ZFS benchmarks on Linux

Thomas LEIBOVICI – CEA/DAM
thomas.leibovici@cea.fr

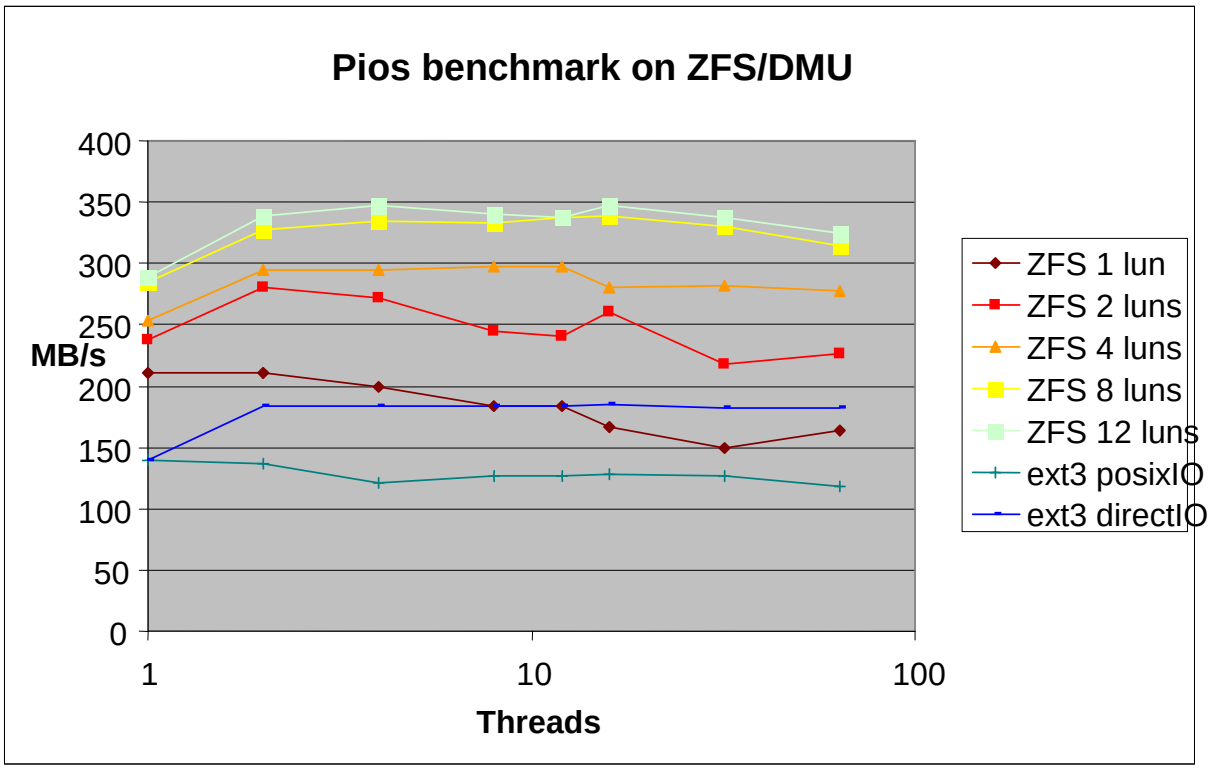
This document describes benchmarks and I/O profiling done with the 'pios' tool over ZFS/DMU on Linux.

The hardware used for theses tests is:

- 4 Intel Xeon 3.16GHz processors
- 4 GB memory
- Red Hat Enterprise Linux 4 update 2
- DDN S²A9550 with SATA disks (8+2P luns x 12) - LPFC connections.

ZFS striping performance

The first benchmark consists in striping a various amount of luns in a ZFS pool, with 1 to 64 pios' threads writing in *dmuio* mode.
As a reference for theses values, we also measured performances that we got with a lun formatted in ext3, writing with *directIO* and *posixIO* pios' modes.

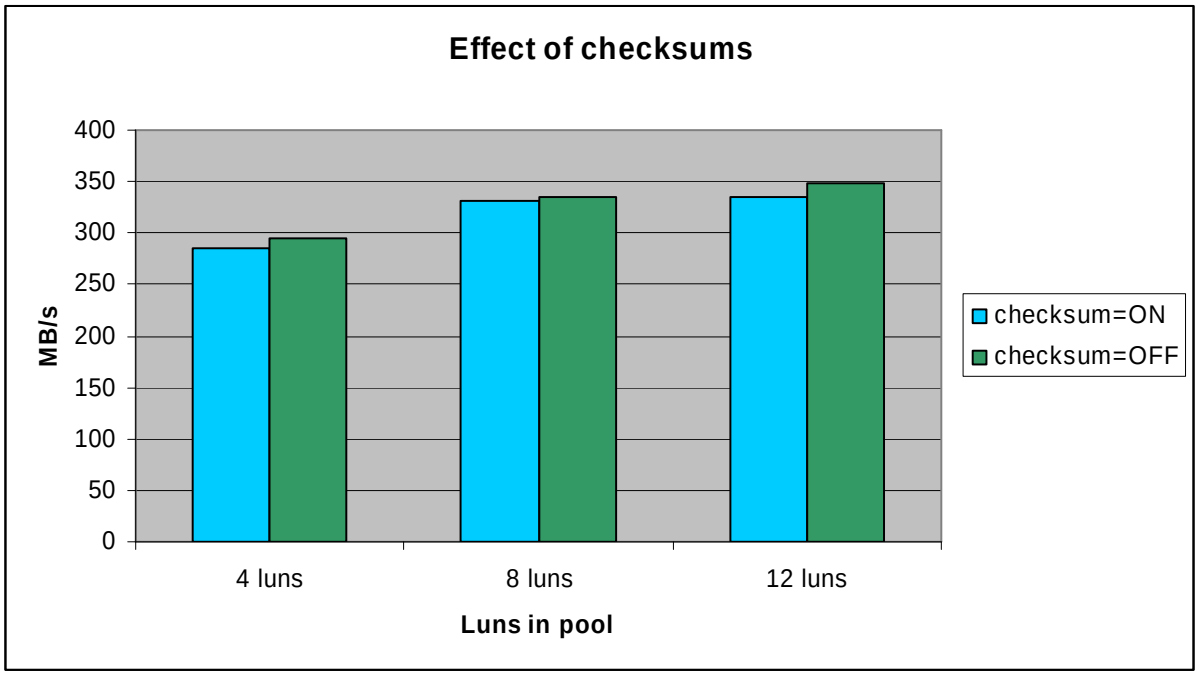


When doing a ‘dd’ command directly on such a device, we get about 350MB/s.
Thus, considering ZFS overhead, getting 210 MB/s with 1 lun in ZFS pool is quite a good performance.
What’s more, performances with 1, 2 and 4 threads exceed those of ext3.
However, there is no speed-up at all using multiple-threads, and the throughput decreases as number of threads increases.
Regarding stripe, we get a poor scalability: only 30MB/s more with a second lun in pool, 10MB/s speed-up with 2 more luns… And the throughput reaches it maximum at 350MB/s with 12 luns.

Not very enjoying… Thus, in the next benchmarks, we will try to increase theses performances and analyze where the bottlenecks are.

Impact of checksums

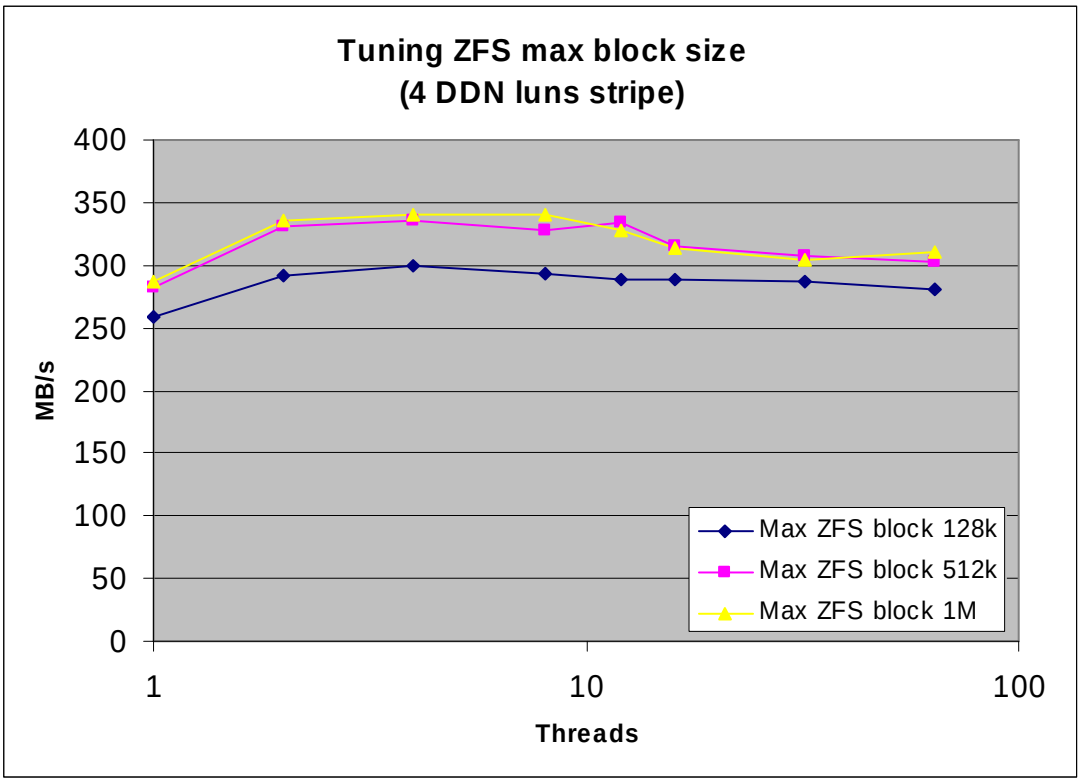
In this benchmark, we compare performances with/without SHA256 checksums on data. (This graph displays the result with 4 pios threads. We get about the same results with another thread count).



We get a small improvement (~10 MB/s) when disabling checksums (green bars).
However, this feature appears not having a huge impact on performances.

Changing ZFS max block size

Natively, ZFS doesn't write data blocks larger than 128kB, in order to optimize "CopyOnWrite" operations. However, we know that writing huge blocks is much more efficient with DDN S²A9550. Because of this, we tried to see if we got more bandwidth when increasing this max block size.



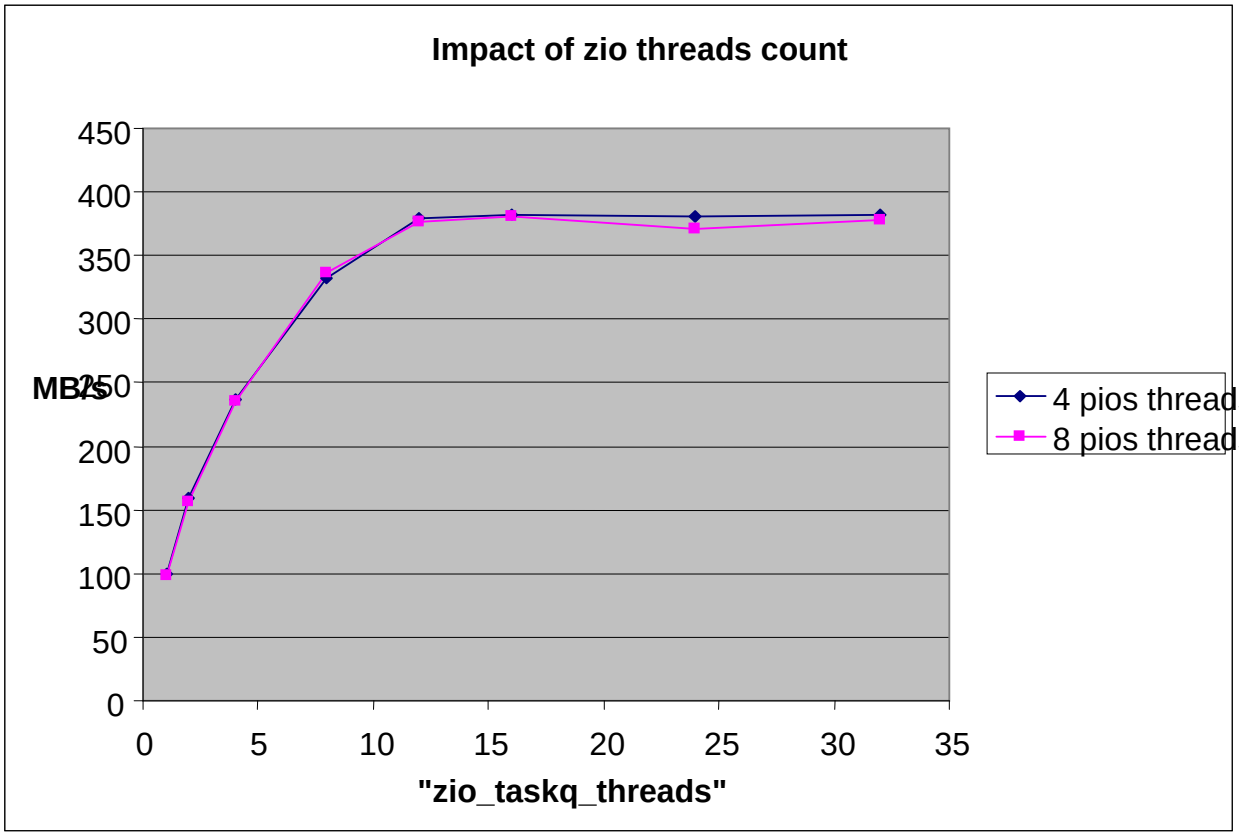
When changing this size to 512kB or 1MB, we get a significant improvement (20 to 50MB/s). Of course, this bench was done writing data from scratch, so CoWs were almost limited to metadata blocks. Thus, this modification should only be considered if Lustre is mostly writing objects from scratch, without modifying small pieces of data.

However, it appears this tuning does not enhance threads scalability or stripe performances.

ZIO threads

Most ZFS' IOs are done by a pool of threads called "zio_taskq_threads" (also called "spa_zio_issue" threads). By default, this pool consists of 8 threads. Let's modify this value to see its impact on performances (it can be changed in "libzpool/spa.c").

In this benchmark, we are writing to a 6 luns zpool. ZFS max block size is set to 1MB. We then change zio threads count from 1 to 32:



It first appears zio_taskq_threads count has a strong impact of ZFS' throughput. Also, the default value of 8 threads is not optimal. We get the best performance using at least 12 zio threads.

IO and load profiling

For profiling ZFS workload, we used several system tracers and profiling tools like strace, gstack and gprof. We also instrumented ZFS in order to determine how much time each thread was doing effective work (checksumming, doing IOs, …).

In the following benchmarks, we launch the following 'pios' command:

```
pios -c 2M -o 128M -n 64 -t 4 -s 128M -L dmuio -p ZFS
```

We use a 1MB ZFS max block size, and a 6 luns zpool.

Using 8 zio_taskq_threads

Runtime: 24.57s
Bandwidth: 333MB/s

Thread type	count	Time spend for checksums (s/thread)	Time spend for IOs (s/thread)	Nbr of simultaneous IOs
spa_sync	1	**11.81**	0.61	1
spa_zio_issue readers	8	~0	~0	-
spa_zio_issue writers	8	~0	**19.10**	7.54
spa_zio_intr	8	~0	0	-

It appears that most of the workload consists of spa_zio_issue threads' IOs. Those IOs are well parallelized (7.54 simultaneous IOs with 8 writters).
A bad point is that checksumming data is mostly done by a single thread: the spa_sync thread (48% of runtime).

Using 16 zio_taskq_threads

Then, we do the same test with more "spa_zio_issue" threads, in order to see the evolution of the load profile (this can be done in "spa.c", modifying the value of "zio_taskq_threads").
We are now using 16 "spa_zio_issue" threads instead of 8:

Runtime: 21.79s
Bandwidth: 375 MB/s

Thread type	count	Time spend for checksums (s/thread)	Time spend for IOs (s/thread)	Nbr of simultaneous IOs
spa_sync	1	**11.75**	0.64	1
spa_zio_issue readers	16	~0	~0	-
spa_zio_issue writers	16	~0	**11.51**	11.35
spa_zio_intr	16	~0	0	-

With more "spa_zio_issue" threads, IOs are faster (12s/thread for IOs instead of 19s), and well parallelized (about 11 simultaneous IOs with 16 threads). Thus, the overall performance is better than using 8 threads: 40MB/s speed-up.

However, the time for checksumming data is about the same (~11.8s), because this job is only done by the "spa_sync" thread. This seems to be an important bottleneck.

Parallelizing checksums

To parallelize data checksumming, we have to go deeper in ZFS sources… To do that, we tried to parallelize operations done by the "dbuf_sync_list" function. To keep the correct order for synchronizing objects, we only parallelized "dbuf_sync_leaf" operations for each parent node: when we have to synchronize a set of leaves for a given parent node, we queue them so a dedicated pool of threads (called "dbuf_sync_leaf" threads) processes them in parallel. We then wait for all this node's children to be synchronized before managing over parent nodes, sequentially and in the same order as the original ZFS behavior.

In the following benchmarks, we launch the following 'pios' command:

```
pios -c 2M -o 128M -n 64 -t 4 -s 128M -L dmuio -p ZFS
```

We use a 1MB ZFS max block size, a 6 luns zpool, and 16 zio_taskq_threads.

Profile

Here is the profile we get using 4 "dbuf_sync_leaf" threads.

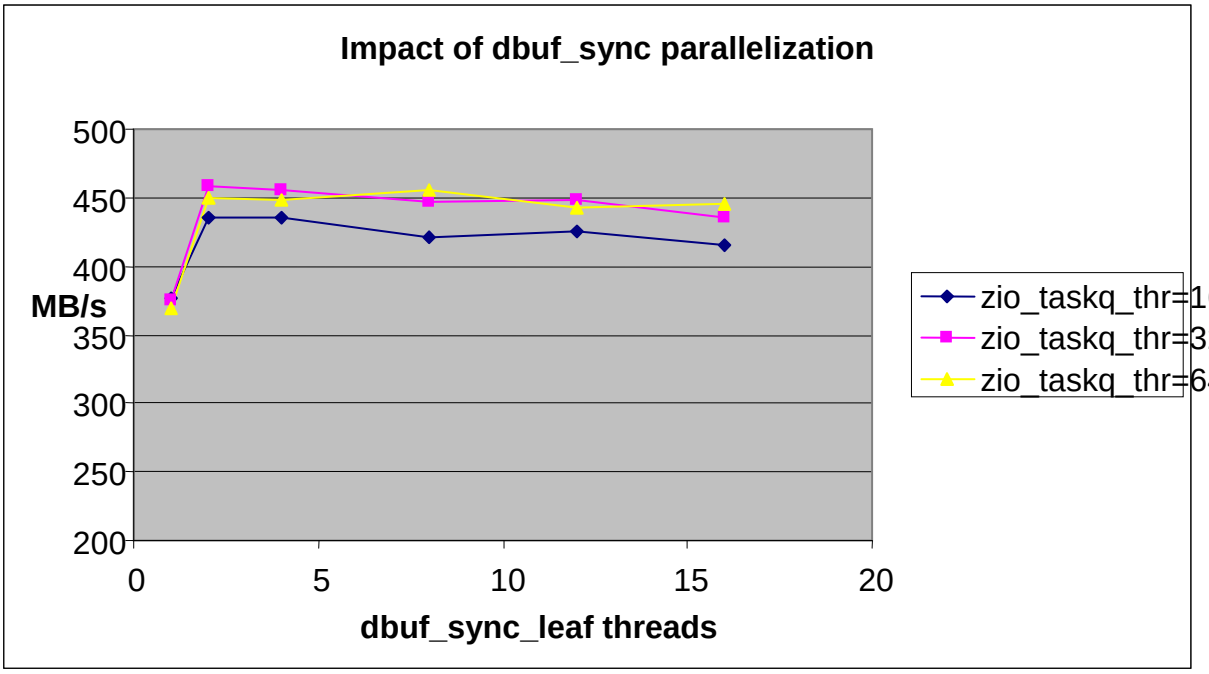
Runtime: 19.64s
Bandwidth: 417 MB/s

Thread type	count	Time spend for checksums (s/thread)	Nbr of simultaneous checksums	Time spend for IOs (s/thread)	Nbr of simultaneous IOs
spa_sync	1	**0.08**	1	0.44	1
dbuf_sync_leaf	4	**4.48**	2.40	**-**	**-**

With this mod, we have a better parallelization of data checksumming (2.4 simultaneous checksums with 4 "dbuf_sync_leaf" threads), and the performance is enhanced (417 MB/s instead of 375 MB/s).

Performance evolution

Let's see performances we get when using different "dbuf_sync_leaf" threads count.



It appears that using several "dbuf_sync_leaf" threads is always better than in the non-parallelized case. However, when using more and more threads, performances are decreasing. We get the best performance using 2 "dbuf_sync_leaf" threads.

Conclusion

This study of ZFS/DMU showed that:

- In the current version, pios benchmark over DMU does not scale using multiple threads;
- Increasing ZFS max block size results in better performances with pios' access pattern over DDN S²A9550 hardware;
- zio_taskq_threads count has also a strong impact on performances;
- An important bottleneck resides in the "dbuf_sync_list" function which is called by a single thread ("spa_sync" thread): it would be interesting to parallelize it. Our "quick" mod for parallelizing calls to "dbuf_sync_leaf" resulted in a significant improvement. I'm sure ZFS experts will do wonderful things by reconsidering "dbuf_sync_list" mechanism.