Fraunhofer ITWM

FhGFS - Performance at the maximum

http://www.fhgfs.com

January 22, 2013

Contents

1. Introduction

FhGFS[1] is the parallel file system from the Fraunhofer Competence Center for High Performance Computing[2]. The distributed metadata architecture of FhGFS has been designed to provide the scalability and flexibility that is required to run today's most demanding HPC applications. In January 2013, Fraunhofer announced the new major release, named 2012.10. Besides adding interesting new features, such as on-the-fly replication of file contents and metadata, the new release introduces a completely redesigned metadata format, which increases performance even more. This document describes a set of benchmarks performed with FhGFS 2012.10 and shows their results.

2. Environment

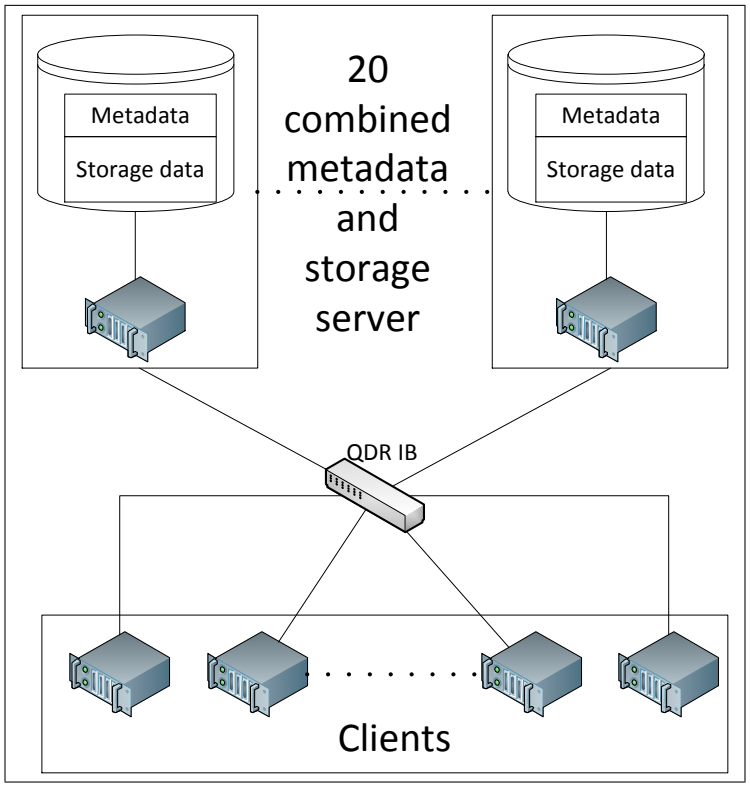


Figure 1: Hardware overview

The benchmarks have been performed on a total of 20 server nodes, which have been used as storage and as metadata servers at the same time (Fig. 1). Each of the nodes was equipped with the following hardware:

- 2x Intel Xeon X5660 (2.8 GHz)
- 48 GB DDR-3 RAM

[1] http://www.fhgfs.com
[2] http://www.itwm.fraunhofer.de/en/departments/hpc.html

- QDR Infiniband (Mellanox MT26428), only one port connected
- 4x Intel 510 Series SSD (RAID 0 with mdraid)

The used operating system was Scientific Linux 6.3 with kernel 2.6.32-279 from the Scientific Linux distribution.

All software, except FhGFS and the benchmarking tools IOR and mdtest, was installed from the Scientific Linux repository.

FhGFS version 2012.10-beta1, which is available to the public since 2012/10, was used. All FhGFS services were configured to use RDMA over Infiniband.

3. Benchmark specifications and results

According to the Parallel File System survey report[3] by the Dice Program, the most important file system metrics for data center representatives are multi-stream throughput, metadata performance and large block data performance. Although most of the metrics described by the report target on large files, the Johannes Gutenberg University in Mainz gathered some statistics, which show the importance of small file I/O. They analyzed typical file systems in data centers and found out, that 90% of the files on actual file systems are only 4KB or less in size[4]. So even if the majority of file system administrators consider benchmarks related to large file I/O to be the most important, the University's study clearly shows the significance of handling small files in a real-world environment. As a result, we decided to specify a set of benchmarks, covering large block streaming throughput, as well as small file I/O and metadata performance.

All of the following benchmarks have been performed by using the tools IOR[5] and mdtest[6]. Each of the measurements was run five times and the result was calculated as the mean average of the single runs.

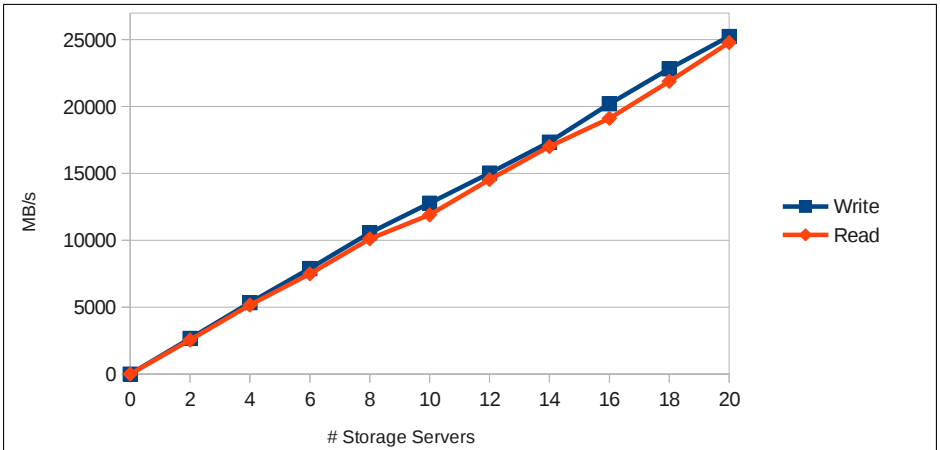
The command lines used to run the following benchmarks can be found in Appendix A.

3.1. Multi-stream throughput

In this benchmark the total throughput of sequential read and write requests with multiple streams was measured.

This test was performed in two different ways. During the first measurement, a constant amount of 160 clients was used and the number of storage servers scaled from two to 20. The second test was performed with all storage servers, increasing the number of clients up to 768.

[3] http://www.avetec.org/applied-computing/dice/projects/pfs/docs/PFS_Survey_Report_Mar2011.pdf
[4] A Study on Data Deduplication in HPC Storage Systems; Dirk Meister et al.; Johannes Gutenberg Universität; SC12
[5] http://sourceforge.net/projects/ior-sio/
[6] http://http://sourceforge.net/projects/mdtest/



# Servers	2	4	6	8	10	12	14	16	18	20
Write (MB/s)	2657	5335	7892	10561	12782	15023	17345	20206	22823	25247
Read (MB/s)	2549	5162	7492	10112	11904	14543	17012	19123	21889	24789

Figure 2: **Sequential throughput, storage server scaling**; 160 clients, 1-20 storage servers, 512k transfer size, filesize of 150GB per server

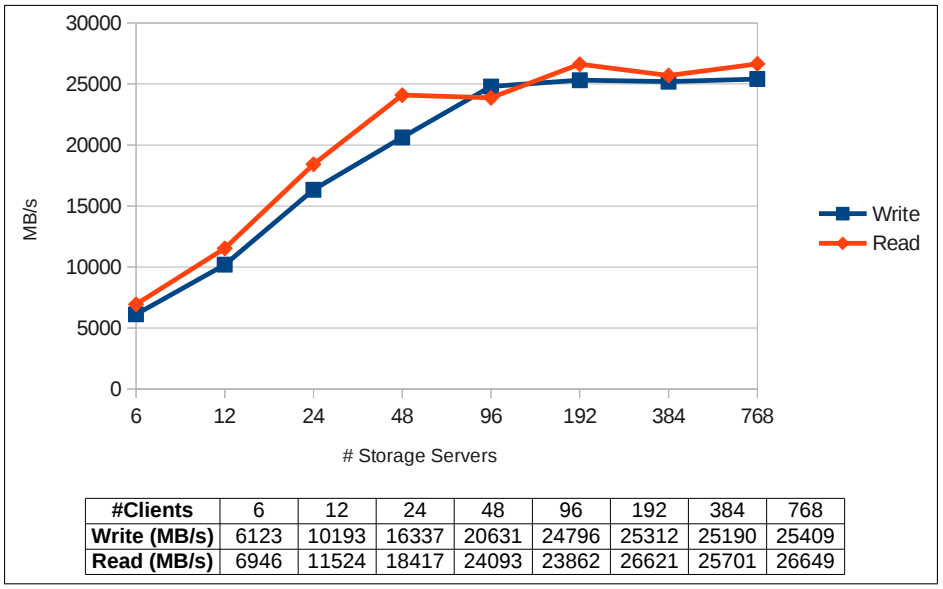
Each file in this run was striped across four FhGFS storage servers with a chunk size of 512kb.

With two storage servers, the system achieved a throughput of 2 657 MB/s when writing and 2 549 MB/s when reading. As we added more storage servers to the system, performance scaled nearly linear (Fig. 2). When using all 20 servers, a sustained write throughput of 25 247 MB/s and a sustained read throughput of 24 789 MB/s was measured.

The local RAID of a single node delivered a sustained writing performance of 1 332 MB/s and a sustained reading performance of 1 317 MB/s. With this information, one can calculate the maximum throughput, which 20 servers are theoretically able to achieve and it can be seen that FhGFS could provide 94% of this value (writing: 94.7%, reading: 94.1%).

With a constant number of 20 storage servers and an increasing number of client processes accessing the file system, one can see, that the sustained performance was stable with a growing number of clients, up to 768. (Fig. 3)

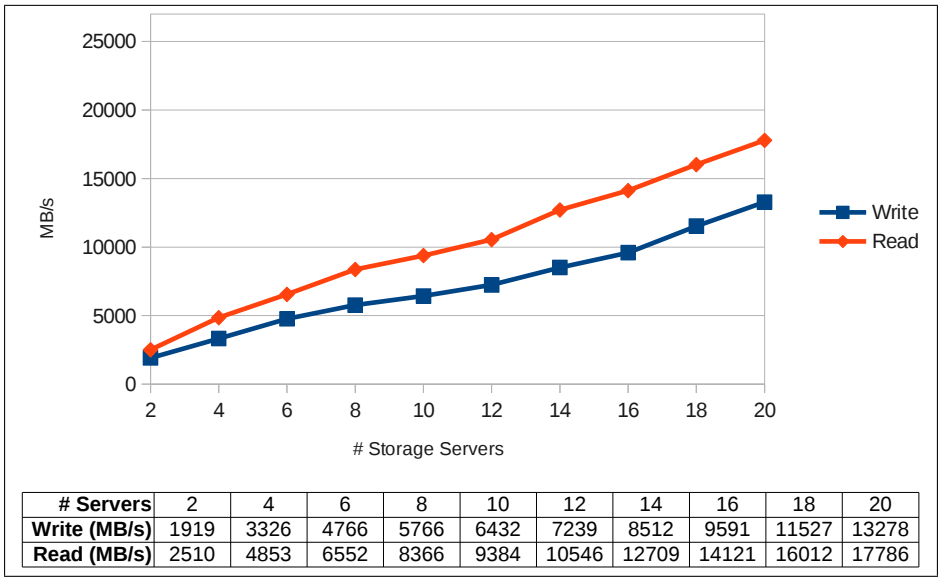
The minimum number of clients required to get the maximum performance was in the range between 96 and 192. Local benchmarks showed that at least six concurrent processes on a single machine are needed to achieve the maximum local throughput. As this benchmark used the combined resources of 20 machines, it is very reasonable that the amount of processes needed to reach maximum performance must be about 20 times higher as on a single machine.



#Clients	6	12	24	48	96	192	384	768
Write (MB/s)	6123	10193	16337	20631	24796	25312	25190	25409
Read (MB/s)	6946	11524	18417	24093	23862	26621	25701	26649

Figure 3: **Sequential throughput, client scaling**; 20 storage servers, 6-768 clients, 512k transfer size, filesize of 150GB per server

3.2. Shared file throughput



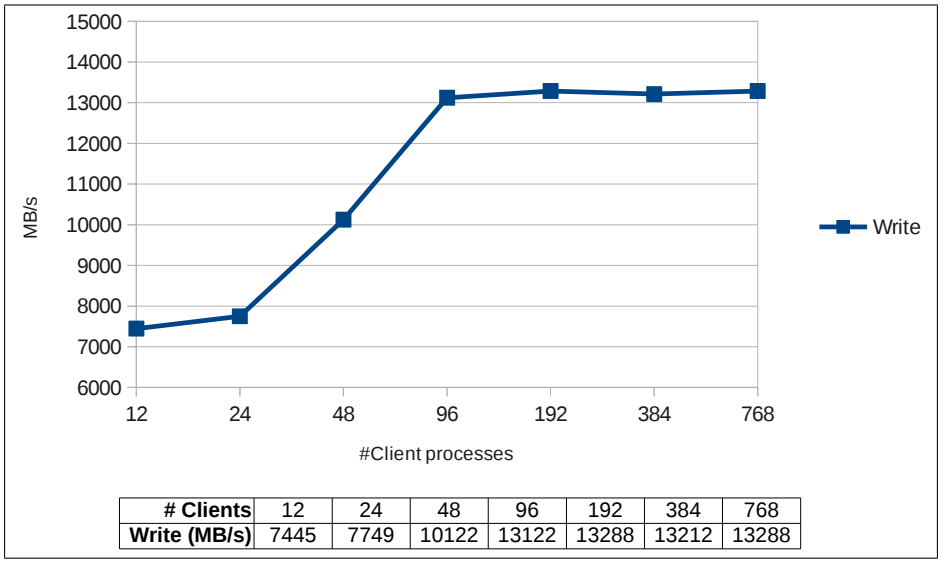
# Servers	2	4	6	8	10	12	14	16	18	20
Write (MB/s)	1919	3326	4766	5766	6432	7239	8512	9591	11527	13278
Read (MB/s)	2510	4853	6552	8366	9384	10546	12709	14121	16012	17786

Figure 4: **Shared file throughput, storage server scaling**; 192 clients, 1-20 storage servers, 600k transfer size, filesize of 150GB per server

In this scenario clients accessed one single shared file and performed read and write operations on it. Most applications access files in an unaligned manner, as the record size is defined by the type of data and the used algorithms. Therefore we chose 600kb as transfer size in IOR. Of course, this is not optimal for the file system, but by using an unaligned access pattern, we simulated real-world examples. Once again, we performed two different measurements. The first one was with a constant number of 160 clients and a growing number of storage servers. The second one used all storage

servers again, and scaled up to 768 clients sharing one single file. As we only wrote one file, we needed to set the number of targets for this file to 20 to distribute it evenly over all servers.

The benchmarks showed a good scalability with a growing number of storage servers (Fig. 4). It is notable, that the shared file write performance was only about 75% of the read performance. We could also observe this behaviour by using the SSD disks directly on a local node (1 403 MB/s read and 1 108 MB/s write), so this seems to be a general limitation of the used systems.



# Clients	12	24	48	96	192	384	768
Write (MB/s)	7445	7749	10122	13122	13288	13212	13288

Figure 5: **Shared file throughput, client scaling**; 20 storage servers, 6-768 clients, 600k transfer size, filesize of 150GB per server

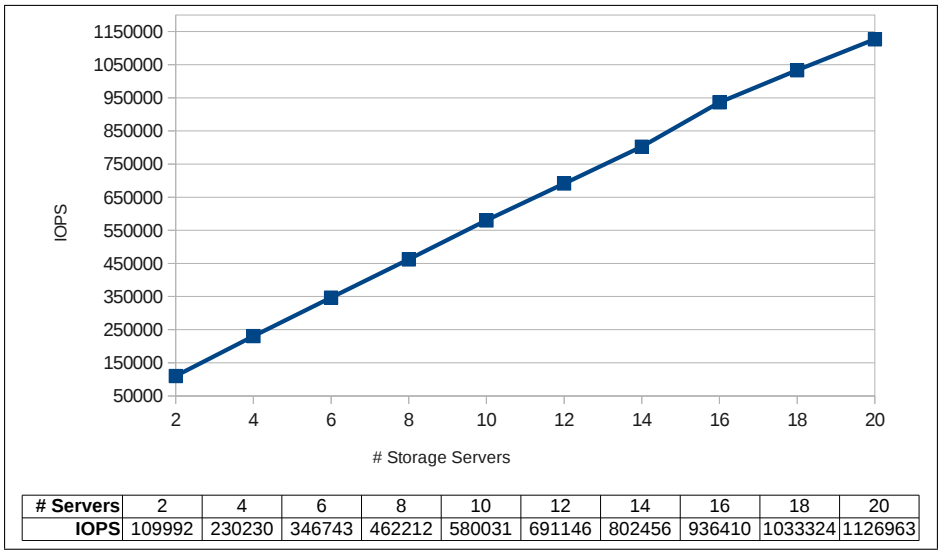
Shared file throughput on 20 servers, with a number of clients growing up to 768, showed that all client processes can write to the same file without any notable performance loss (Fig. 5).

Once again, we reached the maximum performance at a point slightly higher than 96 client processes, which corresponded with single node benchmarks. On a local node, six concurrent processes were needed to achieve maximum performance.

3.3. IOPS

In this benchmark the I/O operations per second were measured. This value basically represents the mean access time of the file system and is very important for dealing with small files and for algortihms, which access files in a very random way. To measure this metric, 4k random writes were used with a total number of 160 clients and a growing number of storage servers.

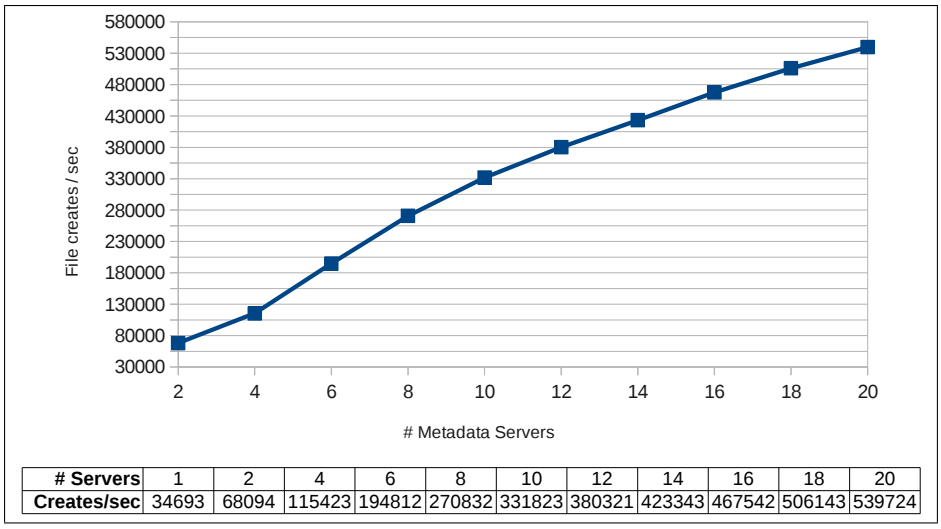
These measurements resulted in a perfectly linear scale. While two storage servers were able to perform 109 992 operations/sec., one can observe 1 126 963 operations/sec. for 20 storage servers (Fig. 6).



# Servers	2	4	6	8	10	12	14	16	18	20
IOPS	109992	230230	346743	462212	580031	691146	802456	936410	1033324	1126963

Figure 6: **IOPS**; 1-20 storage servers, 160 clients

3.4. Metadata performance



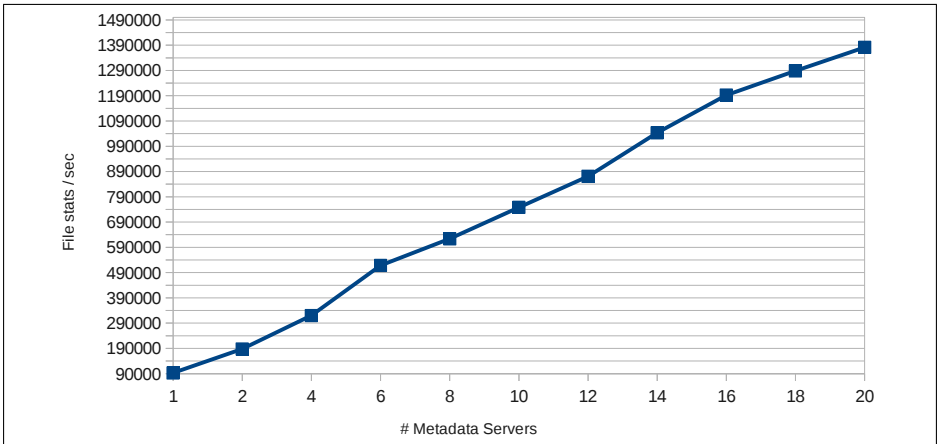
# Servers	1	2	4	6	8	10	12	14	16	18	20
Creates/sec	34693	68094	115423	194812	270832	331823	380321	423343	467542	506143	539724

Figure 7: **File creation**; 1-20 servers, up to 640 client procs (32*#MDS)

The number of file creates per second and the number of stat operations per second were determined with an increasing number of metadata servers. The number of clients was always proportional to to the number of servers and was calculated by multiplying the number of servers with 32. This is important because benchmarks on a single node showed that the local RAID-arrays only perform well when using between 20 and 40 accessing processes. Therefore we needed to adjust the number of processes according to the number of used servers.

Both metadata performance benchmarks showed a rise in performance when adding new metadata servers. While file creation rates on a single metadata server were in the range of 35 000, a file system with 20 metadata servers was able to create significantly more than 500 000 files per second (Fig. 7). Stat call performance showed a rise from about 93 000 operations on one server to 1 381 339

operations on 20 servers (Fig. 8).



# Servers	1	2	4	6	8	10	12	14	16	18	20
Stats/sec	93007	187844	320134	518401	623682	748334	871234	1043231	1192625	1288153	1381339

Figure 8: **Stat calls**; 1-20 servers, up to 640 client procs (32*#MDS)

4. Conclusion

FhGFS was designed to be the primary choice for scratch filesystems in the HPC market. Our results demonstrate that FhGFS is a highly scalable parallel filesystem and in most scenarios the throughput performance is not limited by FhGFS itself, but only by the underlying hardware. Although metadata benchmarks did not show a perfectly linear scalability, the results prove that adding metadata servers to the system leads to a significant increase in metadata performance. Therefore it is easily possible to load-balance metadata access in busy environments with a load of access from different users.

A. Used command lines for benchmarks

A.1. Multi-stream throughput

```
mpirun -hostfile /tmp/nodefile -np ${NUM_PROCS} \
/usr/bin/IOR -wr -C -i5 -t512k -b${BLOCK_SIZE} \
-F -o /mnt/fhgfs/test.ior
```

with *${NUM_PROCS}* being 160 for the first test and increasing values from six to 768 for the second test. The parameter *${BLOCK_SIZE}* depends on the number of client processes and servers. Every run writes and reads 150GB multiplied by the number of active servers of total data, so the individual client process' block size can be calculated by dividing this value by the number of client processes.

A.2. Shared file throughput

```
mpirun -hostfile /tmp/nodefile -np ${NUM_PROCS} \
/usr/bin/IOR -wr -C -i5 -t600k -b${BLOCK_SIZE} \
-o /mnt/fhgfs/test.ior
```

with *${NUM_PROCS}* being 192 for the first test and increasing values from six to 768 for the second test. The parameter *${BLOCK_SIZE}* depends on the number of client processes and servers. Every run writes and reads 150GB multiplied by the number of active servers of total data, so the individual client process' block size can be calculated by dividing this value by the number of client processes.

A.3. IOPS

```
mpirun -hostfile /tmp/nodefile -np 160 \
/usr/bin/IOR -w -C -i5 -t4k -b${BLOCK_SIZE} \
-F -z -o /mnt/fhgfs/test.ior
```

with *${BLOCK_SIZE}* depending on the number of servers. Every run writes 150GB multiplied by the number of active servers of total data, so the individual client process' block size can be calculated with $(150GB * \#servers)/160$.

A.4. Metadata performance

For the create benchmark:

```
mpirun -hostfile /tmp/nodefile /tmp/nodefile -np ${NUM_PROCS} \
mdtest -C -d /mnt/fhgfs/mdtest -i 5 -I ${FILES_PER_DIR} -z 2 \
-b 8 -L -u -F
```

For the stat benchmark:

```
mpirun -hostfile /tmp/nodefile /tmp/nodefile -np ${NUM_PROCS} \
mdtest -T -d /mnt/fhgfs/mdtest -i 5 -I ${FILES_PER_DIR} -z 2 \
-b 8 -L -u -F
```

with *${NUM_PROCS}* being $\#servers * 32$. The total amount of files should always be higher than 1 000 000, so *${FILES_PER_DIR}* is calculated as
$\${FILES\_PER\_DIR\} = \lceil 1000000/64/\$NUM\_PROCS \rceil$.