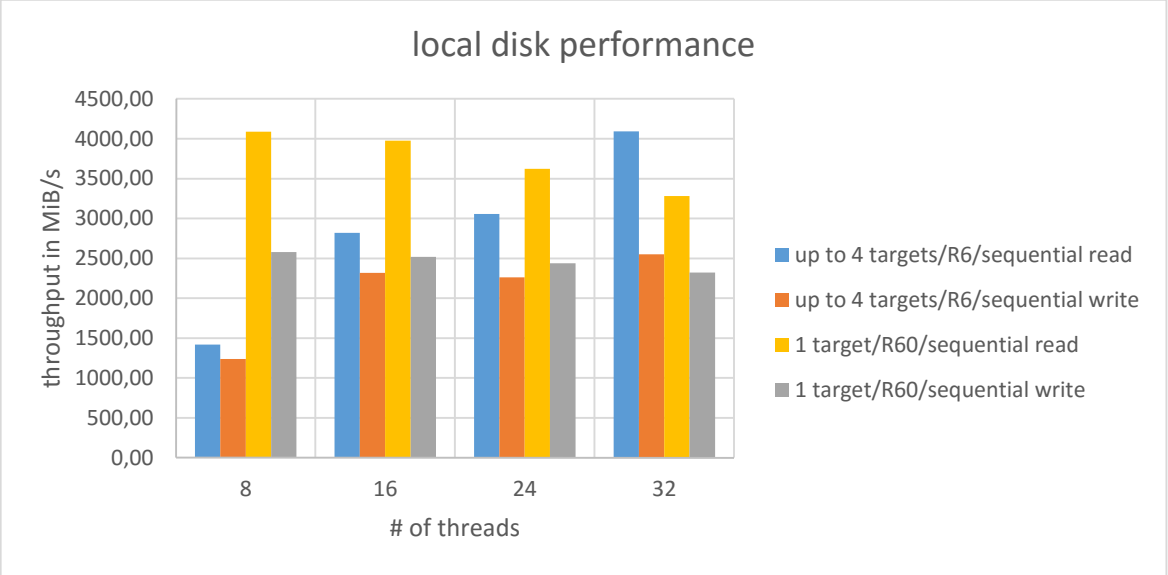
*-b <no>* sets the amount of data that each process reads/writes

*-i 2* run tests two times – we will use the mean value – the standard deviation was very low so two consecutive runs are sufficient

*-o <dir>* directory to be used for the tests

## 5. Test results

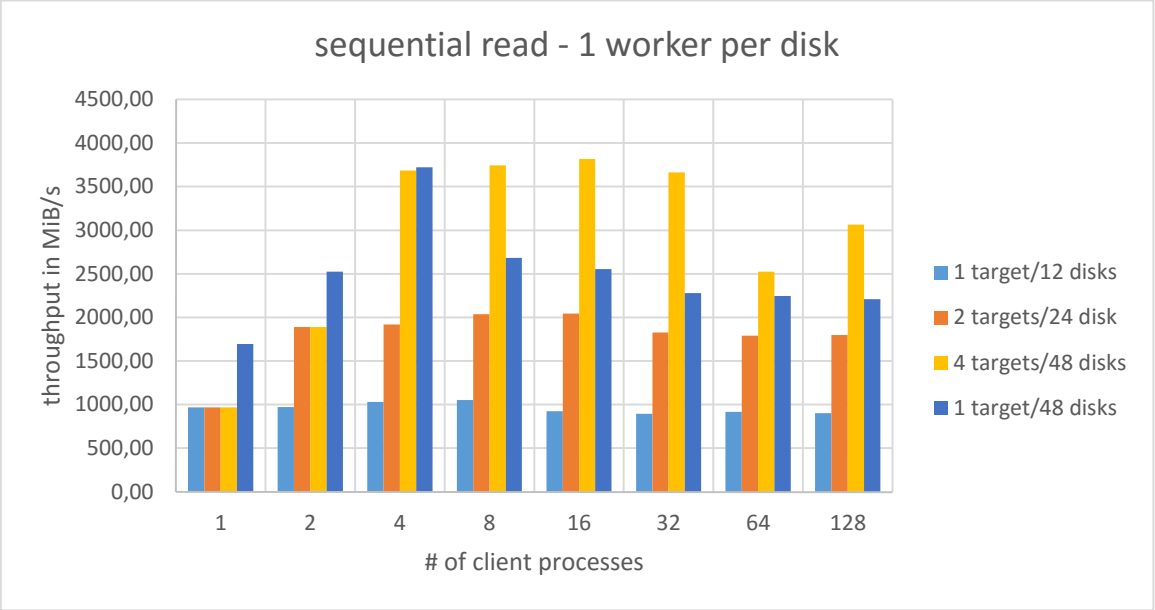
Before running the actual BeeGFS tests we want to evaluate what the hardware itself is capable of doing. Therefore, we will run local iozone tests with a number of local threads reading and writing to the disk arrays.




For the test with "up to 4 targets" 8 threads work on a single target, 16 threads work on 2 targets (8 assigned per target) etc.

The R60 target performs best when accessing it with 8 or 16 threads – and read performance decreases a bit with higher worker count. For the individual targets the write performance doesn't scale much from 2 to 3 to 4 targets, the controller limits the performance to about 2.5 GiB/s writes. The reads scale to slightly above 4 GiB/s

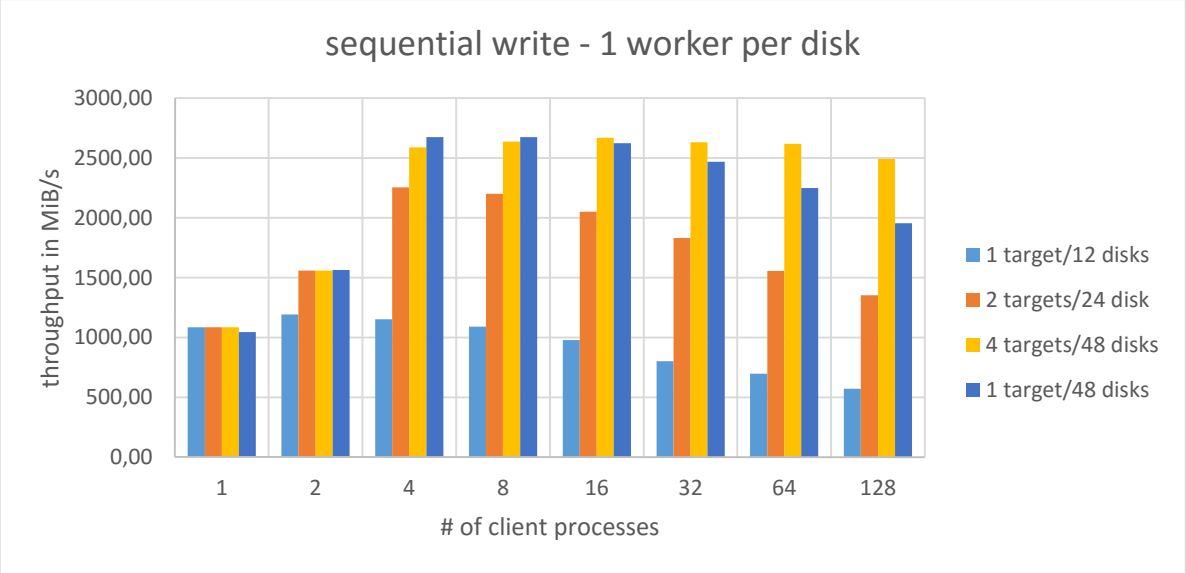
The first BeeGFS test compares the configuration with 4 RAID6 arrays in 10+2 configuration with a single RAID60 in 4 x (10+2) configuration. We will execute sequential reads and writes with a number of client processes ranging from 1 to 128. For the first test we will use one worker thread in the BeeGFS storage server per disk. So for a test with 1 target 10+2 we would start 12 WorkerThreads, for a test with 3 targets 10+2 we would start 36 WorkerThreads. Files are not striped across several targets but just stored on one.




For the tests with 1, 2, and 4 individual RAID6 targets we see that the performance peaks between 8 and 32 processes to a maximum – and reduces a bit when going to 64 and 128 processes due to the higher number of concurrent accesses to the same amount of disk drives

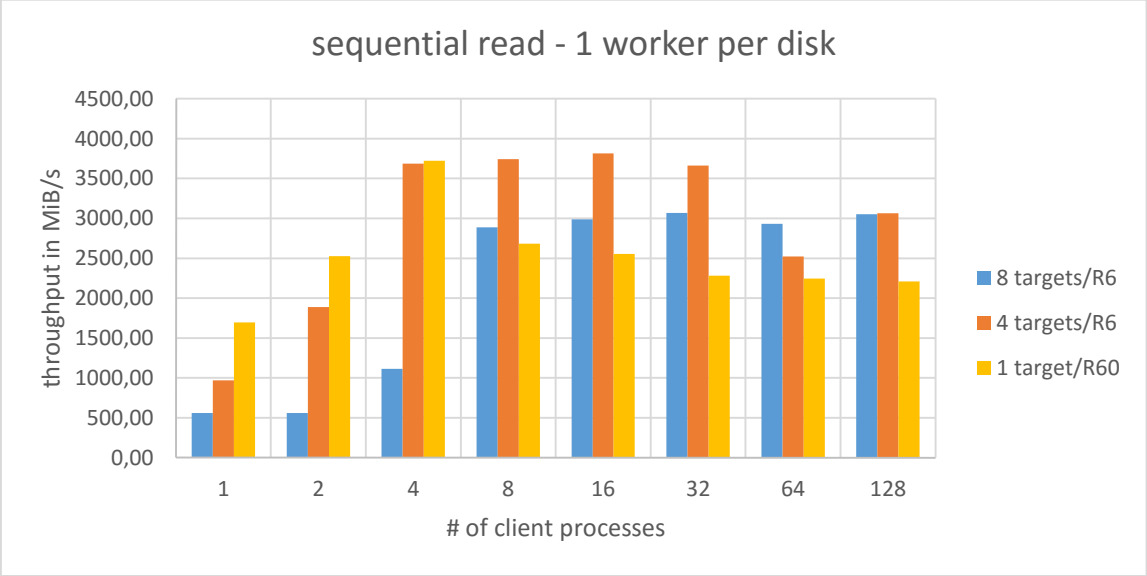
The individual RAID60 target performs a bit differently: whereas it outperforms the configurations with individual targets when running 1 and 2 concurrent processes it gets to a draw when running 4. The results for 1 and 2 are quite logical – the individually faster RAID60 can be used by 1 and 2 threads while 4 individual targets can't be used by just 1 or 2 processes when BeeGFS file striping is disabled.

For 4 client processes the RAID60-target performs equally fast as the 4 individual RAID6 targets, but when scaling the number of processes further the 4 individual targets stay at a high level of performance (basically matching what the hardware is capable of delivering), the RAID60 configuration cripples down and performs significantly worse

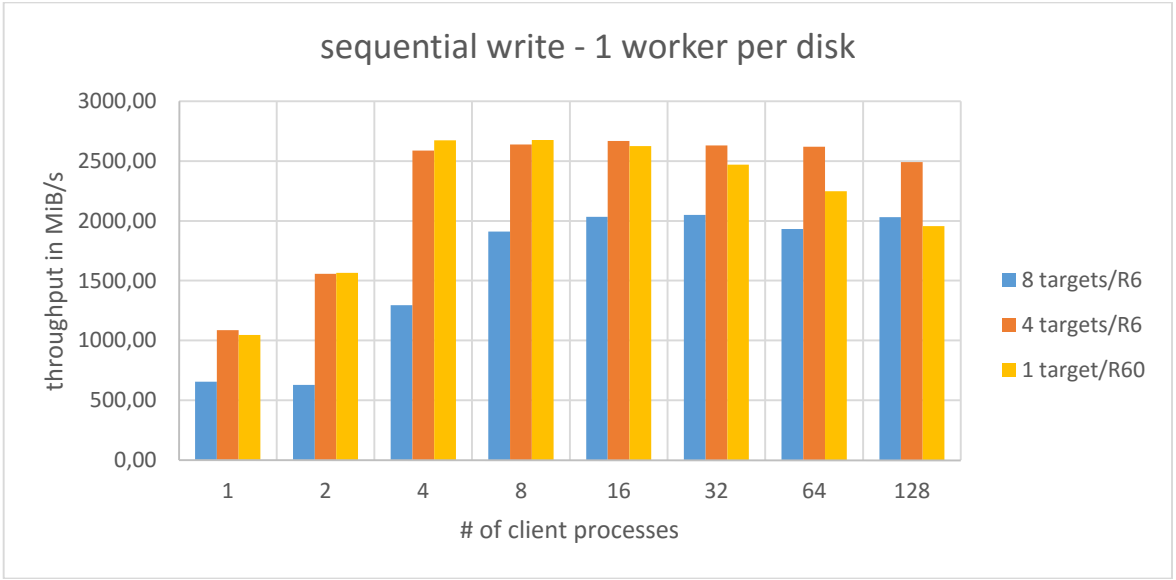


When doing sequential writes the RAID60 configuration has no benefit for lower process counts – but also doesn't lose as much performance as in the read tests when scaling the number of concurrent client processes up. Still it performs significantly worse than the 4 individual targets when 64 and 128 clients are accessing the system.

So the result implies that a larger number of targets is beneficial for concurrent access because BeeGFS separates IO requests efficiently and distributes them across the targets. To drive this further we want to compare the 4 individual RAID6 10+2 targets and the single RAID60 4x(10+2) target with a configuration of 8 individual targets in RAID6 4+2 configuration.



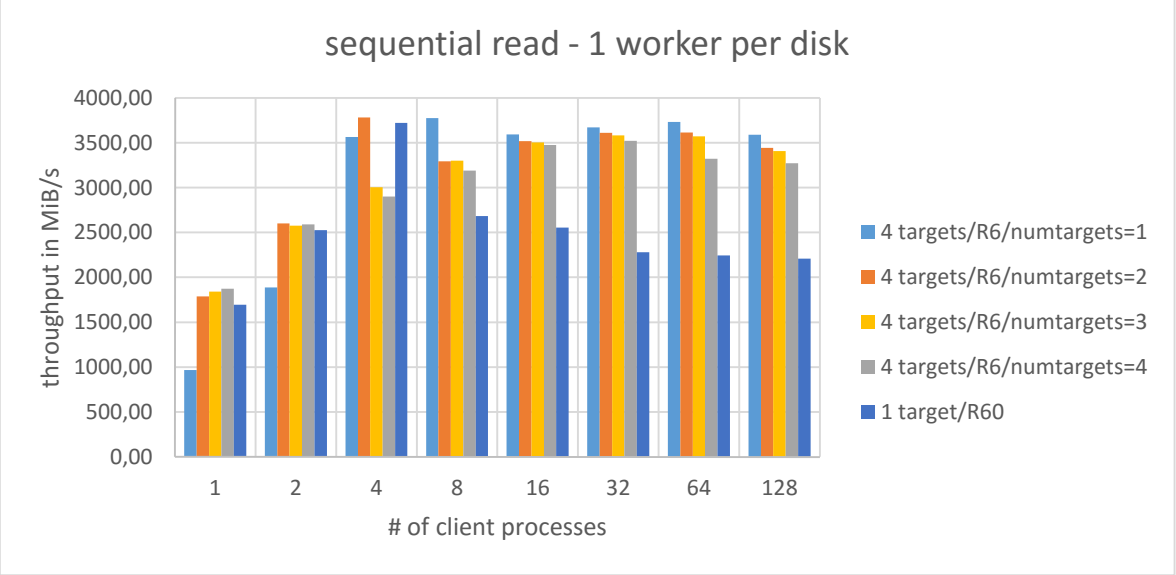

When doing sequential reads we see a significant difference in performance with the 8-target configuration – just for 64 processes we see a slight advantage – and for 128 processes a draw with the 4-target configuration. For all other tests the new configuration performs worse. Especially with a low process count (1, 2, 4) it has a disadvantage as the individual targets are much slower than before.



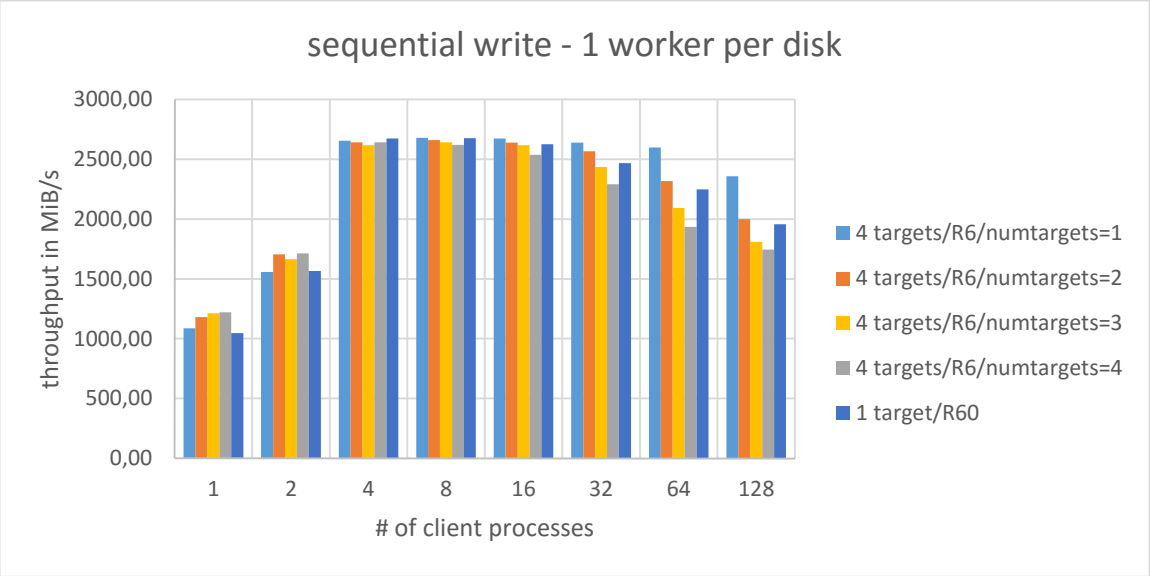
For sequential writes the 8-target configuration performs worse than the 4-target configuration in all tests. This is quite logical when looking at the RAID-mechanism – in a RAID6 4+2 just 4 disks are actually writing user data in a full-stripe where two disks are writing parity data. If the configuration is 10+2 then 10 disks are writing user data.

So 4 targets, each in 10+2, have 40 disks writing user data – but 8 targets in 4+2 have just 32 disks doing so. That alone could explain a 20% difference in performance – additionally the RAID controller seems to have a higher overhead when handling 8 arrays over 4 and becomes less efficient.

In a last test we want to answer the question if the file-striping of BeeGFS can close the gap that we have seen when running a low number of client processes (1, 2) between the configuration with 4 individual targets vs. the configuration with the single RAID60 target. We will re-run with stripe-settings of 1, 2, 3 and 4 per file.




For 1 and 2 client processes any numtargets setting larger than 1 can exceed the performance of the RAID60 target. For 4 client processes a setting of 3 and 4 for numtargets falls behind the RAID60-configuration. For all larger process counts a setting of 1 on the setup with 4 individual targets gives the best result – and the RAID60 configuration is significantly worse.




For sequential writes we see less benefit for smaller number of client processes – but larger disadvantage for higher process counts.

## 6. Conclusion

We have shown that the BeeGFS storage architecture is capable of delivering the performance of the underlying hardware to clients attached through the network.

The test results clearly show that it should be preferred to have multiple individual targets over a single target built from a very high number of disks. When increasing the number of targets too much (and therefor making the targets individually smaller and slower) no benefit can be seen or the result can get worse as the number of disks writing actual user data shrinks or other side effects spoil the result. Targets with a size of 10 to 12 drives give good results for typical use cases.

For special workloads with a low number of concurrently accessing processes a setup with a single large target is still worse than using BeeGFS' stripe setting to aggregate the performance of several targets – and the latter is more flexible as it can be set per file/per directory.

## 7. Contact information

If you want to keep updated about BeeGFS then you can follow BeeGFS on twitter (www.twitter.com/BeeGFS) or subscribe to the BeeGFS newsletter (https://groups.google.com/group/beegfs-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