Analysis of high capacity storage systems for e-vlbi

Matteo Stagni - Francesco Bedosti - Mauro Nanni

May 21, 2012

IRA 458/12

Abstract

The objective of the analysis is to verify if the storage systems now available on the market for a reasonable price (COTS) can satisfy the storage needs for vlbi observations and postponed e-vlbi.
Radioastronomic observations require the sizes of high capacity systems to be in the order of tenths of Terabytes. These systems should handle the storage of data streams at the rate of 1 - 2 - 4 - (8) Gbit/s. The streams generated during 'observation scans' create files which sizes have the average dimension of some gigabytes, but they could also rise up to tenths of gigabytes.
An observative run today lasts up to 20 days, during this time an average of 100 TB per antenna is produced and data could be retained on the tanks for about 100 days, consequently it is transferred to the correlator and then deleted from the tanks.
Provided these conditions , our analysis has taken into consideration SuperMicro systems equipped with 3Ware RAID cards. We tried to define if these systems could handle the workload foreseen, testing different RAID sets and different filesystems, under the condition of huge sequential files writing and reading acting in an environment where this was the only task running.
We expect these systems to be dedicated for a single antenna in a one to one relationship. During the acquisition time there should be no other activities that could interfere significantly on file writings, so our tests were deployed in this environment.

1 Tech Specs

We have purchased two types of SuperMicro systems. The first one, as summarized in Table 1, is a Super-Storage Server RS24S2-24TB INTEL 4U Rackmount with 24 slots. It sports 2 XEON Quad Core processors E5620 @ 2,40GHz 5,86GT/s SKT1366 with 12 MB cache, 12 GB of DDR3 RAM @1333MHz, a 3Ware RAID controller 9650SE with 24 ports enabled up to SATA2 specs and an INTEL 10 gigabit FIBER PCIE network adapter. Hard drives purchased for storage purpose are Hitachi HDS722020 SATA2. The operative system used is Scientific Linux 6.0. This system is now in production for Alma as a storage server and there wasn't enough time as for the second one to perform a thorough test session. Because of this the comparison we are able to produce between the two systems is limited to the short preliminary test described later on.

The second one, an RS48S2, has the same specifics as the first, but it differs from the type of motherboard which supports SATA3 specs and the 3Ware RAID controller 9750DR that can support up to 24 drives, and the operative system running is Scientific Linux 6.2. Hard drives purchased for storage purpose are Hitachi HDS723020 SATA3. All of the SATA drives are 2 TB size.

N.	OS	CPU	RAM	Motherboard	3WARE RAID card
1	Scientific Linux 6.0	2 Intel XEON E5620 @2.4GHz	12GB	X8DTL-IF	9650SE
2	Scientific Linux 6.2	2 Intel XEON E5620 @2.4GHz	12GB	X8DTH-IF	9750DR

Table 1: Systems configurations:
1. Motherboard X8DTl-IF supports up to 2 PCI - express with 8 SATA2 drives @ 300 MB/s
2. Motherboard X8DTH-IF supports up to 7 PCI - express with SATA3 drives @ 600 MB/s

1.1 Systems comparison

We have first run a quick preliminary test in order to evaluate the differences between the two systems considered. Tests were deployed using the command:

dd of=pippo.dat if=/dev/zero bs=1M count=20100

We tested different RAID and filesystem configurations:

As seen from Table 2, tests run under configuration n. 1, we reach a good performance in writing under native Ext4 filesystem, especially when the array is made up of a single drive, 3 drives or 6 drives, whereas under configuration n.2, using XFS filesystem we obtain a weaker performance in the smaller arrays, but there is an overwhelming performance difference when we take into account larger arrays such as 12 or 24 disks.

Further on we decided to pursue our tests tanking into account only configuration n.2, in order to get the best per performance possible. So the next step was to decide which kind of RAID array to use.

In the next paragraphs we would like to remind the differences between the main raid sets considered viable for our tests.

RAID 0 (block-level striping without parity or mirroring) has no (or zero) redundancy. It provides improved performance and additional storage but no fault tolerance, so a single drive failure destroys the entire array. This is not good because we would like to have a minimum fault tolerance, especially during observing sessions.

RAID 5 (block-level striping with distributed parity) distributes parity along with the data and requires all drives but one to be present to operate; the array is not destroyed by a single drive failure. However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt.

There was a doubt about whether to use RAID0 or RAID5, because we wanted to know if there was a noticeable performance difference in a writing task. As we found out in our preliminary tests, later on confirmed by further testing, there is not a significative difference in performance between the two. So our choice was finally RAID5.

		Config. 1 on Ext4	Config. 2 on XFS	
Disk N.	**RAID**	**MB/s**	**MB/s**	**Notes**
1	JBOD	130	91	
3	RAID5	250	207	Balanced
6	RAID5	450	545	Balanced
12	RAID5	600	1100	Balanced
24	RAID5	655	1200	Balanced
24	RAID0	680	1200	Balanced

Table 2: Systems configurations test results

2 Filesystems

Once the choice on RAID had been done, we had to face a subsequent choice about which was the best performing filesystem. The filesystem we decided to run tests on were Btrfs, a new filesystem for linux under development, Ext4, the native filesystem for linux, JFS, a filesystem developed by IBM and XFS, a filesystem developed by Silicon Graphics.

2.1 Btrfs

Btrfs (B-tree file system) is a GPL-licensed copy-on-write file system for Linux. Development began at Oracle Corporation in 2007.

Btrfs is intended to address the lack of pooling, snapshots, checksums and integral multi-device spanning in Linux file systems, these features being crucial as Linux use scales upward into larger storage configurations. These are interesting features because in the future we might need to increase the storage space easily and quickly.

Max. file size	Max. volume size	single dd file writing speed
16 EiB	16 EiB	347 MB/s

Table 3: Btrfs logical limits

Sadly, due to a poor implementation, at the moment Btrfs presents several problems. The main one is shown in Figure 11 on Page 12. We ran a test with a batch script to write fully a 12 2TB disk RAID5 array with the command **dd of=pippo.dat if=/dev/zero bs=1M count=20100** with a 2 seconds delay in-between the writings. After 354 writings we received an output telling us the 21TB partition was completely written. Moreover the iozone test run(Page 14 Figure 15) shows that also the reading speed is not satisfactory.

2.2 Ext4

The ext4 or fourth extended filesystem is a journaling file system for Linux, developed as the successor to ext3. Kernel 2.6.28, containing the ext4 filesystem, was finally released on December 2008.

Ext4 uses checksums in the journal to improve reliability, since the journal is one of the most used files of the disk. This feature has a side benefit: it can safely avoid a disk I/O wait during journaling, improving performance slightly.

Even though the declared limits in Table 4 should be enough for our needs, while performing our tests we found out that it was impossible to overcome the actual implementation of volume size limit on linux of 16 TB. Even though the performances were promising as shown in Figure 12 on Page 12 and Figure 16 on Page 14 regarding the iozone test, we were not pleased with this limitation.

Max. file size	Max. volume size	single dd file writing speed
16 TiB	1 EiB	948 MB/s

Table 4: Ext4 logical limits

2.3 JFS

Journaled File System or JFS is a 64-bit journaling filesystem created by IBM. An implementation for the Linux kernel is available as free software under the terms of the GNU General Public License (GPL). JFS' journaling is similar to XFS where it only journals parts of the inode.

According to reviews and benchmarks of the available filesystems for Linux, JFS is fast and reliable, with consistently good performance under different kinds of load, contrary to other filesystems that seem to perform better under particular usage patterns, for instance with small or large files. Another characteristic often mentioned, is that it's light and efficient with available system resources and even heavy disk activity is realized with low CPU usage.

Max. file size	Max. volume size	single dd file writing speed
4 PB	32 PB	845 MB/s

Table 5: JFS logical limits

JFS, compared to the two previous filesystems tested, was the first filesystem to complete the task of fully writing the volume size of 21 TB on the 24 disks RAID5 array on the dd test, as shown in Figure 13 on Page 13. Although JFS is slower compared to the Ext4 performance in writing, it had the best results in the reading task during the iozone test as shown in Figure 17 on Page 15.

2.4 XFS

XFS is a high-performance journaling file system created by Silicon Graphics, Inc. XFS is particularly proficient at parallel IO due to its allocation group based design. This enables extreme scalability of IO threads, filesystem bandwidth, file and filesystem size when spanning multiple storage devices.

The filesystem was released under the GNU General Public License in May 2000 and was first merged into the mainline Linux kernel in version 2.4 (around 2002), making it almost universally available on Linux systems.

Max. file size	Max. volume size	single dd file writing speed
8 EiB	16 EiB	1053 MB/s

Table 6: XFS logical limits

XFS on paper had excellent prerequisites that were later confirmed by our tests. In fact the dd test we made on XFS shown in Figure 14 on Page 13 produced the best results of all the filesystems tested. Moreover the graph shows clearly there is no significant decrease in performance during time, though regular, the writing behavior presents a pattern that after 10/12 writing sessions there is a cyclical decrease of writing speeds, occasionally down to 850 MB/s, which is acceptable.

On the other hand the iozone test (Page 15 Figure 18) showed some unexpected behavior in the reading speed. Even though the writing results were in line with the previous dd test, the reading behavior was not easy to interpret. Perhaps due to the optimization for large file operations, the readings performed under a small block size were extremely poor. The reading behavior improves only when we reach the 256 kb block

size, which is actually the RAID5 array block size, and sometimes improves further at 512kb which is the standard block size for the XFS filesystem.

3 dd Tests

All tests were run under configuration n.2
Command: **dd of=pippo.dat if=/dev/zero bs=1M count=20100**
A bash script was generated to fill up with file writings the whole array capacity with a 2 seconds delay between the writings in order to simulate what happens during an observation session: usually the antenna moves the dish between one observation and the other, so there is a time gap between one stream of data and the other. The average time gap should be 10/15 seconds, we tried to keep this gap in the simulation the lowest possible. The file size of 20 GB was chosen after taking an average of what a common observation file dimension is now.

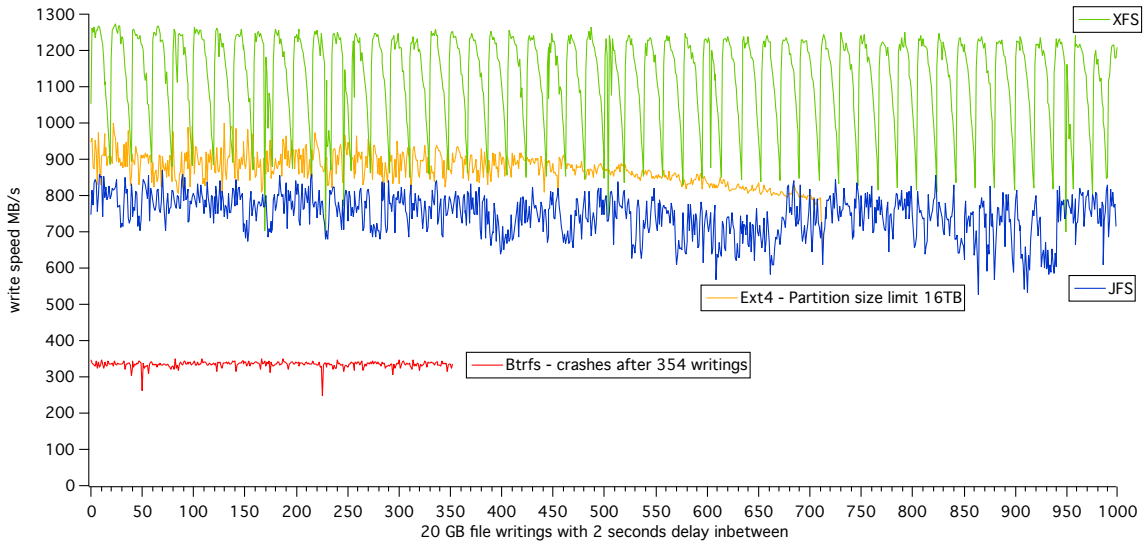

Figure 1: writing performance of the 4 filesystems on a 12 disks RAID5 array

The graph in Figure 1 summarizes the behavior of the four filesystems considered. XFS outweighs the other three filesystems significantly in terms of writing speed.

Filesystem	μ	σ
Btrfs	334 MB/s	8 MB/s
Ext4	874 MB/s	44 MB/s
JFS	747 MB/s	58 MB/s
XFS	1131 MB/s	121 MB/s

Table 7: Average speed and standard deviation

XFS filesystem has the highest results in average writing speed during the dd writings test, though it has the highest standard deviation as well. The Ext4 filesystem has the lowest standard deviation, due to the native implementation on the linux os.

4 Iozone Tests

The following tests were executed with iozone benchmark software (http://www.iozone.org/) on a RAID5 array with 12 2TB disks under system configuration n. 2.

Command given was:

iozone -i 0 -i 1 -r 4k -r 8k -r 16k -r 32k -r 64k -r 128k -r 256k -r 512k -r 1m -r 2m -r 4m -r 8m -r 16m -s #G -f /path... -b filename.xls

in order to use only write (**-i 0**) and read (**-i 1**) features of iozone for the chosen block sizes of the filesystem. Iozone produces a file of the chosen size (**-s #G**) and writes the output (**-b**) to an excel file.

The files sizes chosen for the tests were of 10 GB, 20GB and 50 GB.These file sizes were chosen in order to avoid the effect of caching and to mimic the typical maximum file size of an observation file which is around 50 GB.

In section 4.1 you can find a general performance comparison between the different filesystems doing reading and writing under a RAID5 12 2TB disks array.

4.1 Iozone comparison between read and write performance of all filesystems

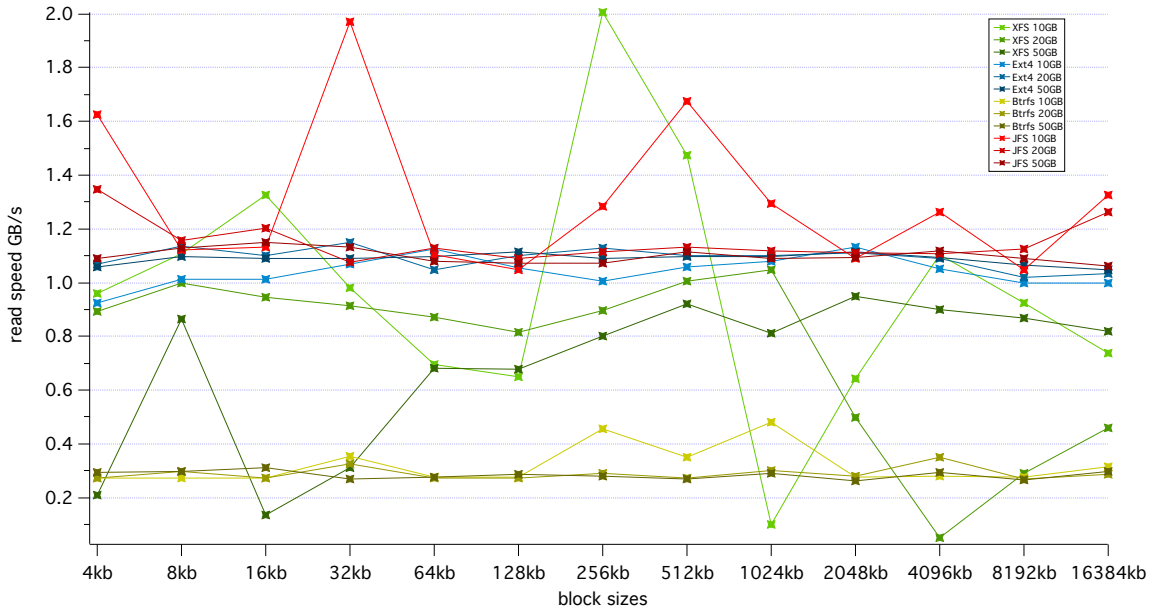

Figure 2: iozone read performance

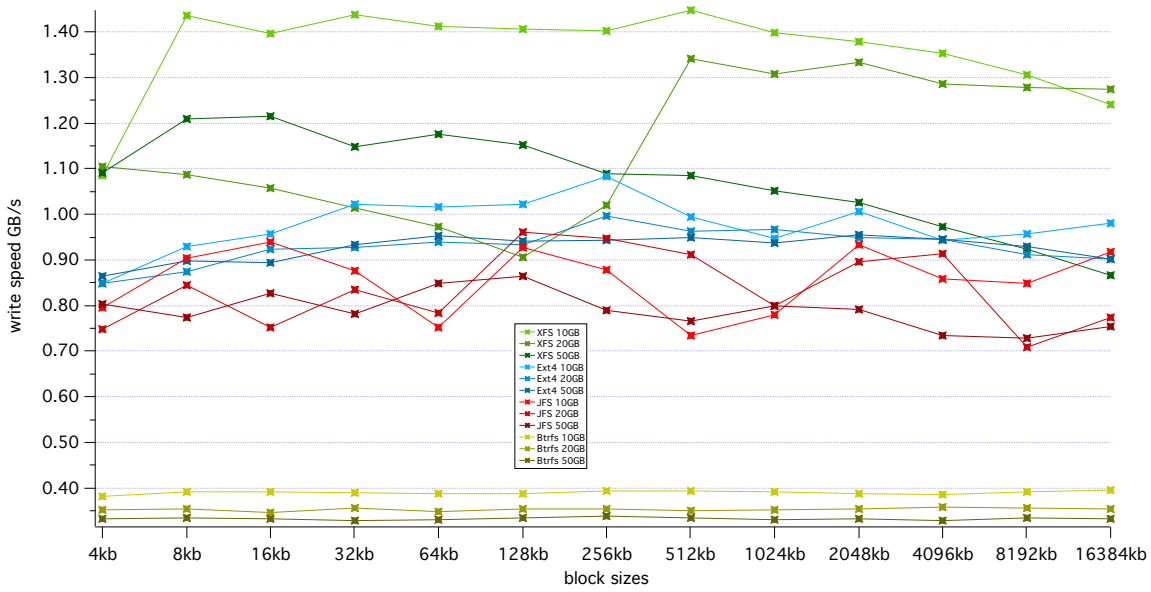

Figure 3: iozone write performance

5 XFS

XFS provides journaling for file system metadata, where file system updates are first written to a serial journal before the actual disk blocks are updated. The journal is a circular buffer of disk blocks that is never read in normal filesystem operation. The XFS journal is limited to a maximum size of both 64k blocks and 128MB with the minimum size dependent upon a calculation of the filesystem block size and directory block size.

On XFS the journal contains logical entries that describe at a high level what operations are being performed (as opposed to a physical journal that stores a copy of the blocks modified during each transaction). Journal updates are performed asynchronously to avoid incurring a performance penalty.

XFS makes use of lazy evaluation techniques for file allocation. When a file is written to the buffer cache, rather than allocating extents for the data, XFS simply reserves the appropriate number of file system blocks for the data held in memory. The actual block allocation occurs only when the data is finally flushed to disk. This improves the chance that the file will be written in a contiguous group of blocks, reducing fragmentation problems and increasing performance.

For applications requiring high throughput to disk, XFS provides a direct I/O implementation that allows non-cached I/O directly to userspace. Data are transferred between the application's buffer and the disk using DMA, which allows access to the full I/O bandwidth of the underlying disk devices.

The XFS guaranteed-rate I/O system provides an API that allows applications to reserve bandwidth to the filesystem. XFS will dynamically calculate the performance available from the underlying storage devices, and will reserve bandwidth sufficient to meet the requested performance for a specified time. This feature is unique to the XFS file system. Guarantees can be hard or soft, representing a trade off between reliability and performance, though XFS will only allow hard guarantees if the underlying storage subsystem supports it. This facility is most used by real-time applications, such as video-streaming.

Because of the evidence produced by our tests we decided to spend some more time on a deeper investigation on the XFS filesytem. The next sections illustrate the test on a 24 2TB disks both RAID0 and RAID5 array. The last section is about the results produced on the same dd test performed on a 12 2TB disks RAID5 array, but under the condition of a degraded array.

5.1 dd XFS tested on a 24 disks 2TB array

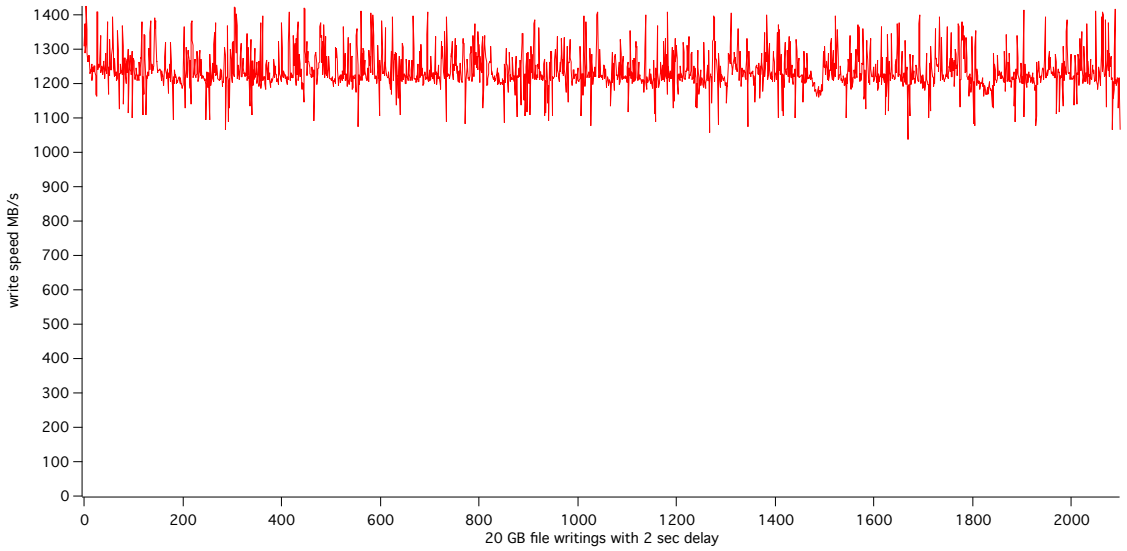

Figure 4: XFS on RAID0

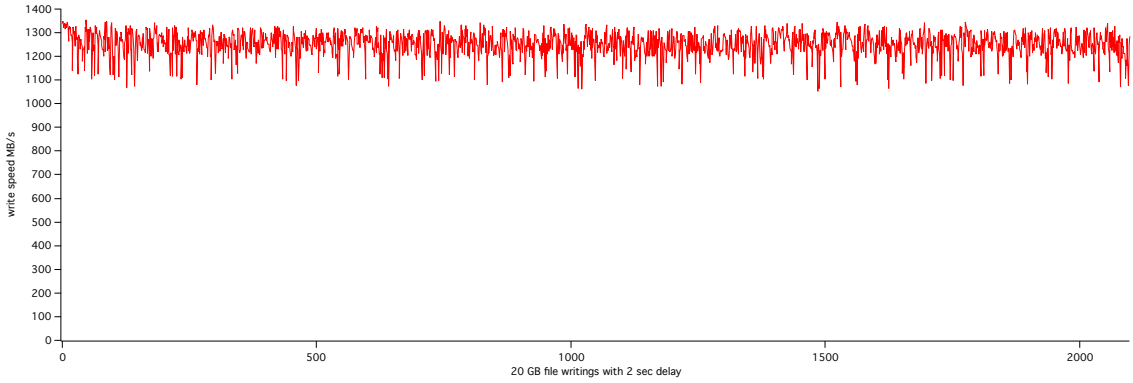

Figure 5: XFS on RAID5

As seen in Figure 4 and Figure 5 graphs, we tried to investigate if there was an appreciable difference of performance between a RAID0 and a RAID5 under a 24 disks array, but actually the results tell no differences.

5.2 iozone XFS tested on a 24 disks 2TB array

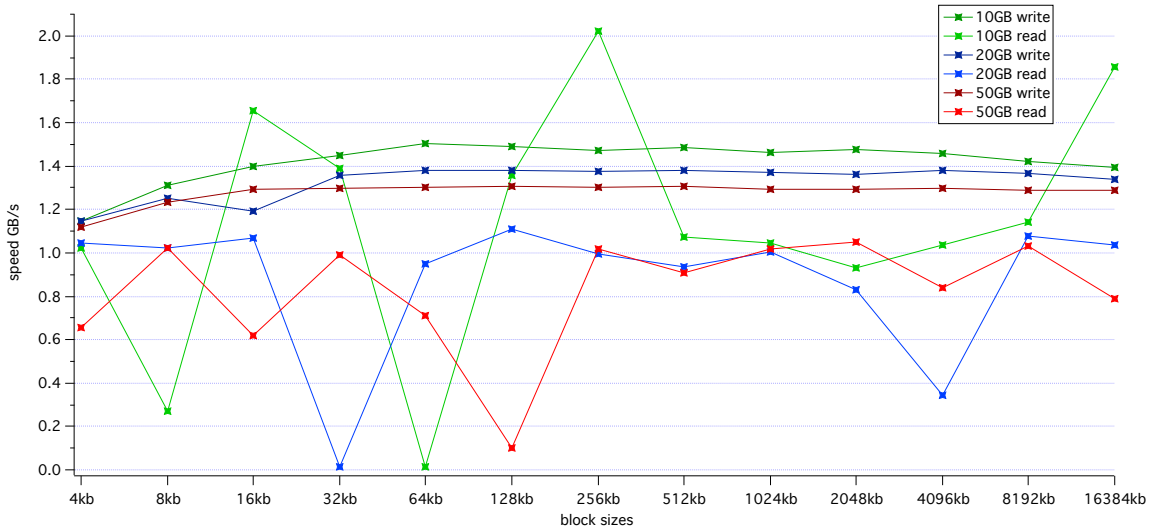

Figure 6: XFS filesystem on RAID0 & 24 disks

The iozone these on the 24 disks array confirms the writing speed for the 20 GB file and shows clearly there is a high stable speed for both the 10 and 50 GB files tested. However the reading speeds still presents an unpredictable behavior.

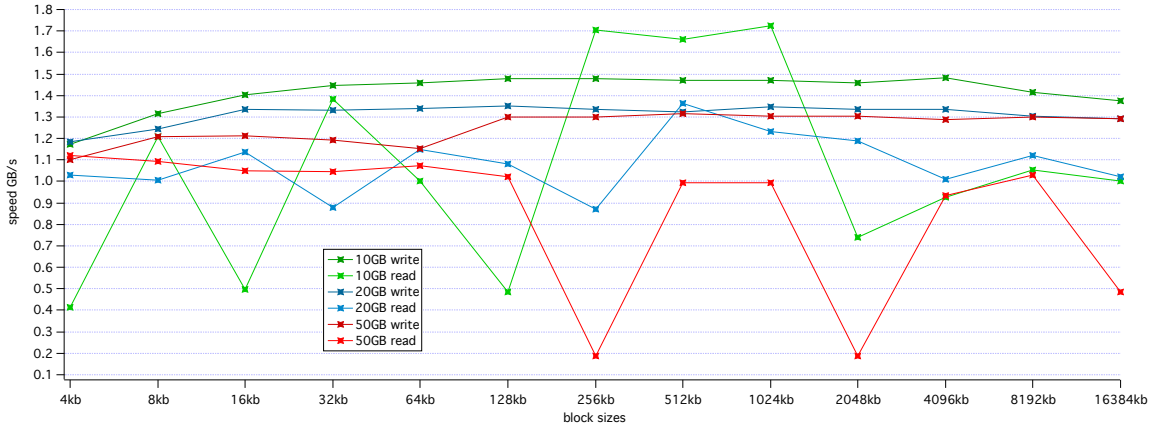

Figure 7: XFS filesystem on RAID5 & 24 disks

5.3 dd XFS tested on a DEGRADED 12 disks 2TB array

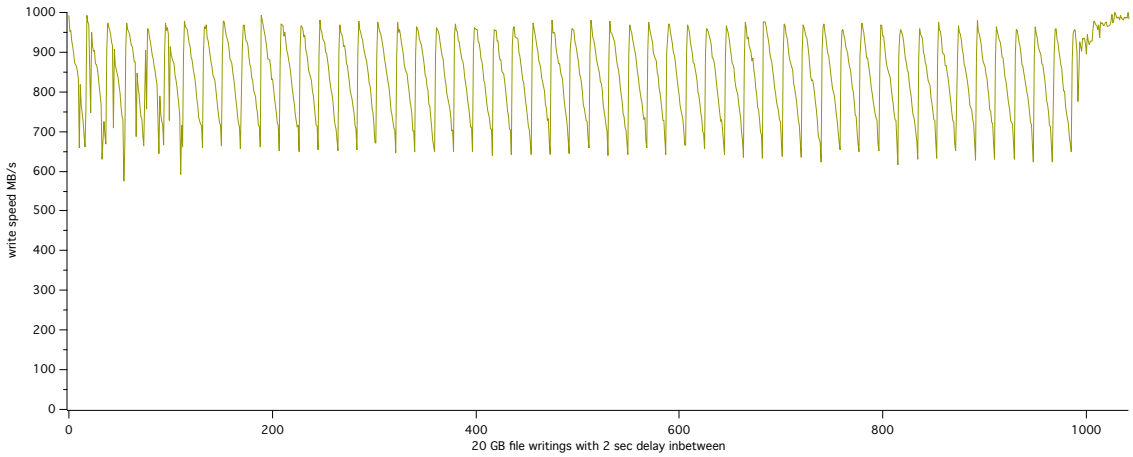

Figure 8: XFS filesystem on 12 disks RAID5 degraded array

Our final test on the XFS filesystem was performed in the condition of a degraded array removing a single drive from the RAID set. As forecasted, there was a writing speed decrease, but it was an acceptable one, in a degraded condition XFS behaves more or less like Ext4 in the average writing speed. This could be significative in the event of a faulty drive during and observation run. The observation data stream could still be written at slower pace and not interrupted. According to the results provided to our last tests the system configured with 24 disks can easily stand a 4 Gbps writing speed and seems to grant an 8Gbps speed as well.

5.4 Reading while writing

In order to investigate further what could happen in a production environment we decided to perform a test about what could happen while we were writing files. We imagined that during a writing session we could read the previous data written for the purpose of correlating it in quasi real time. So while one thread was writing we introduced another thread reading the perviously written files with the command:
dd if=/data/pippo of=/dev/null bs=1M count=20100

The results show that XFS behaves vey poorly when a massive reading thread is thrown, then regains the same behavior when the reading ends.

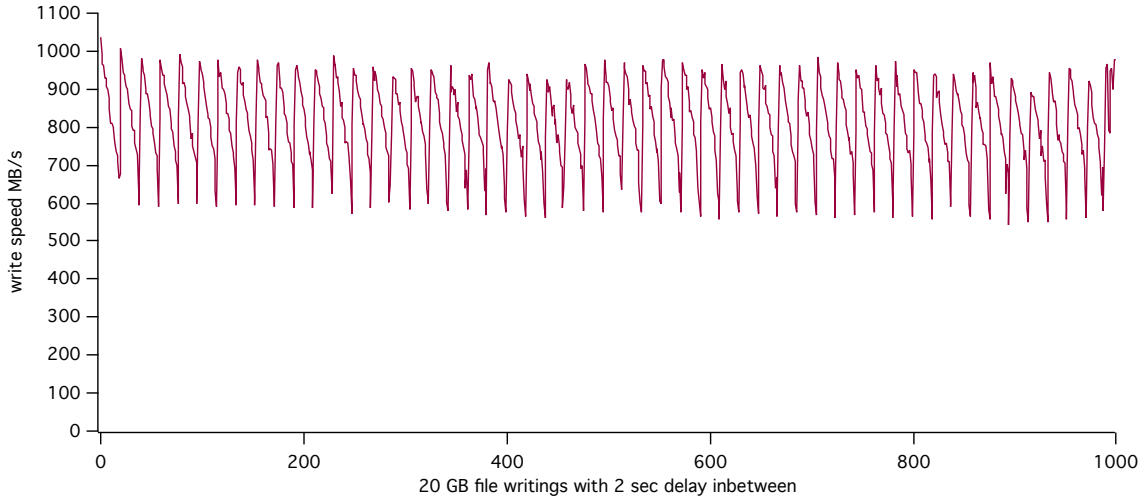

Figure 9: XFS filesystem while writing on 12 disks RAID5 array

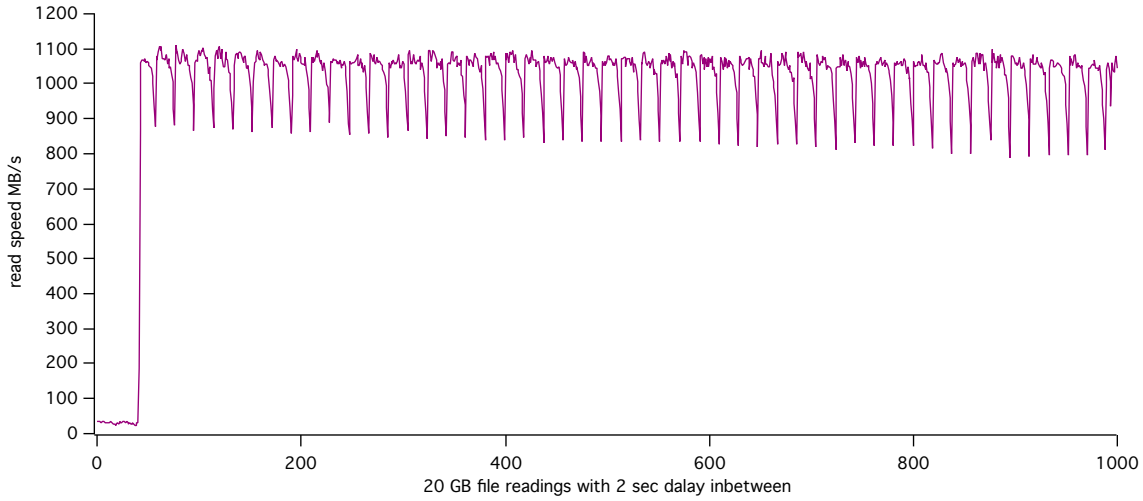

Figure 10: XFS filesystem while reading on 12 disks RAID5 array

A dd Graphs

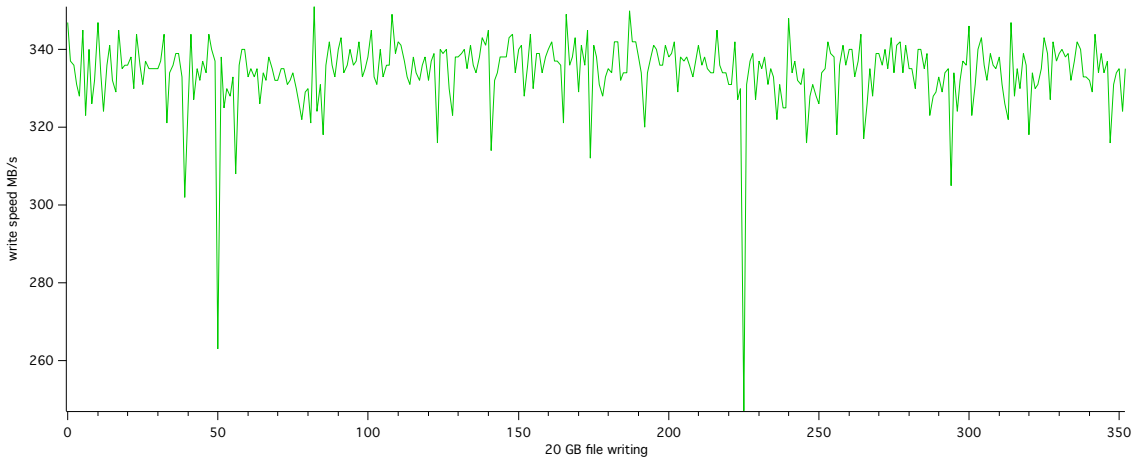

Figure 11: Btrfs performance: stops writing after 354 times telling the array capacity is full

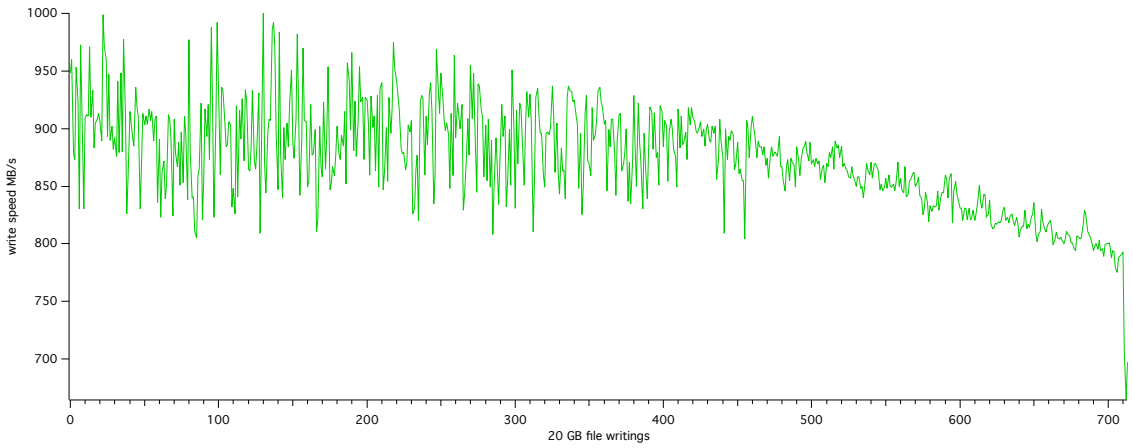

Figure 12: Ext4 performance: the partition size limit is 16TB

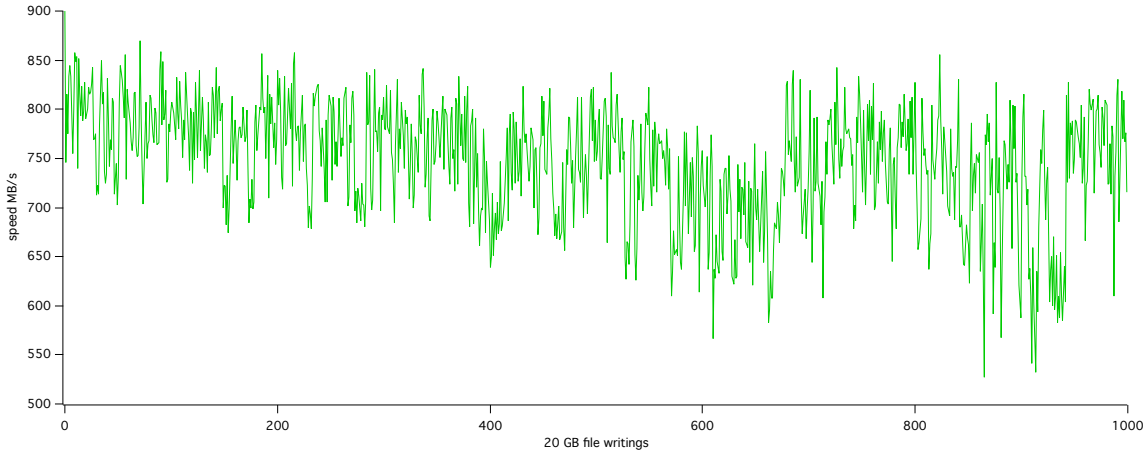

Figure 13: JFS performance

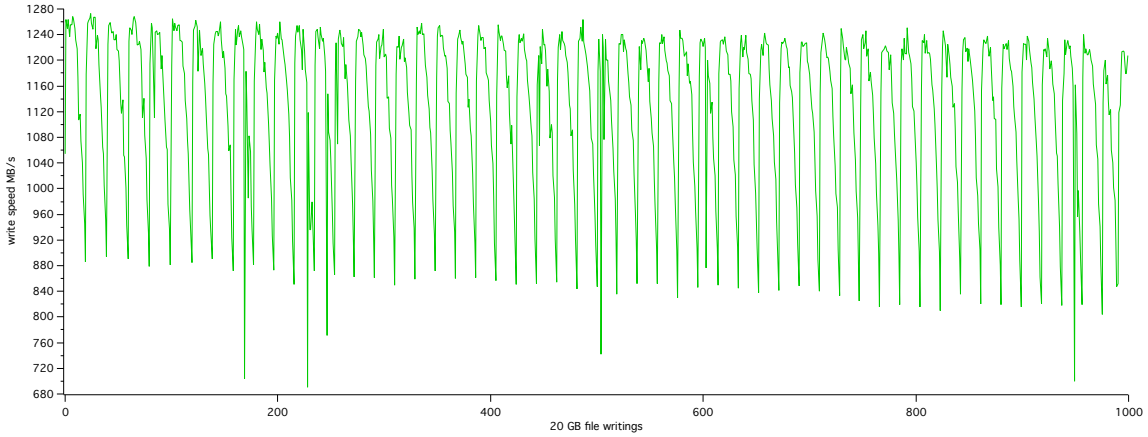

Figure 14: XFS performance

B iozone Graphs

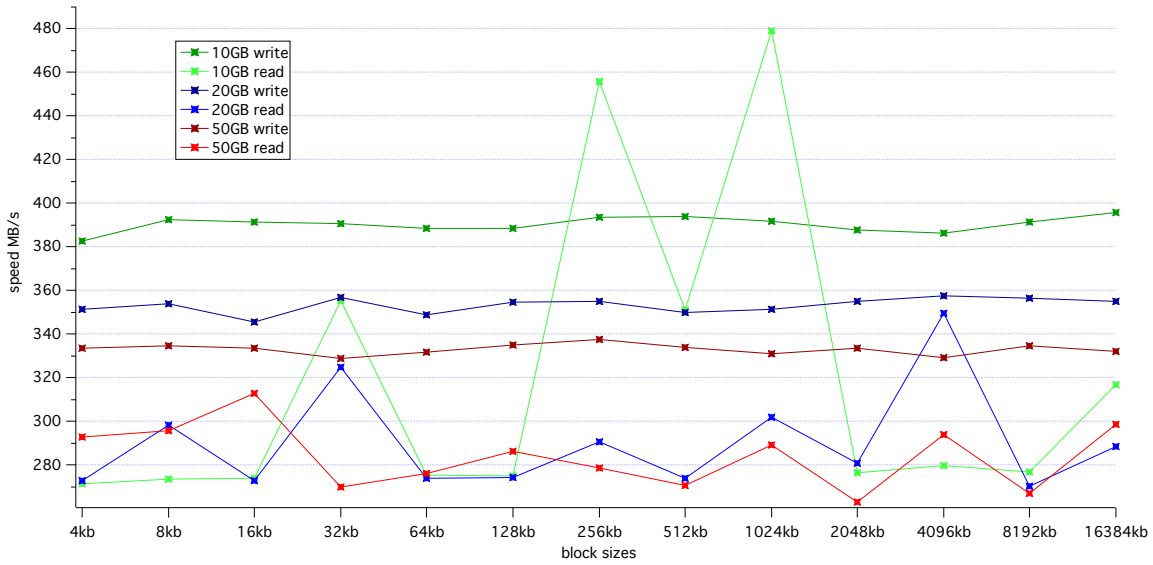

Figure 15: Btrfs filesystem

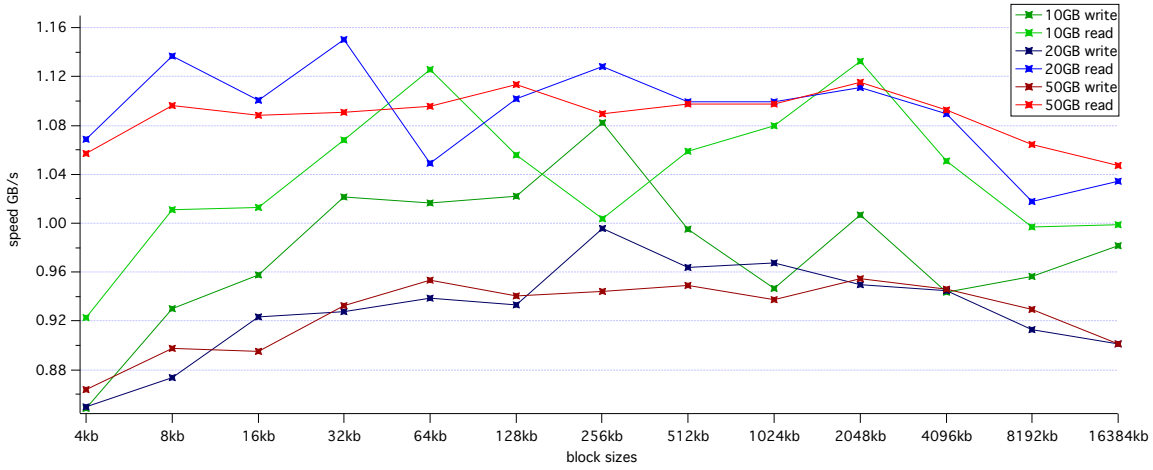

Figure 16: Ext4 filesystem

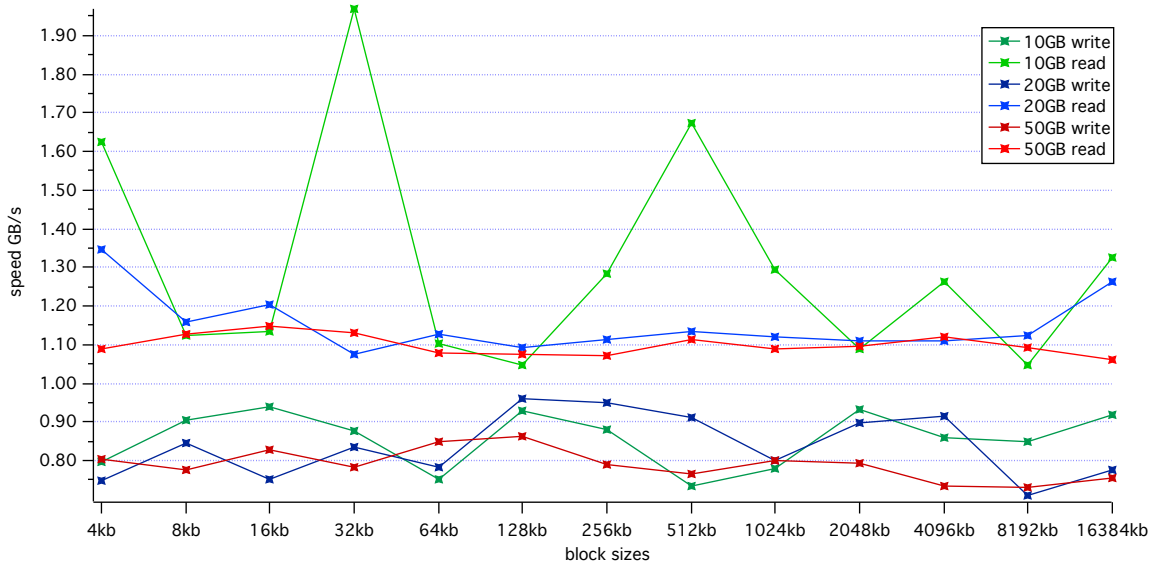

Figure 17: JFS filesystem

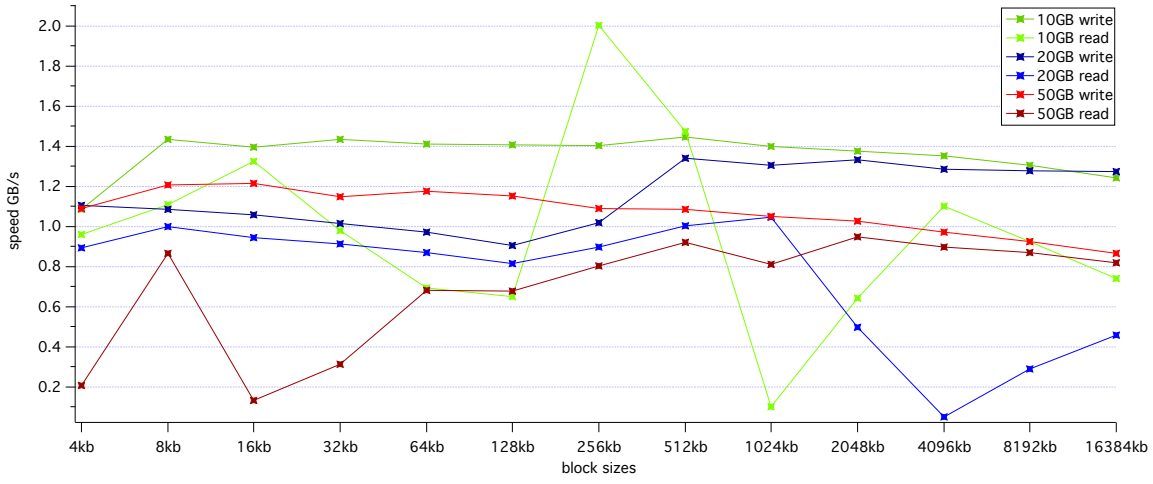

Figure 18: XFS filesystem