SEVENTH FRAMEWORK PROGRAMME

FP7-ICT-2013-10

DEEP-ER

DEEP Extended Reach

Grant Agreement Number: 610476

D4.5

Results of I/O performance measurements

Approved

Version:	2.0
Author(s):	K. Thust (JUELICH)
Contributor(s):	C. Clauss (ParTec), W. Frings (JUELICH), A. Galonska (JUELICH), F. Kautz (FHG-ITWM), J. Kreutz (JUELICH), A. Zitz (JUELICH)
Date:	04.05.2017

Project and Deliverable Information Sheet

<table>
<tr><td rowspan="7">DEEP-ER Project</td><td colspan="2">Project Ref. №: 610476</td></tr>
<tr><td colspan="2">Project Title: DEEP Extended Reach</td></tr>
<tr><td colspan="2">Project Web Site: http://www.deep-er.eu</td></tr>
<tr><td colspan="2">Deliverable ID: D4.5</td></tr>
<tr><td colspan="2">Deliverable Nature: Report</td></tr>
<tr><td rowspan="2">Deliverable Level:
PU *</td><td>Contractual Date of Delivery:
31 / March / 2017</td></tr>
<tr><td>Actual Date of Delivery:
31 / March / 2017</td></tr>
<tr><td colspan="2">EC Project Officer: Juan Pelegrín</td></tr>
</table>

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

<table>
<tr><td rowspan="6">Document</td><td colspan="2">Title: Results of I/O performance measurements</td></tr>
<tr><td colspan="2">ID: D4.5</td></tr>
<tr><td>Version: 2.0</td><td>Status: Approved</td></tr>
<tr><td colspan="2">Available at: http://www.deep-er.eu</td></tr>
<tr><td colspan="2">Software Tool: Microsoft Word</td></tr>
<tr><td colspan="2">File(s):DEEP-ER_D4.5_results_of_io_performance_measurements_v2.0-ECapproved</td></tr>
<tr><td rowspan="4">Authorship</td><td>Written by:</td><td>K. Thust (JUELICH)</td></tr>
<tr><td>Contributors:</td><td>C. Clauss (ParTec), W. Frings (JUELICH), A. Galonska (JUELICH), F. Kautz (FHG-ITWM), J. Kreutz (JUELICH), A. Zitz (JUELICH)</td></tr>
<tr><td>Reviewed by:</td><td>H.Servat (Intel), N.Eicker (JUELICH)</td></tr>
<tr><td>Approved by:</td><td>BoP/PMT</td></tr>
</table>

Document Status Sheet

Version	Date	Status	Comments
1.0	31/March/2016	Final	EC submission
2.0	04/May/2017	Approved	EC approved

Document Keywords

Keywords:	DEEP-ER, HPC, Exascale, I/O Architecture, Benchmarking

Table of Contents

List of Figures

Executive Summary

In this deliverable we present the benchmark results and their analyses for different I/O components of the DEEP-ER system.

After a short introduction the hardware and software setup are summarised. On this basis the following sections present the results of different metrics for the cluster and the booster part and for global and local storage. Finally, the continuous benchmarks are shown as well as a summary of this deliverable and possible future analyses regarding I/O.

1 Introduction

This deliverable describes and analyses the results of the synthetic I/O benchmarks of the DEEP-ER system. In the context of different benchmarking related deliverables towards the end of the project, the focus of the present document lays on the raw performance data, longer term performance analysis and caching effects.

In order to capture a broad range of scenarios, deliverable D5.4 “Resiliency performance measurements” explores I/O in the context of resiliency with a focus on local storage for buddy checkpointing. While both D4.5 and D5.4 consider more synthetic tests, D6.3 “Final report on applications experience” describes the benchmarking results of applications.

In the present DEEP-ER system different I/O components are relevant for different use cases. Therefore, section 2 provides a brief overview of the benchmarking setup used for our efforts. Subsequently, sections 3 and 4 describe the measurements on the Cluster Nodes (CNs) and the Booster Nodes (BNs). For both parts of the system the performance for the global and the local file system has been analysed. Finally in section 5 the results of the long-term continuous benchmarks are presented to show changes of the system performance over time.

2 Benchmarking setup

2.1 Hardware setup

By the time of writing this deliverable the envisioned DEEP-ER system is not completely available, yet. Hence, the benchmarks were conducted on the so-called “software development vehicle” (SDV) as Cluster Nodes and KNL nodes that are already available as Booster Nodes. Since the actual hardware components chosen for these preliminary test-systems are identical to the ones scheduled to be available in the final DEEP-ER system we do not expect major deviations from the current results once the DEEP-ER system is available.

Regarding the file system the available setup consists of two storage servers and one metadata server.

The technical specifications are

- 16 Cluster Nodes, each equipped with:
  - 2 × Intel Xeon E5-2680v3, 12-Core (Haswell)
  - 128 GB RAM (DDR4 / PC2133 ECC)
  - 400 GB NVMe Intel DC P3700 SSD
  - 2 × 1.0 TB S-ATA 6 Gb/s HDDs
  - EXTOLL Tourmalet NIC
- 8 Booster Nodes, each equipped with:
  - 1 × Intel Xeon Phi 7210 64-Core (Knights Landing)
  - 16 GB MCDRAM + 96 GB RAM (DDR4 / PC2133 ECC)
  - 400 GB NVMe Intel DC P3700 SSD
  - EXTOLL Tourmalet NIC
- File Servers (2 Storage + 1 Metadata)
  - 2 × Intel Xeon E5-2609 v3
  - EUROstor ES-6600 with 4 × 8Gbit FC connector and 24 × 6 TB SAS Nearline
  - 2 × 200 Solid State Drive SATA Mix Use MLC 6Gbps 2.5in Hot-plug Drive
  - EXTOLL Tourmalet NC

The storage’s RAID controller organises the disks in one RAID set with 23 disks and 1 hot spare. In this RAID set there are 4 RAID volume sets with RAID 6 over the 23 disks, where each of the volume sets offers 31.5 TiB. Those are available to the storage servers via fibre channel, 2 volumes for each storage server. To allow for more flexibility only one of the two volumes on the storage servers is used to form the global files system /sdv-work. This design choice was taken to be able to have two independent global files systems at the same time, e.g. for testing or update and migration scenarios. Given some overhead of the file system the global storage available to the users is approximately 60 TiB.

2.2 Software setup

The benchmarks were conducted using the tools described in D4.3 “Definition of test cases and patterns”. The main analysis part in sections 3 and 4 consists of measurement of the bandwidth (using IOR and partest). In contrast metadata operations (mdtest) were mainly analysed continuously over time as described in section 5. Due to technical issues, the

LinkTest measurements, described in D4.3, did not yield meaningful results and are hence excluded from the long term analysis.

Beyond the general description of the parameters as described in D4.3 the following describes some specifics about the employed tools which are relevant for all benchmarks in this deliverable.

Many measurements are more specific to the file system and I/O patterns than to the client writing the data. Consequently, parts of the benchmarks have only been conducted on the Cluster, while the results are relevant for the Booster part as well.

2.2.1 IOR

IOR is a generic file system benchmark with a very broad range of parameters, e.g. transferSize (volume transferred in a single call), blockSize (total amount of data transferred per process), interfaces (POSIX, MPI-IO, HDF5, …) and different strategies of flush operations. In this deliverable we focus on the I/O performance using the POSIX interface. It is important to note that in IOR the amount of data written – given by the `blockSize` parameter – is measured per process and the aggregate size is the product of `blockSize × processes`.

2.2.2 Partest

As one component of the checkpointing process in the DEEP-ER project SIONlib is used to determine synthetically the I/O performance for checkpointing. Most optimisations leveraged by SIONlib to improve I/O performance are designed to circumvent metadata problems at large scale. Measurements from other systems showed those optimisations to become important for I/O from 1k to 10k tasks and to be crucial at very large scale with far more than 100k tasks. On the given DEEP-ER system the best performance is still expected to be reached with task-local I/O. This especially holds since BeeGFS is specifically designed to handle task-local I/O efficiently.

SIONlib comes with its own benchmarking program “partest” which provides estimates for the performance of different I/O patterns. Partest is introduced and described in more detail in D4.3.

2.2.3 Global file systems

For the DEEP-ER system there is a dedicated file system running BeeGFS with the additional features developed within the project. This file system is mounted on all Cluster and Booster nodes as well as on the login nodes as `/sdv-work`. As the predecessor system DEEP shares the same login nodes, for some tests the performance of this previous BeeGFS file system used in the DEEP project is used for comparison.

2.2.4 Local file systems

Each Cluster and Booster node is equipped with a local NVMe. These devices are available directly through the mount point `/nvme/tmp`, which uses ext4 as file system. On top of this local file system a BeeGFS on demand instance (BeeOND) is mounted as `/mnt/beeond` and provides the new features developed in this project like asynchronous flushing to the global file system.

2.2.5 Cache effects

As pointed out in the introduction, in contrast the accompanying deliverables D5.4 and D6.3, within this document we try to determine the effect of file system caches. This especially includes different approaches to prevent data from being cached.

There are caches both in the local client and on the storage server. On its way from an application on a compute node to a disk on the storage server, data may see different caches, including (but not necessarily limited to)

- File system cache on the client
- BeeGFS caches on the client
- Network buffers
- BeeGFS caches on the storage server
- File system caches on the storage server
- Caches in the RAID controller

This makes it very challenging to analyse the exact origin of caching effects for every specific benchmarking setup.

3 Benchmarks on the SDV

The SDV forms the cluster component of the DEEP-ER system. As some analyses are of a more general nature they are not repeated for both parts of the system (Cluster and Booster) or both storage types (global and local) but only one component was tested. Regarding the different parts of the system, the SDV had the advantage of more nodes being available, which made it the preferred platform for testing. With respect to global vs. local file system the focus in this deliverable is on the global file system. The local file system is tested as well but the less intensive than in D5.4.

3.1 Global file system

3.1.1 Bandwidth overview

For a first overview of the I/O bandwidth available via the global file system we show the write performance from 8 nodes of the SDV to the global file system in Figure 3-1. The measurements are conducted using IOR and include writing from one process per node (8 procs) and two processes per node (16 procs) as well as writing to a single shared file (‘shared file’) and writing to individual files per process (‘file per process’). Unlike in partest, file access to single shared files in IOR does not use SIONlib. The data points for this plot as for all following represents the average value of 10 measurements (iterations) and the error bars represent standard deviation. As pointed out in the introduction the aggregate size is the product of the block size for a single process and the number of processes.

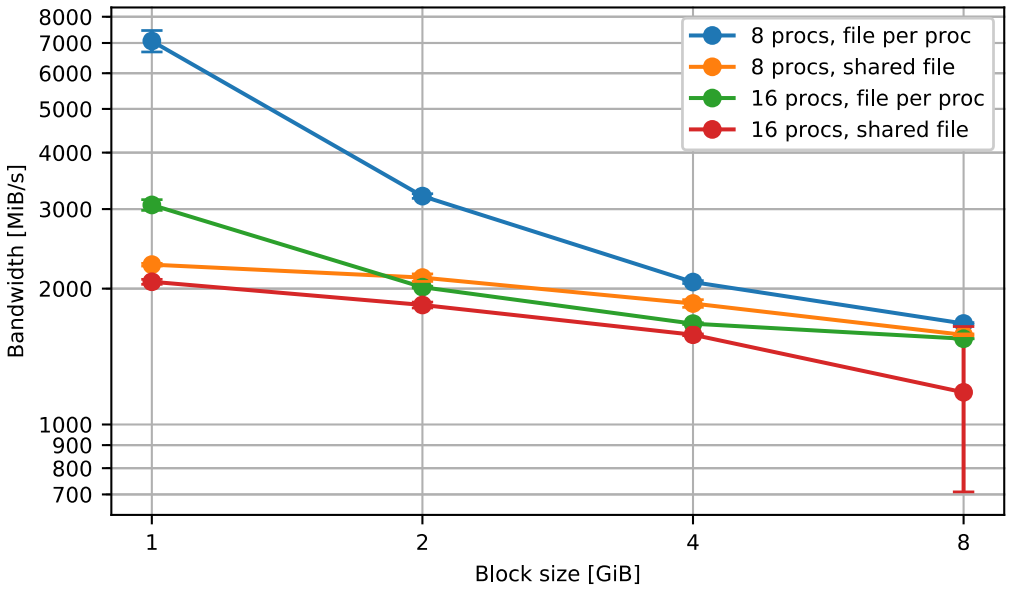

Figure 3-1: IOR write bandwidth for varying block sizes and different number of processes

Very prominent is the high bandwidth for writing with 8 processes to individual files. The main reason for this result is caching which is discussed in detail in this deliverable. Comparing the result of 8 processes with the one of 16 processes shows that the relevant cache does not scale with the number of processes since doubling the block size for 8 processes results in a comparable bandwidth as the one for 16 processes. Likely candidates for this cache are either a shared cache per node or they reside in the server. Regarding the file access, concurrent access to a single shared file reduces the efficiency of the file system cache

significantly. This results in a very similar bandwidth of approximately 1600 MiB/s for 8 GiB per process for all but the 16 processes, shared file test. Most of the 10 iterations for 16 processes to a shared file yield roughly 1470 MiB/s but some outliers result in the lower average and larger standard deviation. The reason for that might be some concurrent access to the file system, as it was not always exclusively available for benchmarking by a single user. In contrast, the standard deviation shown for 8 processes and one file per process is the result of more evenly distributed values. As this case is strongly affected by the cache this shows that the file system handles requests more evenly than the cache. Most other measurements show an almost constant performance over all iterations.

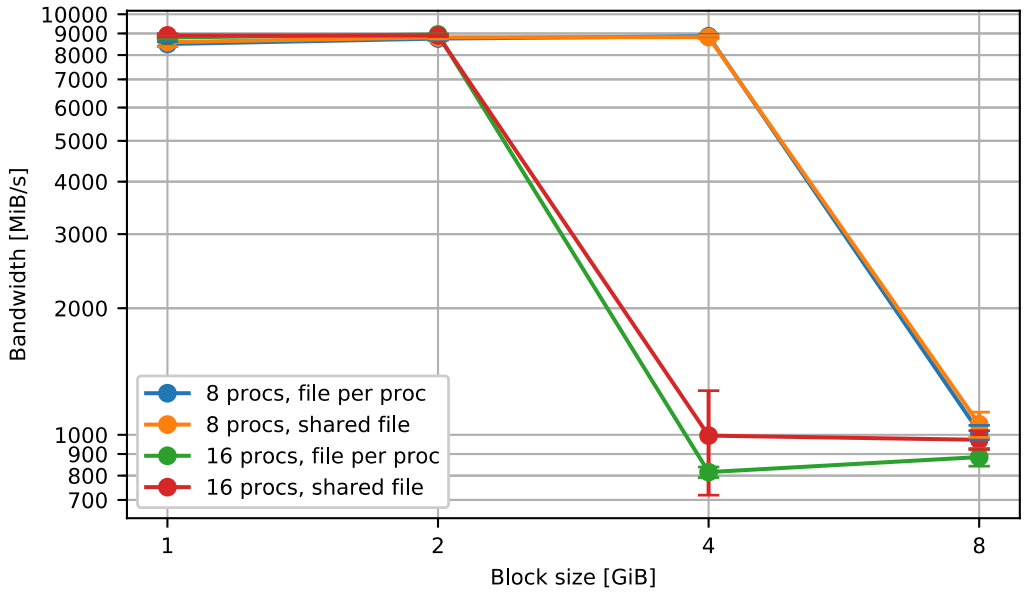

Figure 3-2: IOR read bandwidth for varying block sizes and different number of processes

Figure 3-2 shows the read performance for the same measurements. Here the effect of the cache is even more pronounced. For the write case parts of the data can be written to cache and only the remainder needs to be written directly, this leads to a continuous decline of the cache efficiency with larger file sizes. The read case is different in that it reads from cache as long as all the data fits into it and for larger data all of the relevant data is evicted. Both, IOR and partest write data first and read back the same data. This immediate reading of the same data pronounces the caching effects but is not purely synthetic. A possible scenario is reading back data after checkpointing with no significant I/O in between. Similar to the write case reading also shows the dependence on the aggregate size, so the relevant cache in the read case also does not scale with the number of processes. The results are consistent with the expectation for the 32GiB RAM each of the two storage servers has, as some amount is always needed for the operating system. This suggests the global file system cache as most relevant for these measurements. Measurements that circumvent local caching by reordering tasks for the read process are discussed in section 3.1.3 and support this suggestion

Regarding the bandwidths, the read speed rates are approximately 8.5 GiB/s and 1 GiB/s for cached and uncached reads, respectively. The latter corresponds to the maximum performance of the storage as measured locally on the storage systems. The very high bandwidth of 8 GiB/s and above is limited by the network bandwidth. From a single storage server to an SDV node the raw network bandwidth measured with the network performance benchmark `iperf3` is about 4 GiB/s, so the expected bandwidth when writing to both

storage servers is roughly twice as high. This of cause presumes writing from at least two SDV nodes, which is given in this setup.

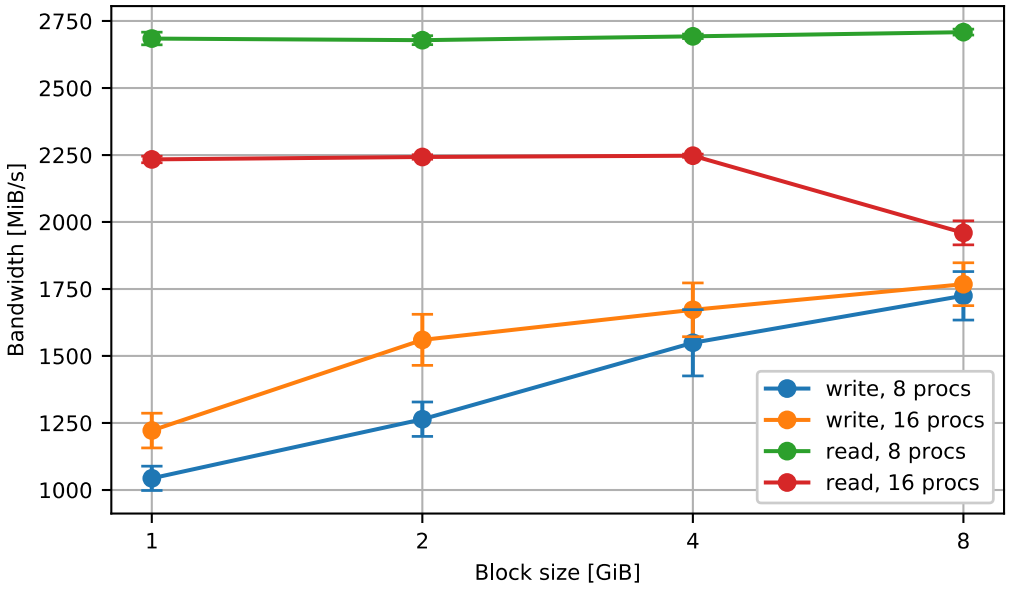

Figure 3-3: IOR read and write bandwidth for varying block sizes and different number of processes on the DEEP cluster (/work)

The very dominant effect of caches is something new for the more recent version of BeeGFS on the DEEP-ER system. While still converging at a comparable bandwidth, Figure 3-3 does not show a caching behaviour as its successor does. Although significantly smaller a cache effect can still be seen in the read performance of 16 nodes. Here the bandwidth drops between 4 GiB and 8 GiB per process (64 GiB and 128 GiB aggregate size), which corresponds to twice the amount of memory available for the DEEP storage servers as compared to the DEEP-ER storage servers. It remains unclear why the write bandwidth increases with increasing block size.

3.1.2 The effect of fsync

In order to measure the storage bandwidth provided by the system it is desirable to reduce caching effects. One possible strategy is to use data sizes that are large compared to the cache size for benchmarks. This works fine in for the read case, as described in the previous section. The write case is more difficult, as data is always first stored in the write cache and only the data, which does not fit into the cache, is limited by the actual storage bandwidth. To avoid this asymptotic behaviour and to be able to measure accurately, explicit files system hints are required.

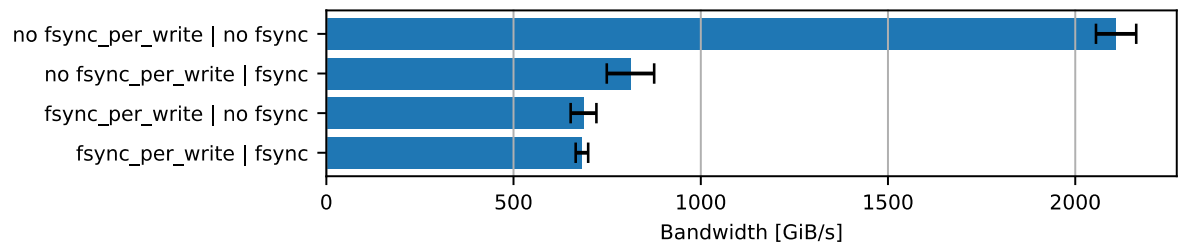

Figure 3-4: IOR write bandwidth for varying kinds of synchronisation

Figure 3-4 shows the effect of `fsync` calls, either at the end of the writing process (‘fsync’) or after each individual write call (‘fsync_per_write’). For reference also the write performance without any calls and a test with both options applied is also added to the graph. Of course the letter adds a useless `fsync` call after all single `write` and `fsync` operations are done. Nevertheless, this last `fsync` operation shall basically render to be a `noop` for the timescale under investigation. The setup for these tests is

- 8 nodes with 1 ppn
- Single shared file with 16 GiB aggregate file size.

All measurements show low or moderate standard deviation. The `fsync` call shows the expected effect in reducing bandwidth when adding a synchronisation at the end of the writing process and reducing it even more when synchronising after each write call, e.g. due to the overhead associated with every `fsync` call. Adding a final synchronisation to already synchronised data does not change the behaviour. Comparing the numbers to the result of the initial measurements shows that `fsync` seems to slow down the writing process more than regular large size write do. This is also expected as the large size of the data only reduces the relative amount that fits into the cache and so writing without the `fsync` call still benefit from it.

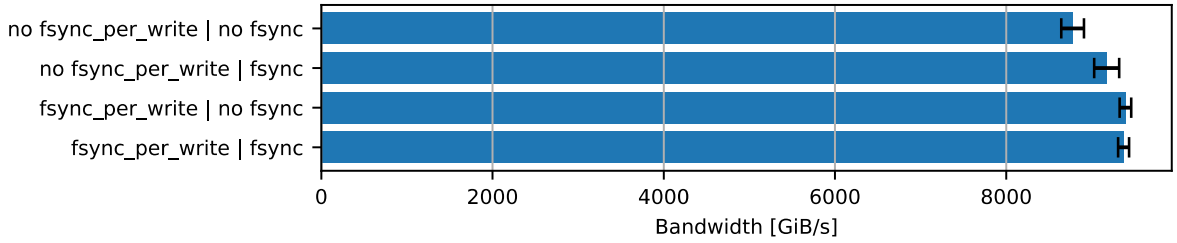

Figure 3-5: IOR read bandwidth for varying kinds of synchronisation

Regarding the read bandwidth `fsync` does not have strong effects. A possible explanation for the unintuitive effect, that more `fsync`s yield a higher bandwidth, is that these calls cause a more rigorous eviction of old cache data and beneficial placement of the new data.

3.1.3 Reordering and direct I/O

In the context of caching IOR offers an option to not read back the data each process wrote in the previous phase, but to read another processes data. This is an important tool to determine if caching happens locally or globally. In case of local caches per node this option will completely nullify the benefits for those processes that read data written on another node. For caches that are local to the process, no process could read cached data. In our tests no changes in the performance of read operations were observed. The good agreement of the read performance with the network bandwidth already suggests that the relevant cache is on the storage system and not on the local client. The results from reordering the data support this assumption.

Instead of flushing caches, POSIX allows programmers to force direct I/O by using the flag `O_DIRECT`, when opening a file. For the setup

- 8 nodes, 1 ppn
- Block size 1 GiB (= 8 GiB aggregate data size)

we have found close to constant performance regardless of the type of I/O (shared file vs. file per proc) or any additional `fsync` operations. The write performance was about 1440 MiB/s with a standard deviation of <2% for most of the tests. In contrast the bandwidth without `O_DIRECT` varies largely in a range of 670 – 6200 depending on the type of I/O. The read performance with `O_DIRECT` shows approximately 350 MiB/s, also with a very low standard deviation. This very low value results from a very low level of optimisation, as this flag explicitly asks to avoid buffering, which is a major part of optimisation. So measurements with direct I/O can usually be seen as lower bound for the bandwidth.

3.1.4 Transfer sizes

One common issue limiting I/O performance is the amount of data written in a single `write` call. In IOR the variable determining this size is called `transferSize`. Depending on the data layout, scientific applications may write rather small amounts of data per call. A possible example is looping over a container with particles and writing each of them individually instead of writing the data structure as a whole.

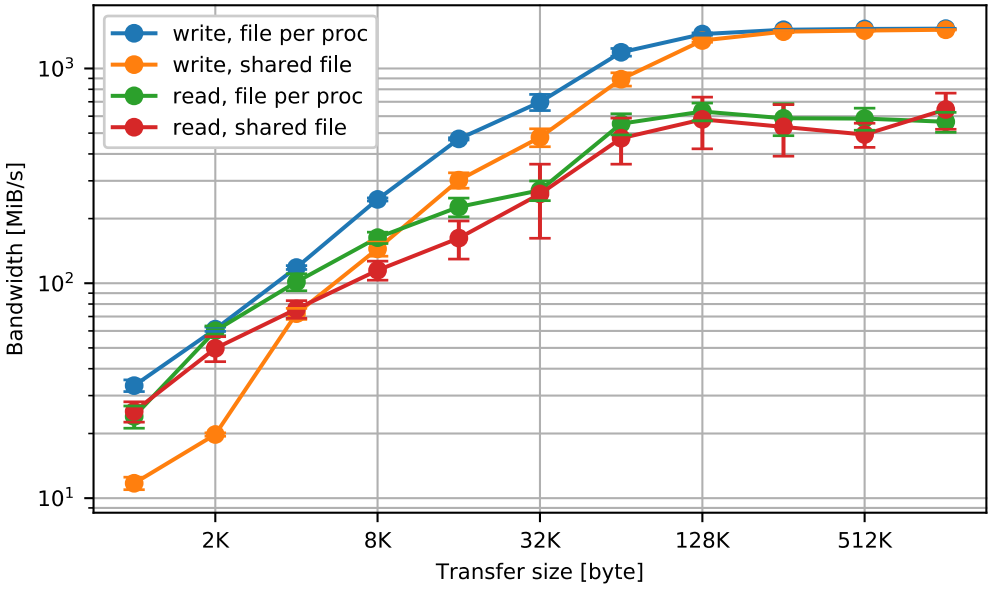

Figure 3-6: IOR bandwidth for shared and individual files with direct I/O for varying transfer sizes

In Figure 3-6 we show the read and write bandwidth for different transfer sizes. The setup for these tests is

- 8 nodes, 1 ppn
- Block size 32 MiB (=256 MiB aggregate size)
- Direct I/O

With direct I/O the bandwidth between read and write changes qualitatively: while for regular I/O reading is usually significantly faster than writing the behaviour swaps for direct I/O (given reasonable transfer sizes). Even with direct I/O there are usually some minor caches which allow write calls to return faster than read calls. A possible scenario is caching of the data of a single call, which would allow a write call to return faster, as parts of the data can be written to disk while waiting for the next call. For these minuscule caches read operations would not benefit at all, as the caches are long evicted before the same data is read again.

Due to the large scale not only in the transfer size, but also in the bandwidth we use logarithmic scale for both axes. Although especially the measurements for reading from a shared file show larger standard deviation all read and write methods show a roughly linear increase of bandwidth up to 32 or 64 KiB. At this size the lines flatten into their final bandwidth. The file system block size on in the global file system is 512 KiB, which makes this finding less than obvious as direct I/O for sizes smaller than the file system block size should degrade the performance. The reason for this behaviour is unclear.

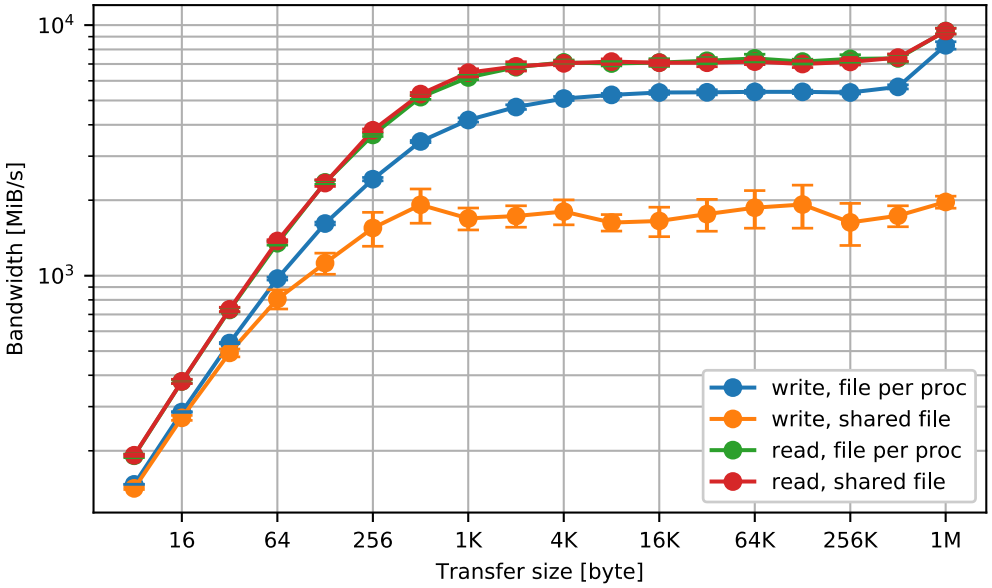

Figure 3-7: IOR bandwidth for shared and individual files without direct I/O for varying transfer sizes

Similar to the measurements of direct I/O we have also measured the performance without direct I/O as shown in Figure 3-7. These tests also use 8 processes distributed over 8 nodes but the block size was increased to 128 MiB resulting in an aggregate size of 1 GiB. The increased block size helped to reduce the standard deviation of the measurements and could be handled easier than in the direct I/O tests, due to the higher overall bandwidth. Another important change is the range of transfer sizes tested. IOR does not allow transfer sizes less than 1 KiB for direct I/O but down to 8 Bytes for buffered I/O. Therefore we also addend the possible sizes below 1 KiB.

The buffered I/O also shows saturation curve with a mostly linear increase for small sizes and a plateau for larger sizes. In contrast to the direct I/O the transition is smoother and the maximum bandwidth is already reached for a transfer size of about 4 KiB. As in previous examples writing to a shared file yields a comparable performance to the unbuffered tests while the other measurements benefit from caching.

3.1.5 Partest

As mentioned in the introduction, this deliverable focuses on large volume I/O. One common scenario is the checkpointing, which is one of the basic use cases SIONlib designed for. Therefore we also determined the performance of partest.

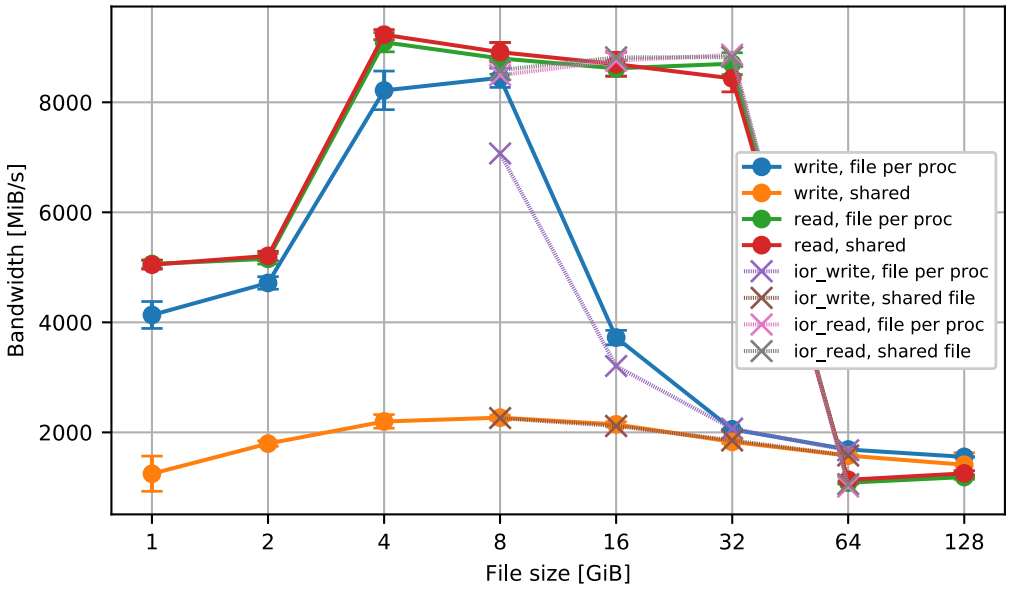

Figure 3-8: partest bandwidth for one shared file and multi-files (one per process)

Figure 3-8 shows the results for the setup

- 8 nodes, 1 ppn
- “bufsize” = 256 MiB (bufsize mostly translates to “transferSize” in IOR)
- chunksize = -1

The chunksize parameter is used by SIONlib to define the amount of data a process intends to write to a file. Using the value “-1” is special to partest (it is not valid for SIONlib) and divides the aggregate size by the number of tasks, so each of them gets one contiguous block of data in the file.

As already pointed out in the introduction it is important to note that these sizes are aggregate sizes and not individual as it is the case for IOR. In order to compare the results of partest with those of IOR we added the corresponding curves marked with crosses.

The previous analyses with IOR apply here as well and we see strong caching effects for reading and the same eviction of the cache for aggregate sizes greater than 32 GiB. As the partest parameters also include aggregate sizes less than 8 GiB we also see results for 1 – 2 GiB, where the cache performance is lower. Writing also shows the same effects, both for shared and individual files. The slightly higher performance of SIONlib for individual files might also be achievable with IOR with the according settings but the more important mode for SIONlib is shared.

Comparing the different measurements, SIONlib’s optimisations do not yield a significant benefit. The results with a normal shared SIONlib file are close to identical, compared to a regular shared file. Improvements of SIONlib (which is intensively tested and optimised with GPFS) to make more efficient use of BeeGFS are ongoing work.

SIONlib offers two different back ends: the default ANSI-C and POSIX. The ANSI-C back end uses `fread` / `fwrite`, file pointers and is usually buffers at least one file system block. On the other hand the POSIX back end uses `read` / `write`, file descriptors (integers) and often does little to no buffering. These were tested on 8 nodes with 1 – 2 ppn and an aggregate file size of 128 GiB. Writing with ANSI-C showed the best performance with about

1440 MiB/s for both numbers of processes. This corresponds exactly to the unbuffered performance of IOR. All other measurements varied between 1000 MiB/s and 1200 MiB/s.

3.1.6 TCP over EXTOLL

In the present setup BeeGFS always uses TCP over EXTOLL to transfer data from compute nodes to the storage servers. We tested the relevance of the availability of native communication over EXTOLL for the I/O bandwidth and found no measureable effect for large scale data transfer. This does not hold for I/O patterns that involve more communication as, e.g. buddy checkpointing with SIONlib.

3.2 Local file systems

3.2.1 Bandwidth overview

The local file system resides on the NVMes of which one is installed on each compute node (CN or BN). Access to the local ext4 file system is possible either directly via /nvme/tmp or via the local BeeGFS on demand (BeeOND). We first of all show a general overview of the I/O bandwidth using IOR.

As well as on the global file system, also on the local file system we found significant cache effects when writing to individual files without direct I/O. Figure 3-9 shows the aggregated write bandwidth for different process counts. The block size is 1 GiB and constant, so the aggregate size increases with the number of processes.

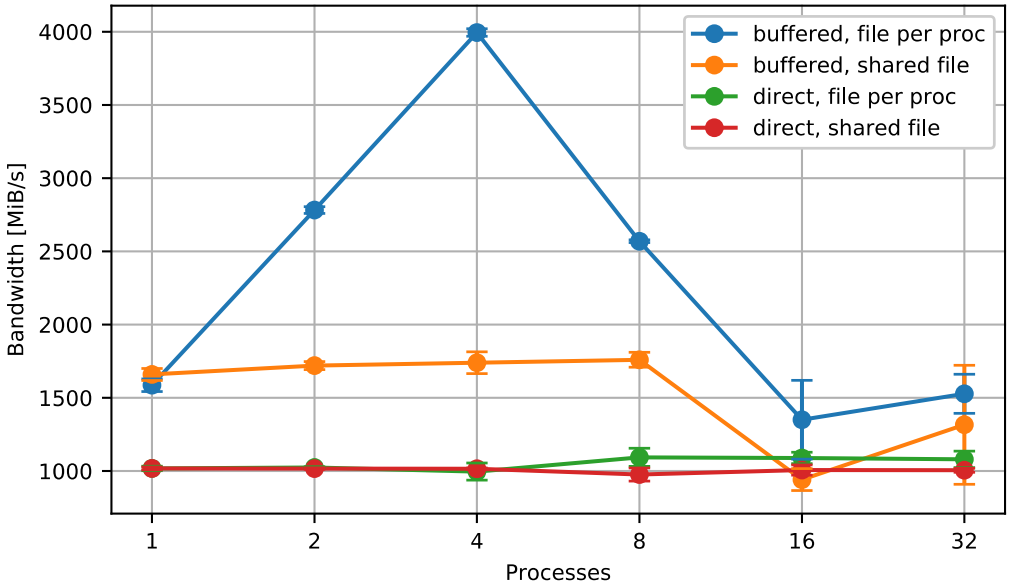

Figure 3-9: IOR write bandwidth for shared and individual files, with and without direct I/O

For both kinds of direct I/O the bandwidth is nearly constant and around 1 GiB/s. In our tests we did not see a simple explanation for the peak of the write bandwidth at 4 processes. The error bars indicate that it is a systematic effect since the values are measured almost constant for each of the 10 iterations.

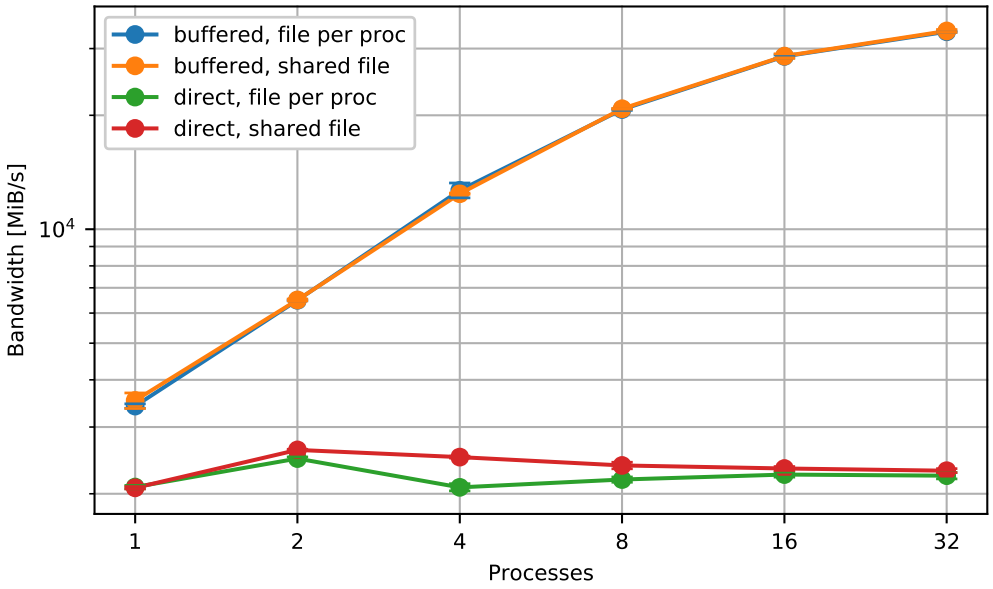

Figure 3-10: IOR read bandwidth for shared and individual files, with and without direct I/O

The read performance shows a more expected characteristic. For direct I/O only varies about 10%. With caching the bandwidth increases linearly with the number of processes up to about 8, where it begins to flatten, most likely due to the exhaustion of the cache or first effects of memory bandwidth saturation (maximum theoretical memory bandwidth for Xeon E5-2680v3 is 68 GiB/s).

3.2.2 BeeOND and "raw" ext4

For a comparison between the write performance directly to the file system on the device (as done in the previous section) and through BeeOND, we kept the aggregate size 4 GiB and constant. This caused the bandwidth for the buffered access to a shared file to stay almost constant as the direct I/O already did for varying aggregate sizes as shown in Figure 3-11.

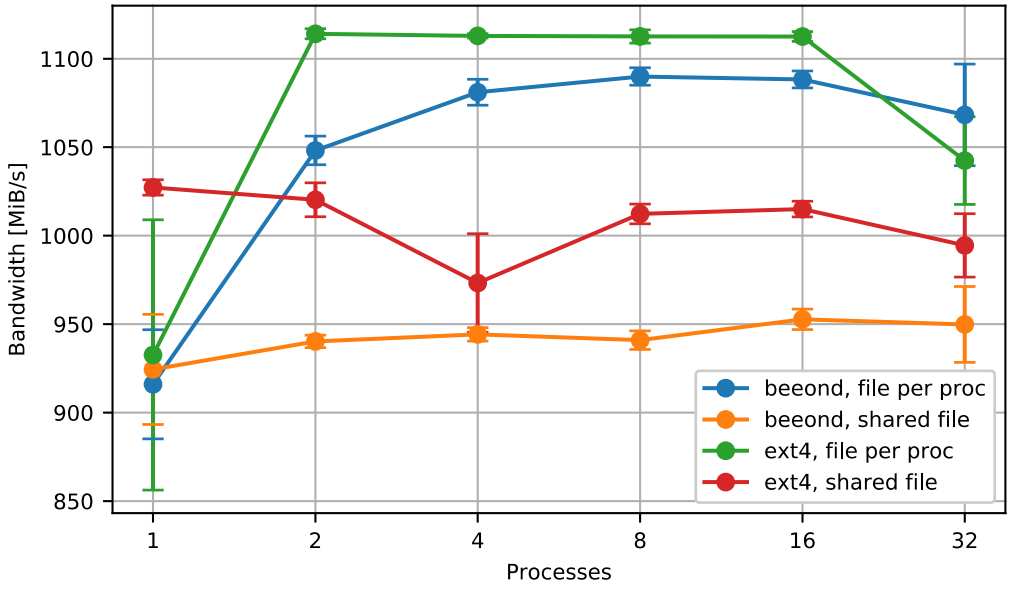

Figure 3-11: IOR write bandwidth to NVMe using direct I/O (ext4 on /nvme/tmp and beeond on /mnt/beeond)

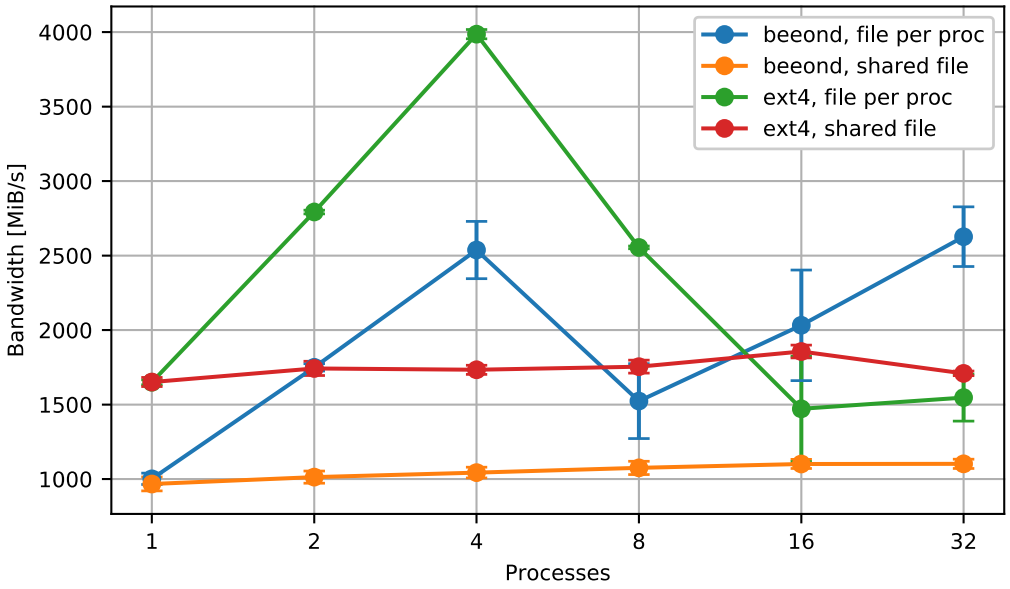

Figure 3-12: IOR write bandwidth to NVMe using buffered I/O (ext4 on /nvme/tmp and beeond on /mnt/beeond)

As in previous measurements, access to individual files allows for more efficient buffering (Figure 3-12). The buffering becomes less efficient when the level of concurrency becomes too high and 8 or more processes use the local file system cache. Since BeeOND operates on top of the local ext4 file system the similar characteristic with a maximum for 4 processes is not surprising. It is worth noting, though, that the maximum bandwidth for up to 8 processes is reduced, the variation increased and the shared buffered I/O shows similar bandwidths as the direct I/O counterparts. Another interesting effect is increased performance for larger process counts. Here the optimisation of concurrent file access in BeeGFS might be accountable. The standard deviation for 4 and 8 processes is larger than in than in the case of directly writing to the NVMe but for 16 and 32 processes it is comparable.

3.2.3 Partest

Write bandwidth showed almost independent of the number of processes. The test writes and reads 4 GiB aggregate data size. As shown in Figure 3-13 there is a general trend towards an increase in bandwidth with processes but the peak is reached at 8 instead of 16 processes. In the measurements for the global file system we saw that partest mostly behaves like shared file I/O in IOR so we also added the IOR measurements for comparison. Those are mostly within the error bars.

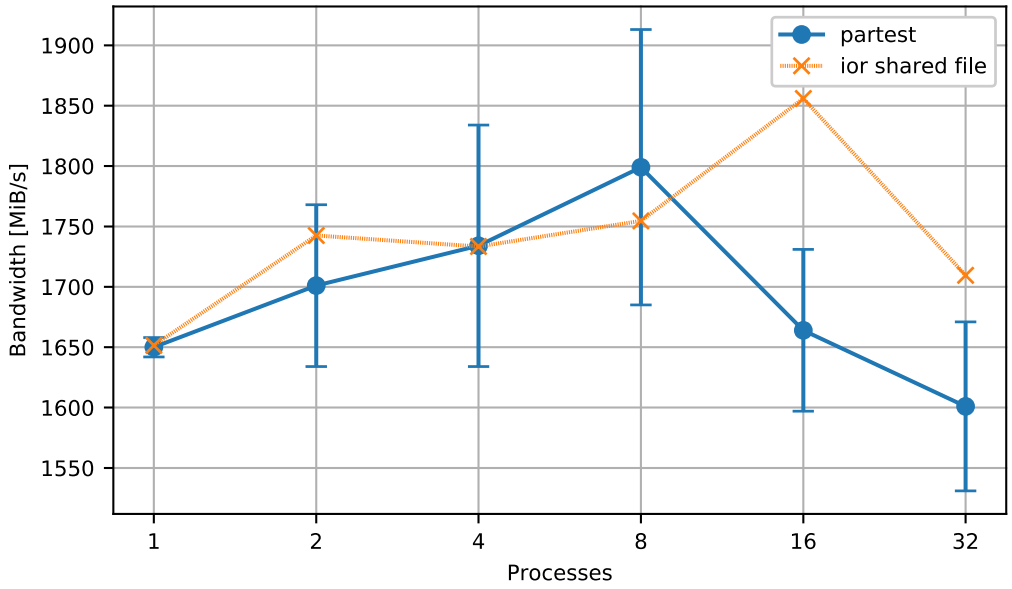

Figure 3-13: partest write bandwidth directly to NVMe (/nvme/tmp)

Although the error bars appear very large, the standard deviation is comparable to the BeeOND measurements and the very narrow limits of the y-axis causes a misleading impression.

The very high peak bandwidth of about 19 GiB/s for reading, as shown in Figure 3-14, might be caused by the larger “bufsize” (256 MiB instead of 32 MiB transfer size for IOR) of single read and write operations. We stress again, that these cache effects do not represent the sustained bandwidth, especially since the limit for the NVMe over PCIe is about 4 GiB/s.

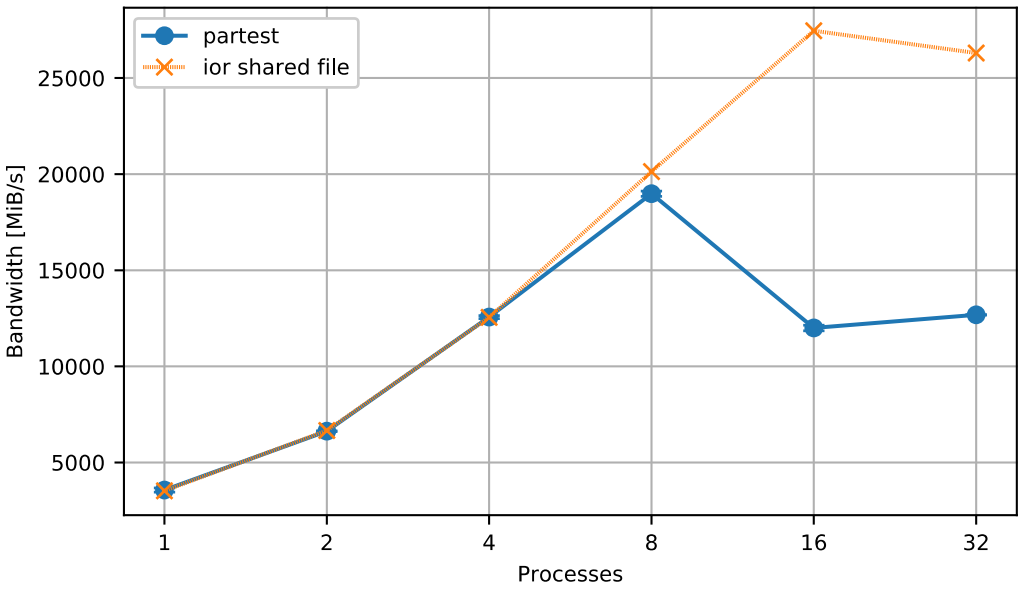

Figure 3-14: partest read bandwidth directly to NVMe (/nvme/tmp)

4 Benchmarks on KNL

The single core performance of the BNs is lower when compared to the CNs and this directly impacts the I/O performance as well. Depending on the benchmarking setup this may shift the bottleneck from the storage or network to the compute part. In the following sections we show and analyse the results for access to the global and local file system.

4.1 Global file system

4.1.1 IOR

Compared to the SDV's performance, the KNL shows a similar bandwidth for writes of large file sizes to individual files. For other configurations the results differ, some of them significantly.

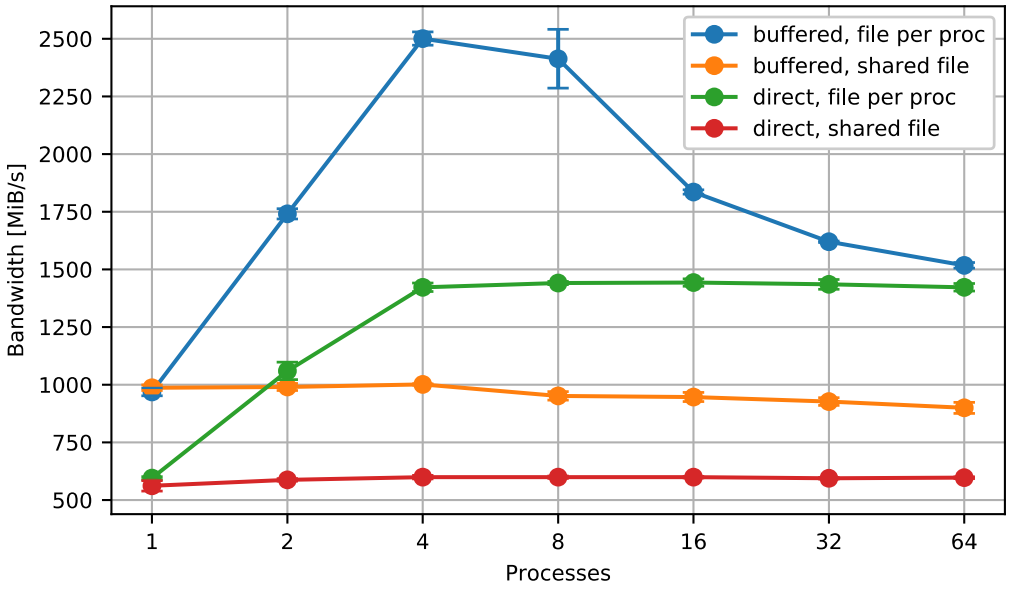

Figure 4-1: IOR write bandwidth to global file system from one KNL

Figure 4-1 shows the results measured on a single KNL node writing to the global file system. This and the following tests use a block size of 1 GiB, so the total volume increases with the number of processes. The bandwidth for a shared file with buffered I/O decreases slightly with increasing number of processes but is almost independent of the number of processes. Similarly direct I/O to a shared file yields almost constant bandwidth, as does direct I/O for 4 or more processes but on significantly higher level. The initial ramp up phase might be related to the rather weak single core performance in the KNL. For up to 4 processes the bottleneck is the throughput of the cores and beginning with 4 processes other resources are the limiting factor. This is consistent with the fact that the buffered I/O converges towards the same bandwidth as direct I/O

In order to determine which effects result from using a single node Figure 4-2 shows the same measurement from two instead of one KNL. The parallel execution uses the "-F" flag to the MPI application launcher (mpiexec) to distribute processes over nodes first before increasing the number on a single node. This way the second process immediately adds the other node and the step to 4 processes increases the number of processes per node to 2. Doing so results in the first increase in bandwidth for shared file I/O from one to two

processes. In contrast to the test with a single KNL, all measurements but direct I/O to a shared file show a similar asymptotic bandwidth for large data sizes and process counts. The direct I/O to a shared file also benefits from the additional node and almost doubles from about 600 MiB/s to 1100 MiB/s. Buffered I/O to individual files shows a close to ideal scaling for in the range in which the data (mostly) fits into the cache.

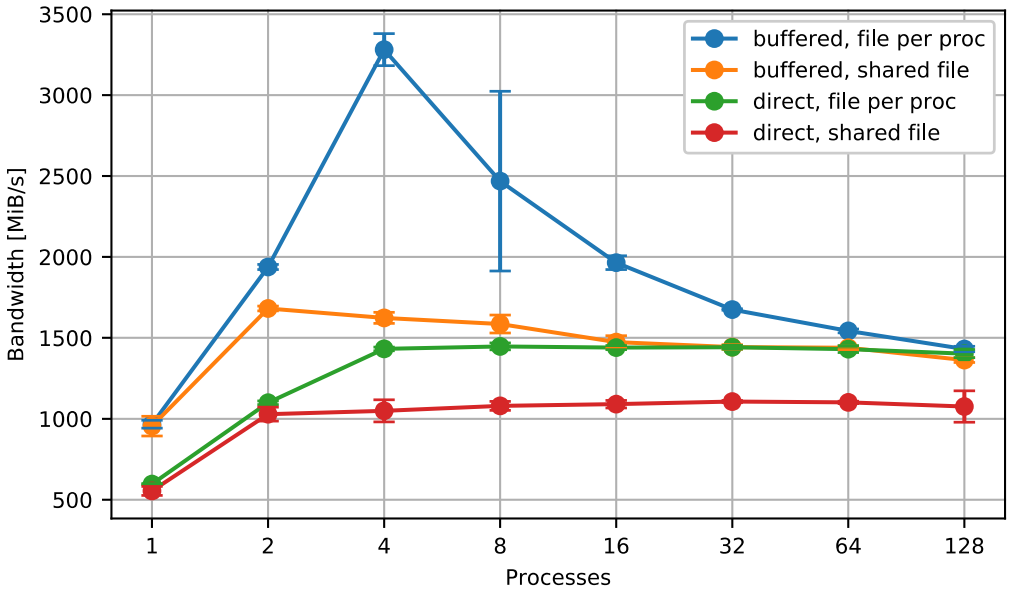

Figure 4-2: IOR write bandwidth to global file system from two KNLs

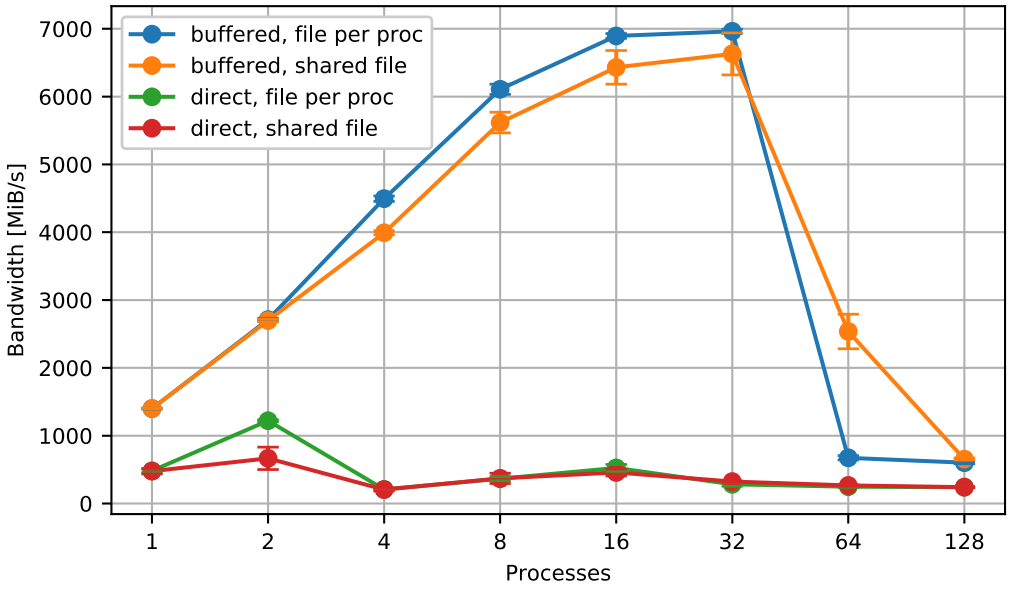

Figure 4-3: IOR read bandwidth to global file system from two KNLs

The read bandwidth to the global file system again shows a strong caching effect for buffered I/O. Figure 4-3 shows the results for two KNLs. When comparing these measurements with those of the SDV we again see the drop in bandwidth for 64 GiB aggregate file size, although in the KNL measurements it takes up to 128 GiB to completely eliminate the caching effect. For direct I/O we get a significantly lower performance than on the SDV, mostly over all process counts.

For a single KNL node the results (not shown) of direct I/O are generally a little faster, but in a similar range of roughly 300 MiB/s. The buffered bandwidths on the other hand are marginally slower.

4.1.2 Partest

The results for partest from a single KNL to the global file system are similar to those of IOR for shared buffered files for small process counts. For larger process counts the bandwidth decreases down to its minimum with 16 processes and then increases again.

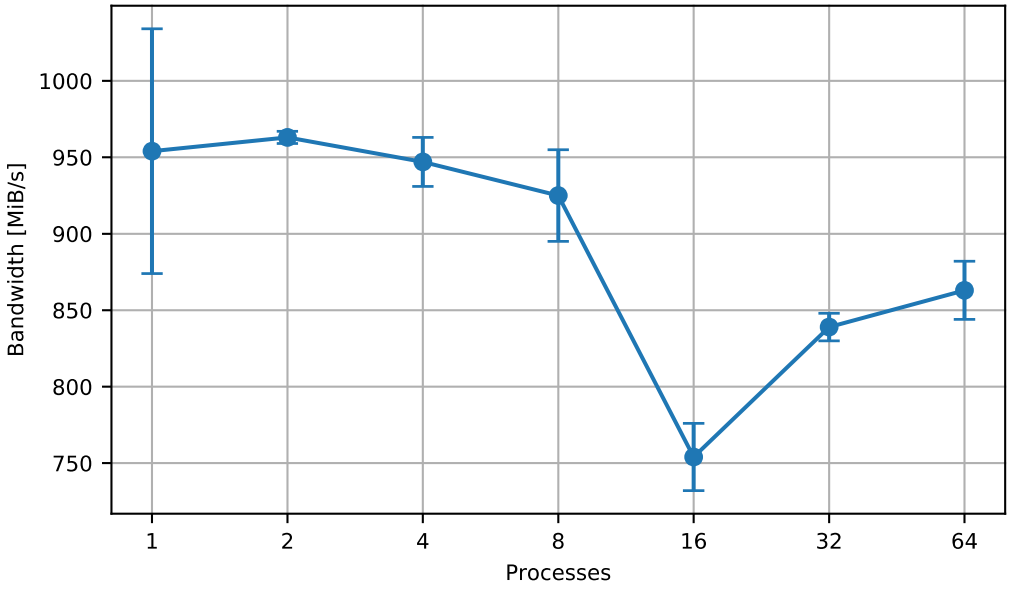

Figure 4-4: partest write bandwidth to global file system

In contrast to the IOR measurements the partest was measured with a constant aggregate size of 4 GiB.

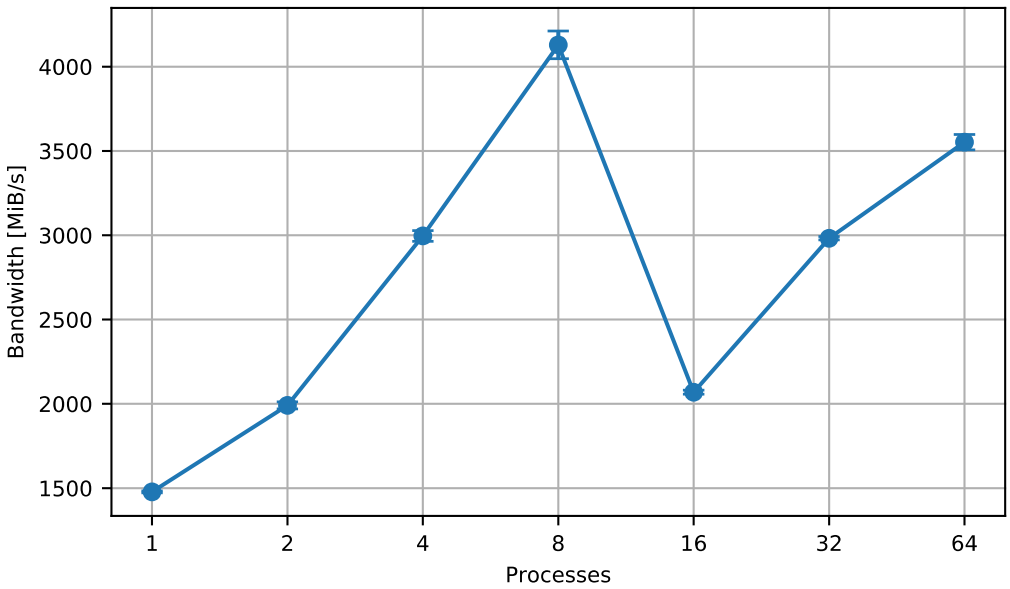

Figure 4-5: partest read bandwidth to global file system

The read bandwidth for the same settings (Figure 4-5) shows the same drop in performance for 16 processes as the write bandwidth and has a similar characteristic as the write bandwidth to individual files to the local NVMe through BeeOND (see Figure 3-12). The small standard deviation suggests a systematic reason, but in our experiments we could not find a clear answer.

4.2 Local file systems

4.2.1 IOR

The general performance of IOR when writing to the local storage of a KNL is comparable to the one of an SDV node. Figure 4-6 shows results for a block size of 1 GiB. For shared file access the performance is a little weaker in the case of direct I/O. The most prominent exception is buffered I/O to a shared file which shows the opposite behaviour, compared to the SDV: on the latter buffered I/O largely outperforms direct I/O, where on KNL direct I/O yields significantly higher bandwidths. This can be interpreted as follows: direct I/O with individual files has the same bandwidth as the SDV when overcoming the limited single thread performance. For direct I/O with a shared file the additional coordination between the processes on the KNL is the limiting factor, again due to the poor single thread performance. Similarly the buffered measurements suffer from the poor single thread performance, compared to the SDV.

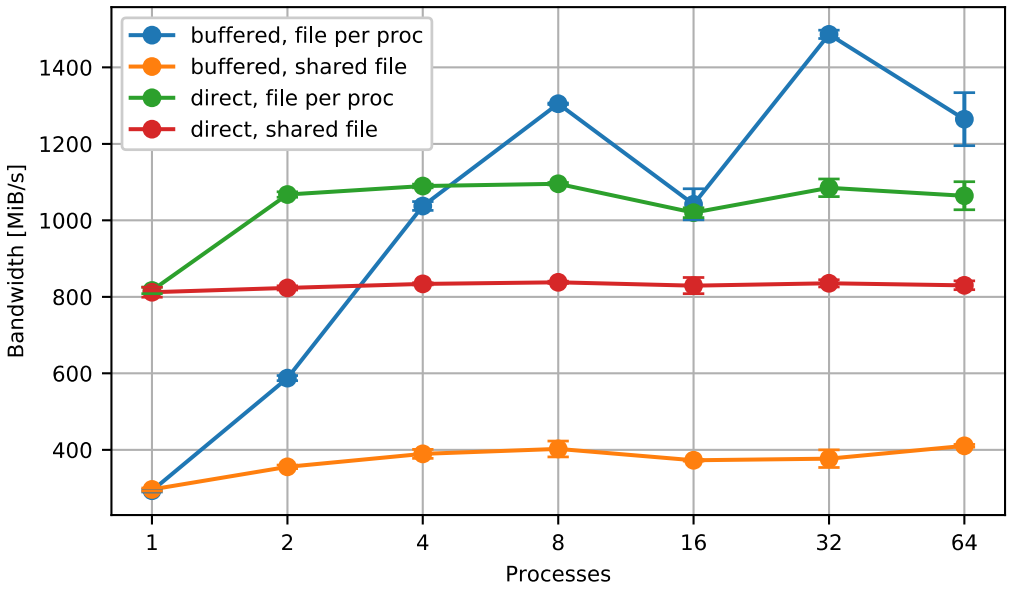

Figure 4-6: IOR write bandwidth to local file system

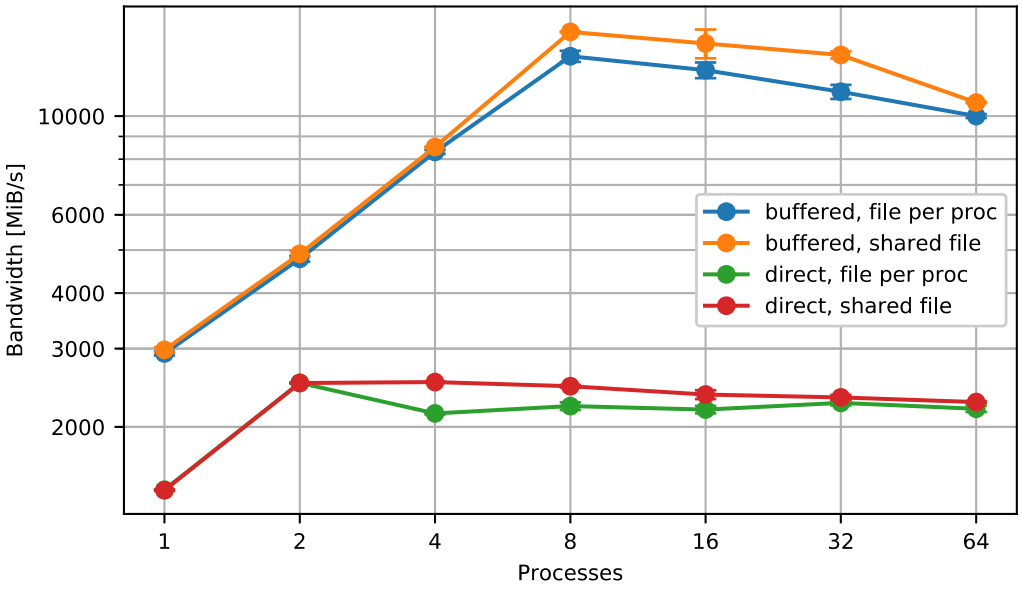

Figure 4-7: IOR read bandwidth to local file system

The read bandwidth (Figure 4-7) on KNL does not show a smooth saturation like on the SDV. It is limited rather abrupt for 8 processes and declines for larger process counts. For direct I/O the performance is similar to the one measured on the SDV. As on the SDV, the buffered reads are mostly read from local RAM, which explains the bandwidth beyond the theoretical limit to the NVMe of approximately 4 GiB/s over PCIe.

4.2.2 Partest

For partest, the bandwidth for different process counts and different sizes were measured separately. Figure 4-8 shows the write bandwidth for varying process counts and the two supported back ends. The aggregate file size is 32 GiB and constant for these measurements. For an easy comparison, the results from IOR measurements are added with crosses. Both benchmarks yield very similar results. The ANSI C back end shows better performance than POSIX, although the effect gets smaller for larger process counts. The slightly better performance of IOR might be caused by caching effects, since these measurements kept the data per process constant, which results in less data being transferred for process counts less than 32.

Reading is dominated by cache effects and is mostly independent of the choice of back end in SIONlib (ANSI-C or POSIX) or benchmark (partest or IOR) itself, so they are not shown.

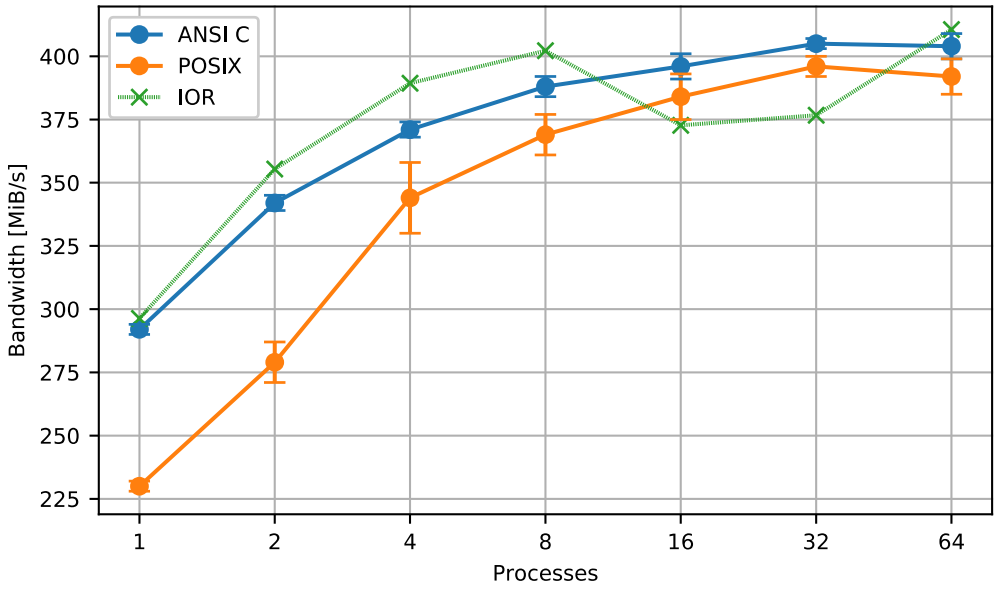

Figure 4-8: partest write bandwidth against processes (IOR with crosses)

With a fixed number of 8 processes we also measured the bandwidth for different file sizes. Figure 4-9 and Figure 4-10 show the write and read results, respectively. For sizes of 4 - 64 GiB the bandwidth is higher than for larger and significantly higher than for smaller file sizes. Reduced performance for larger sizes can often be explained with cache saturation, but the bandwidths in this case are rather slow and this does not explain the lower bandwidths for smaller file sizes.

The reading case shows a classic cache characteristic for larger values. The drop in the bandwidth from 64 to 128 GiB aggregate file size indicates the size of the relevant cache.

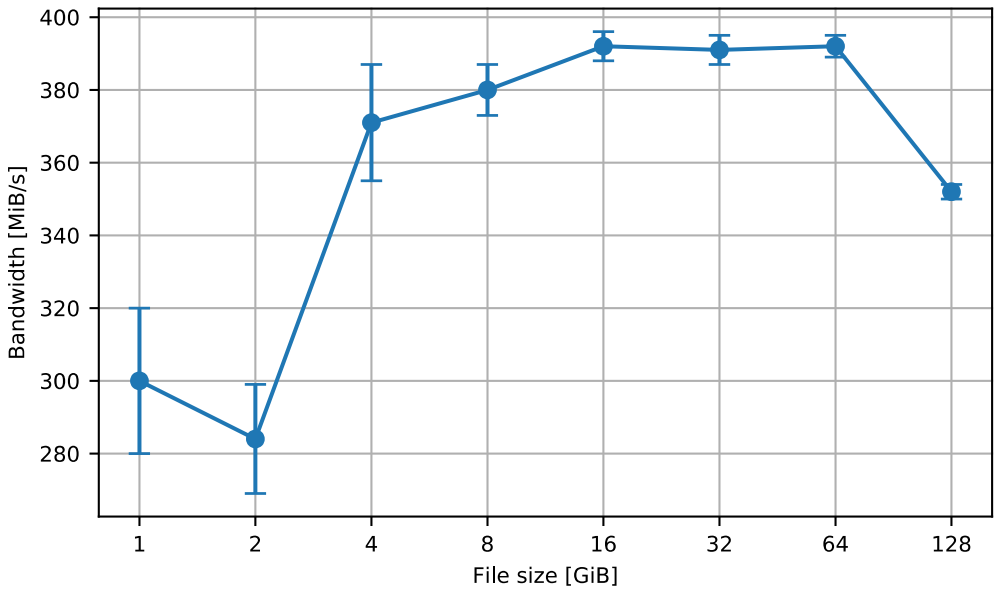

Figure 4-9: partest write bandwidth against file sizes

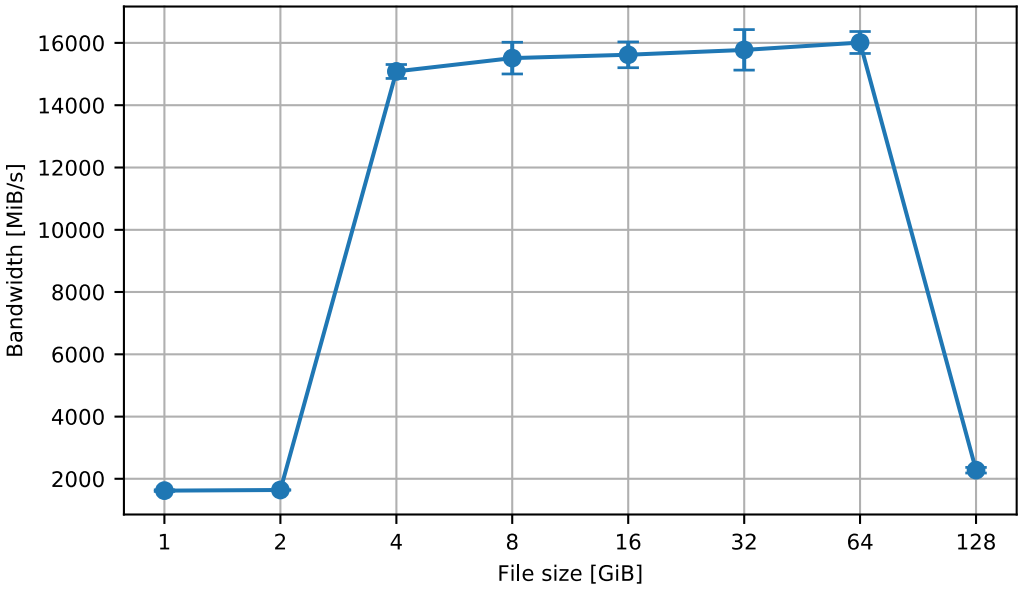

Figure 4-10: partest read bandwidth against file sizes

5 Continuous benchmarks

As planned for Task 4.5, by the time the basic hardware was available, continuous benchmarks were prepared. By the end of February 2016 the measurements started on the SDV and similar measurements for the predecessor system DEEP were implemented beginning of May 2016 for comparison. If there were no technical issues which prevented the tests from being run they were executed once a week and this section shows the results until the time of writing this deliverable.

The DEEP-ER platform is the one described in section 2.1. With continuous tests changes to the system could be documented and the results are shown in the following sections. As the KNL nodes arrived significantly later than the SDV, only long term measurements for the SDV are available.

5.1 IOR

As I/O is one of the key elements in the DEEP-ER project, the bandwidth to the different file systems is an important quantity. Similar to the previous sections we used IOR to measure I/O bandwidths.

Figure 5-1 shows the development of the IOR performance from March 2016 until the time of writing this deliverable in March 2017. This is the time frame for which regular benchmarks were prepared and running. The constant parameters of these measurements are

- POSIX API
- fsync (after read / write operations)
- 24 GiB file size
- 1 node 24 processes

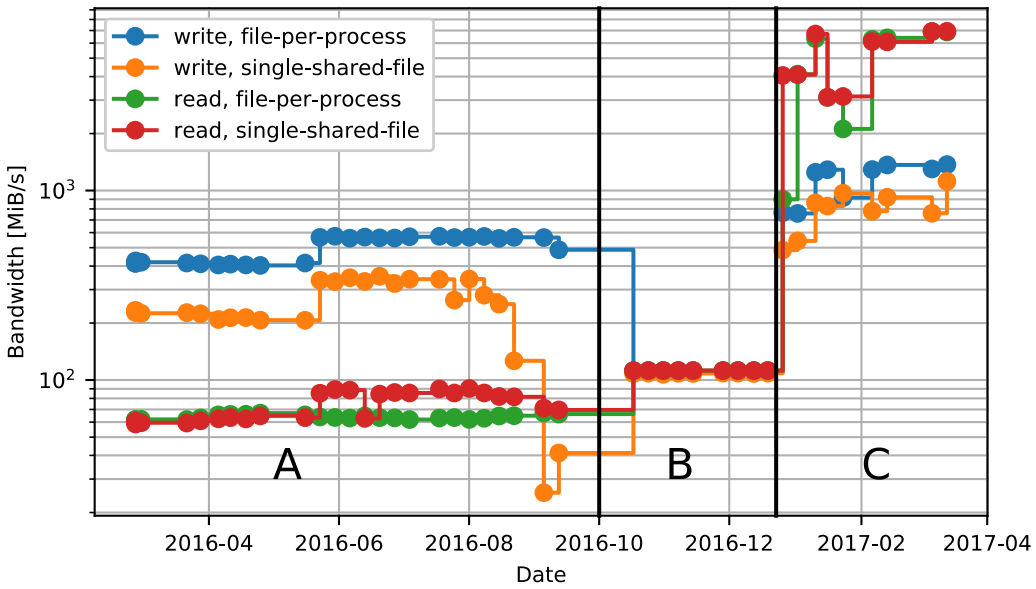

Figure 5-1: IOR bandwidth on /sdv-work over time (POSIX)

The aggregate size of 24GiB is generally rather small for the purpose of I/O benchmarking, but with the degraded performance at the beginning of the measurements larger file sizes would have resulted in a drastic increase of time needed for benchmarking.

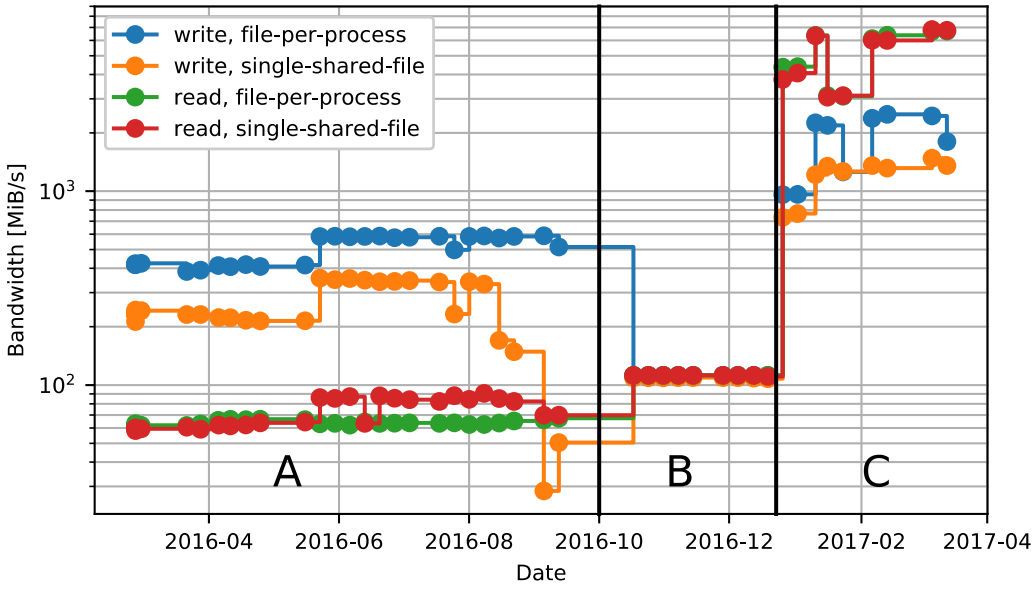

Figure 5-2: IOR bandwidth on /sdv-work over time (MPI-IO)

Similarly Figure 5-2 shows the same measurements for the MPI-IO API. Given the slight variation of values in general, the results are the same and do not show any significant influence of the choice of the API until January 2017 (“A” and “B”). At this time (“C”) the write bandwidth for MPI-IO becomes higher than for POSIX (approximately 2400 MiB/s vs. 1300 MiB/s for individual files and 1300 MiB/s vs. 900 MiB/s for a single shared file). On this scale (1 node, 24 processes) this is the expected behaviour as it is far below the scale were optimisation in MPI-IO or file creation time plays any role. Properly used MPI-IO with large enough file access (compare section 3.1.4 “Transfer sizes”) usually results in the same bandwidth as POSIX shared file I/O for small scales and in this example the small aggregate file size of 24 GiB seems to be buffered better in MPI-IO than in POSIX.

For both APIs (POSIX and MPI-IO) the update of the EXTOLL drivers for better IP over EXTOLL support (“C”) shows a major performance increase. This especially holds for the read performance which suffered most from the previous implementation. After the update the performance increases by almost two orders of magnitude from initially almost 60 – 90 MiB/s to 2 – 6 GiB/s, where letter also includes cache effects, according to the data presented in the previous sections. For the time from the end of October 2016 to beginning of January 2017 (“B”) all values for read and write independent from the API and the file access are constant right above 100 MiB/s. In this time frame the network was set to use the gigabit Ethernet connection, as a temporary circumvention of the very weak read bandwidth when using IP over EXTOLL (which was fixed beginning of 2017). For this setup the network bandwidth defined the bottleneck for the I/O bandwidth. Another important change between September and October is an update of the OS version on the file servers to match those on the KNLs (CentOS 6.4 to CentOS 7.2). There were also NICs of one storage server fixed in this period of time.

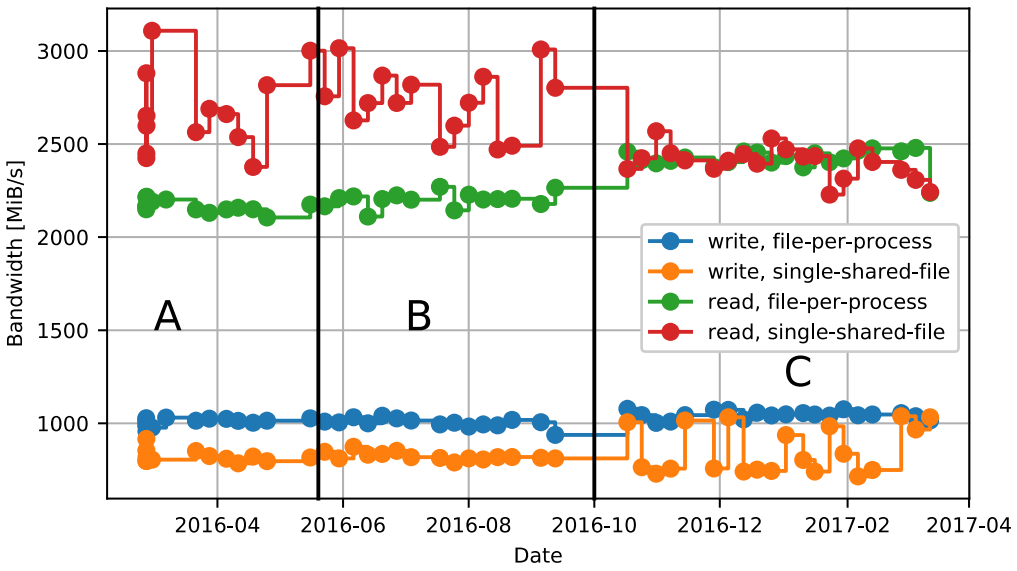

Figure 5-3: IOR bandwidth on /nvme/tmp

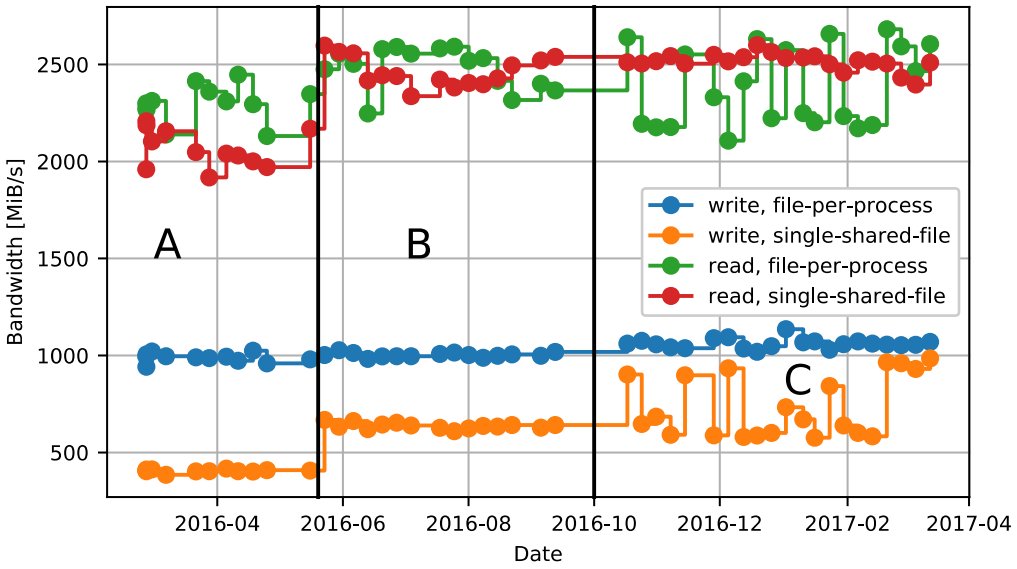

Figure 5-4: IOR bandwidth on /mnt/beeond

5.1.1 Local file systems

Figure 5-3 and Figure 5-4 show the I/O performance to the local storage using the local ext4 file system and the BeeOND, respectively. One major difference in the setup of the tests of the local file system is the file size of 240 GiB, which is ten times as large as for the global file system. The main reason for this is the higher bandwidth that was available at the time the benchmarks were set up.

As can be seen there are no changes comparable to the ones on the global file system. As described in the previous sections, although the `fsync` parameter is used, there can be still variations on the performance due to caching effects. The variation is generally stronger for read operations. Although by far less pronounced there are generally three different regions, which are before end of May 2016 (“A”), before mid October 2016 (“B”) and afterwards (“C”). In May (“A” → “B”) some larger changes to the images of SDV and KNL were made which

likely influenced the performance. Between the last data point in September and the next one in October (“B” → “C”) not only EXTOLL was updated but different software updates were applied. This is a possible explanation for changes in the performance when writing to local storage, which re independent of the network (and hence EXTOLL).

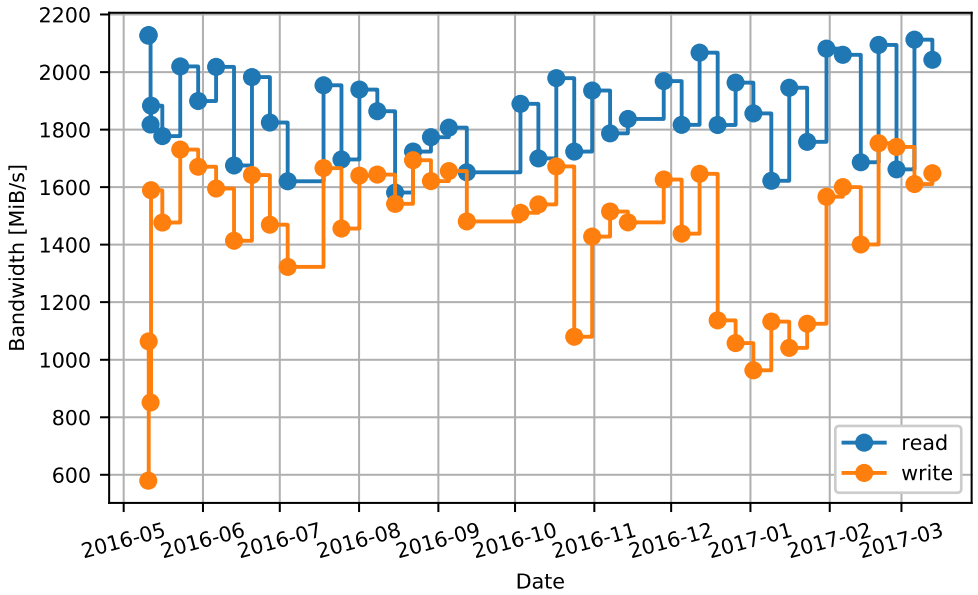

Figure 5-5: IOR bandwidth on the DEEP cluster (/work)

For comparison Figure 5-5 shows the development of the IOR performance on the predecessor system DEEP. Although there are (in part strong) variations in the values, there are no pronounced plateaus or similar features.

5.2 Mdtest

In addition to the I/O bandwidth, we also measured the metadata performance, i.e. the file creation and removal rate. Although a high metadata performance is generally considered less crucial in the DEEP-ER project, we monitored it to be able to distinguish actual I/O bandwidth problems from metadata problems. If, e.g. file creation takes very long, the overall bandwidth measured for small size I/O would drop.

Similar to the measurements with IOR, mdtest shows strong changes after the update of IP over EXTOLL and other software updates. The setup for these benchmarks also uses a single node with 24 processes. For the time at which networking was configured to only use Ethernet, Figure 5-6 already shows drastic improvement in the remove rate, which mostly corresponds to the previous file creation rate. The latter, in contrast, is lower after the update than before, but still roughly three times as high as the previous removal rate. As both, file creation and removal are metadata operations in directories, there is no general reason for different performance. Depending on the implementation this balance can be shifted to the benefit of either of both. The sporadic drops in the remove rate around February 2017 coincide with a disk failure and replacement.

Regarding the metadata performance on the local storage BeeOND (Figure 5-8) performs roughly twice as good as on the global storage. The software and system changes which lead to the improvement of the remove rate on the global file system do not carry over to the local file system where both, the remove and the creation rate loose performance. In its latest

configuration the local ext4 file system (Figure 5-7) shows 5 to 15 times higher rates as compared to BeeOND. The rate of 25k to 30k for file creation and removal respectively, is a known limit for the metadata server of BeeGFS. This can be circumvented by using multiple metadata servers on the same block device to exploit its performance. This effect would only be relevant for the local storage due to its very high performance. For the global file system the dominating bottleneck could not be clearly determined. The drives (2× 200 GiB SSD RAID 1 on the metadata server) should be able to provide comparable performance to the local NVMes of the compute nodes. A plausible candidate is the network latency, since metadata operations are highly sensitive to latency as opposed to bandwidth.

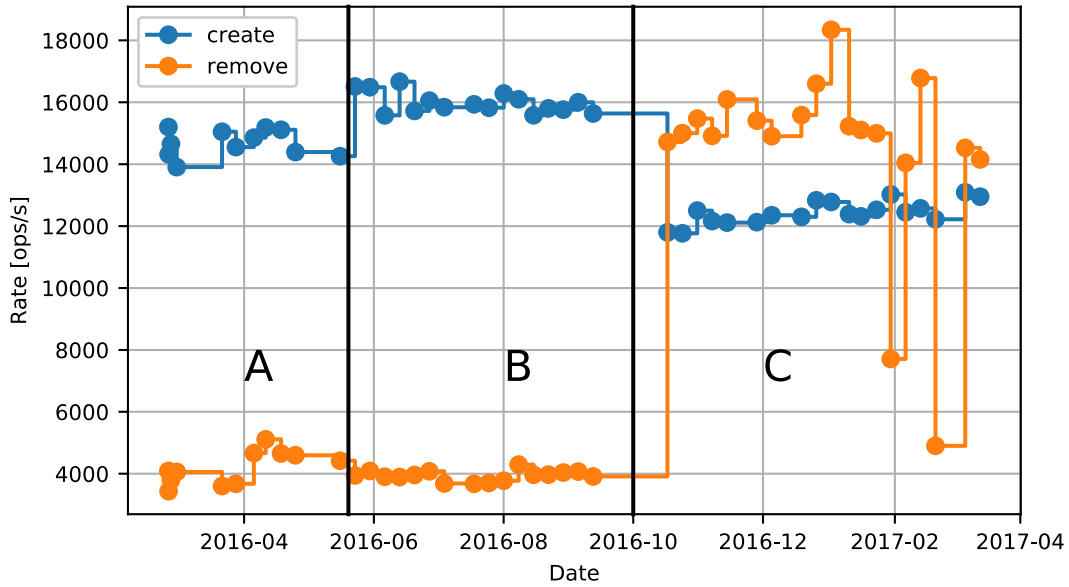

Figure 5-6: mdtest rates on /sdv-work over time

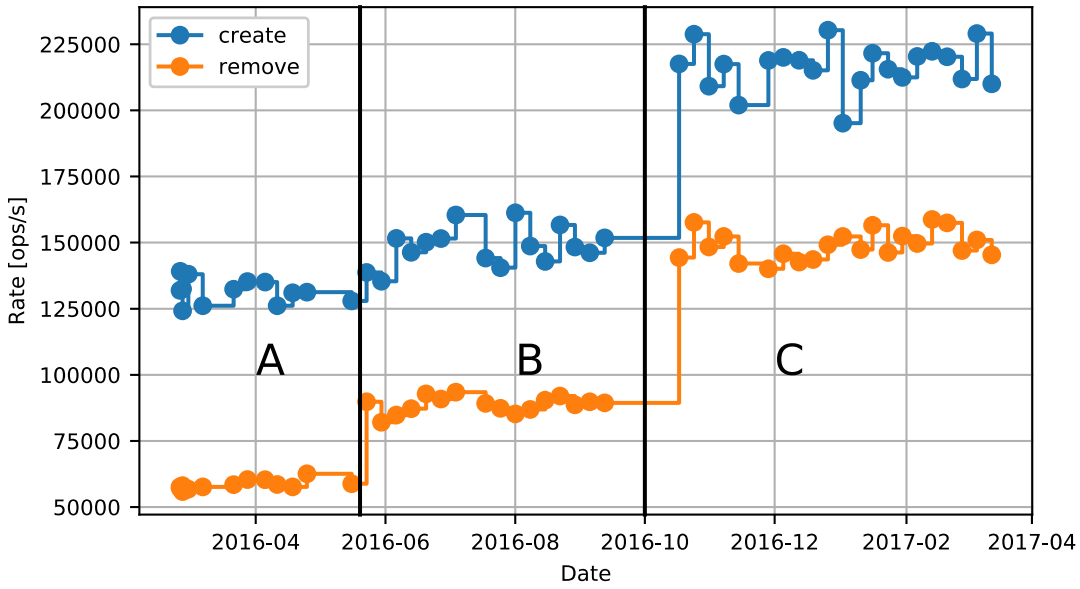

Figure 5-7: mdtest rates on /nvme/tmp over time

As for the results of the IOR measurements we also continuously observed the metadata performance of the DEEP system, which is shown in Figure 5-9. As expected these values do not vary significantly.

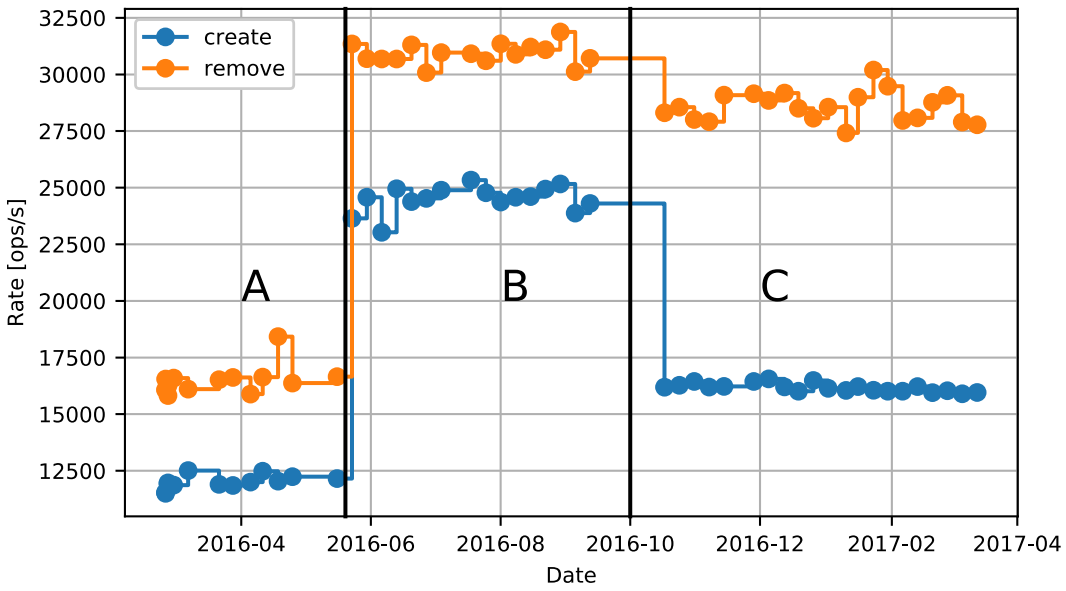

Figure 5-8: mdtest rates on /mnt/beeond over time

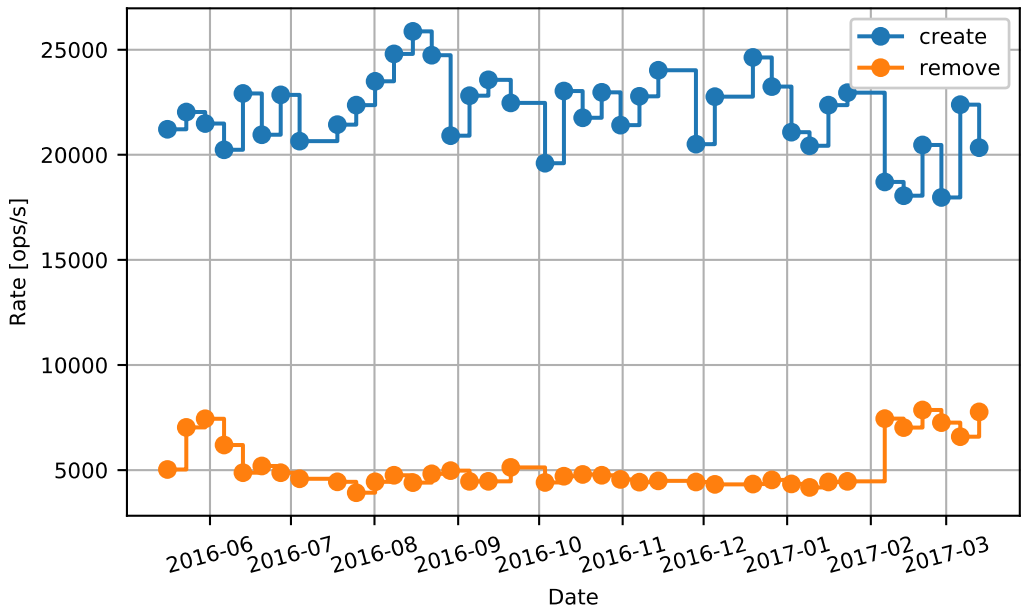

Figure 5-9: mdtest rates over time on DEEP cluster (/work)

6 Summary

In this deliverable we presented the results of the performance measurements of the DEEP-ER I/O systems. We analysed similarities and differences between I/O behaviour on the Cluster and on the Booster, both for global and local storage. To give a general placement of these results we also compared them with those of the predecessor system DEEP. For most of the characteristics possible explanations could be derived while some results still need further analyses.

Generally, the results show very dominant caching effects for reading if not circumvented explicitly by the benchmarking tools.

The continuous benchmarks show in part very large changes of the system over time, most notably the improvements of IP over EXTOLL which is the foundation of all network operations of the global BeeGFS filesystem. This stresses the importance of capturing as much of the system's state as possible in order to get comparable results.

Common maximum bandwidths to for the actual storage access of the global file system are around 1.5 GiB/s for writing and 1.1 GiB/s for reading. Local files systems show a similar write bandwidth.

References

D4.3 “Definition of test cases and patterns”

D5.4 “Resiliency performance measurements”

D6.3 “Final report on applications experience”