Evaluation of Nilfs2 for Shingled Magnetic Recording (SMR) Disks

Technical Report FSL-14-03

Benixon Arul Dhas and Erez Zadok
Stony Brook University

James Borden and Jim Malina
Western Digital Corporation

Abstract

The nature of Shingled Magnetic Recording (SMR) disks makes log structured file systems suitable for such disks. Nilfs2 is a promising log structured file system and is available in the Linux kernel source as an experimental file system. Nilfs2 is actively maintained and stable, providing a good platform for functional and performance evaluation of SMR disks. The Linux Test Project (LTP) test-suite is used for functional testing and Filebench is used to evaluate various macro- and micro-benchmarking workloads. The results show that for most of the workloads for Nilfs2 there is no significant difference between a normal disk and the SMR disk prototype. The results look promising in that there is some performance loss for some workloads because of the firmware modification to support SMR drives. The random-write workload showed a decrease in throughput of about 15%. All other workloads showed throughput decrease of less than 5%.

1 Introduction

Evaluation of Shingled Magnetic Recording (SMR) disks provides using an unmodified version of Nilfs2 offers a baseline for performance of these disks. SMR disks allow only sequential writing and hence demand a log structured write access. Nilfs2 [8], being a strictly log-structured file system, allows for usage of SMR disks with very minimal modification to firmware.

We tested the functionality of Nilfs2 using the SMR disk by using the Linux Test Project (LTP) [4] test suite. This consists of hundreds of tests that are designed to exercise the interfaces provided by file systems for any errors. A large number of these subsets were run on SMR and regular disks. (Throughout the rest of this paper, we refer to "regular" as the non-SMR disk.) The results showed that Nilfs2 was equally stable on both the SMR disk and regular disks. The SMR drive firmware has a feature to enforce that the host wrote sequentially within zones; Nilfs2 did not violate this requirement.

We used Filebench [2] for micro- and macro-benchmarking of Nilfs2 on SMR disks. We evaluated macro benchmarking workloads consisting of webserver, mailserver, and fileserver on SMR and regular disks. These are representative of practical workloads. The results obtained shows that Nilfs2 can be used on SMR disks without significant loss in performance for workloads that are read-intensive or have large sequential writes. We ran the following micro-benchmarking workloads: random read, random read direct, random write, random write direct, sequential read, and sequential write. The results from these micro benchmarking indicate that there is no significant change in the performance due to reads. However, there is a drop in the performance during write in SMR disks, believed to be due to the overhead of the modified firmware in the SMR disk. We saw a maximum decrease in throughput of 15% for random write with direct I/O workload (bypassing the host OS cache). For other workloads, which primarily consisted of non-random writes, we saw that the throughput degradation was less than 5%. For read-intensive workloads there was no significant difference in observed throughput. We see that the Webserver workload has the least performance impact among the macro-benchmarking workloads, because it consists mainly of reads and sequential writes to the Web server log (i.e., very few random writes).

2 Background

Shingled Magnetic Recording (SMR) technology promises eventual increased storage density of the order 3–4× [5] that of current HDDs; each generation of new SMR drives is expected to yield 5–25% space increases, with first generation increases reported at 20–25% [3,6]. However, using the entire disk as shingled could cause some limitations for practical application of these disks. For example, one might have to wait until the entire one-zone disk is full or nearly full to garbage collect freed blocks. Therefore, it may be acceptable to incur some additional firmware overhead and somewhat reduced capacity, and partition the disk into multiple zones to improve performance. Each zone is shingled such that writes within zones have to be physically sequential to avoid data loss; writes across zones, however, can be random as long as sequentiality is maintained within each SMR zone. There are also two zones (the first and the last zone) which are not shingled, thereby allowing random access in these zones; this is a current necessary requirement for running Nilfs2. Note, however, that many permutations of numbers and types of zones are possible (i.e., any mix of shingled and regular zones). Indirection systems [1] are one form of software/firmware to make shingled disks usable by enforcing sequential physical writes for random logical writes. Indirection systems typically have to maintain a logical-to-physical mapping of blocks, such as an SSD's Flash Translation Layer (FTL). This can result in a growing amount of meta-data that has to be maintained for larger disks (even though data sizes are much larger than meta-data sizes). The meta data for such indirection systems mainly includes the mappings from logical LBA to physical LBA. The percentage of space occupied by this meta data is small compared to the disk capacity. However, this is still a large amount of information and is difficult to keep it all in main memory. So in this work we used a log-structured file system at the host, instead of indirection at the block layer or firmware.

3 Benchmarks

3.1 Experiment Setup

We executed LTP and Filebench on the regular HDD and the SMR disk. To avoid any out-of-sequence writes in the block layer, we set the Linux I/O scheduler to *noop*. This was done for both the regular and SMR disks, for the LTP and Filebench tests. CFQ is the I/O scheduler used by default in Linux based systems. So we also repeated the experiments on regular disks with CFQ as I/O scheduler. The SMR disk had two random-access zones (first and last zone). The entire disk had 20,000 zones with each zone spanning 128MB.

3.1.1 LTP

We executed the LTP test suite for regular and SMR disks, both formatted with Nilfs2. We mounted the SMR disk on a system with Intel Pentium IV processor running at 3.40GHz and 2GB of main memory. we mounted the regular HDD on a system with an Intel Pentium IV processor running at 2.40GHz and 640MB of Main memory. The operating system on both machines consisted of a vanilla Linux kernel version 3.8.2 compiled over a CentOS Linux distribution Version 6.2. The operating system was hosted on an ext4 file system mounted over a 40GB Western Digital SATA hard disk. Since the objective was to evaluate any discrepancies in the functionality, the minor differences in the configurations of the machines may not be of much significance. Still, we are planning to repeat these tests on a modern server in the near future.

3.1.2 Filebench

We executed Filebench workloads on a system with Intel Pentium IV processor running at 3.40GHz and 2GB of Main memory. The operating system consisted of a vanilla Linux kernel version 3.8.2 compiled over a CentOS Linux distribution Version 6.2. The operating system was hosted on an ext4 file system mounted over a 40GB Western Digital WDC WD400BD-75JMC0 SATA disk running at 7,200 RPM with a maximum sustained transfer rate of 56MB/sec. The regular and SMR disks were connected to SATA ports via the same PCI card. The regular disk was a 3TB Western Digital WDC WD30EZRX-00MMMB0 SATA disk and the SMR disk was a 2.68TB Western Digital WDC WD30EZRX-00MMMB0 SATA disk. The actual SMR disk before firmware modification had a capacity of 3TB. However, after modifying the firmware to support 20,000 zones of 128MB each, the capacity was limited to 2.68TB.

3.2 Experiment Description

3.2.1 LTP

We used the LTP [4] test suite to evaluate dio, fs, fsx and mm sub tests and other auxiliary sub tests like ipc, math, pty, nptl, sched, syscalls and containers. A brief description of each sub test is provided below,

- The dio test automation suite helps run direct input/output functionality tests and report results.
- The fs test automation suite helps run file system functionality tests and report results.
- The fsx test automation suite helps extended file system functionality tests and report results.
- The mm test automation suite helps run memory management functionality tests and report results.
- The ipc test automation suite helps run inter process communication functionality tests and report results.
- The math test automation suite helps run mathematical functionality tests and report results.
- The pty test automation suite helps run pseudo terminal functionality tests and report results.
- The nptl test automation suite helps run Native POSIX Threading Library functionality tests and report results.
- The sched test automation suite helps run scheduler functionality tests and report results.
- The syscalls test automation suite helps run system calls functionality tests and report results.
- The container test automation suite helps run the container functionality (e.g., utsname) tests and report results.

The results for both SMR and regular disks were the same. To ensure that the results were consistent, we repeated the tests five times. There were some failure cases for both SMR and regular disk. These were containers, dio4, dio10, quota remount test01, ksm03, ksm03_1, ftruncate04, ftruncate04_64, swapoff01, swapoff02, swapon and sysctl03. Since these failures were seen in both scenarios, we conclude that these are not due to the SMR disk but rather than Nilfs2 itself is lacking in some features. Note that this is not uncommon: in our experience many file systems pass the vast majority of LTP tests, but not necessarily all.

3.2.2 Filebench

Each of the Filebench workloads were run for a period of one hour and repeated five times. The average of the five runs was calculated and plotted along with the standard deviation. Unless otherwise noted, all standard deviations were less than 5% of the mean, indicating a fairly stable experiment. We executed the Filebench workloads with the help of bench-scripts with the autopilot [7] tool.

Macro benchmark workloads. The macro benchmarks in Filebench are approximate simulations of real time workloads. The results from these workloads are representative of real workloads like a webserver and fileserver under normal scenarios. Although these may not be as accurate as trace replays, from a file-system evaluation perspective, this is one of the best ways to benchmark.

- The **Webserver** workload consisted primarily of reading small files randomly. We configured Filebench with the default webserver workload with 100 threads running in parallel. We scaled the entire workload to 8GB consisting of 512,000 files each of size 16KB.

- The **Mailserver** workload consisted of random deletion, write and append to small files. We configured mailserver workload to run 16 threads in parallel. We scaled the entire workload to 8GB consisting of 512,000 files each of size 16KB.

- The **Fileserver** workload consisted of random write, append, deletion and getting statistics of moderately sized files. We configured the fileserver workload to run 50 threads in parallel. We scaled the entire workload to 8GB consisting of 64,000 files each of size 128KB.

Micro benchmark workloads. Micro benchmarking allows us to focus on a particular aspect of the file system and capture its performance. We executed benchmarks for random and sequential read/write workloads. We use 4KB I/O sizes because they meet the most common native disk block size (and CPU page-cache size) on most systems today.

- We configured the **Random read** workload to run for a single file of size 5GB accessed by 10 process instances with 10 threads each with I/O size of 4KB.

- We configured the **Random read direct** workload to run for a single file of size 5GB accessed by 10 process instances with 10 threads each with I/O size of 4KB.

- We configured the **Random write** to run for a single file of size 5GB accessed by 10 process instances with 10 threads each with I/O size of 4KB.

- We configured the **Random write direct** workload to run for a single file of size 5GB accessed by 10 process instances with 10 threads each with I/O size of 4KB.

- We configured the **Sequential read** workload to run for a single file of size 5GB accessed by 10 process instances with 10 threads each, with I/O size of 4KB.

- We configured the **Sequential write** workload to run for a single file of size 5GB accessed by 10 process instances with 10 threads each, with I/O size of 4KB.

4 Evaluation

4.1 Macro Benchmarks

Figure 1 depicts the comparison of throughputs of the webserver, mailserver, and fileserver workloads for the SMR disk with noop I/O scheduler, regular disk with noop I/O scheduler, and regular disk with CFQ I/O scheduler. The operations per second for the same experiment is shown in Figure 2. From Figure 1 we observe the following. For the webserver workload, the throughput was almost the same for both SMR and regular disks with noop as I/O scheduler. Also, the throughput for the regular disk with the CFQ I/O scheduler was about 11% more that the previous two cases. We believe that the difference is primarily due to merging of multiple I/O requests at the block layer in the queues maintained at the I/O scheduler. The webserver workload creates lot of small I/O read requests that can benefit from merging at the block layer. This needs to be verified with further experiments.

The Mailserver workload for SMR disks from Figure 1 shows about a 7% decrease in throughput compared to a regular disk with noop as I/O scheduler. This decrease can be attributed to the overhead incurred at the device firmware for write requests. The CFQ I/O scheduler for the regular disk shows an increase in throughput

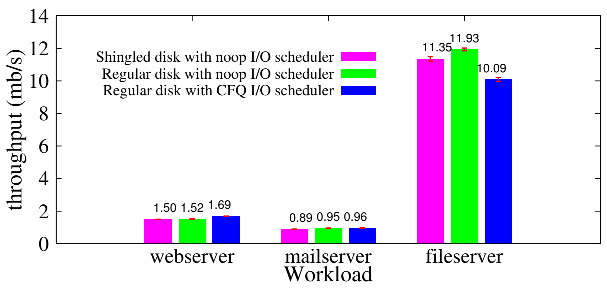

Figure 1: Throughput plot for macro benchmark workloads of webserver, mailserver, fileserver run using Filebench

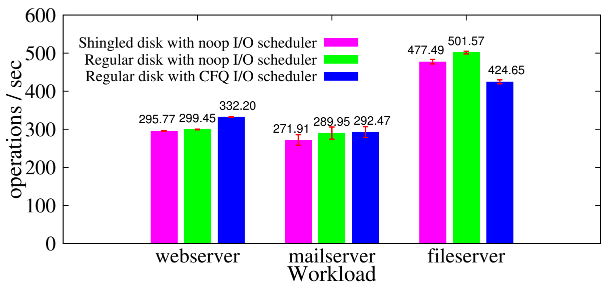

Figure 2: Operations per sec plot for macro benchmark workloads of webserver, mailserver, fileserver run using Filebench

of 1% when compared to the regular disk with noop as I/O scheduler.

The Fileserver workload in Figure 1 shows a similar pattern as that of the mailserver with a higher absolute throughput. The SMR disk throughput is 5% lower than the regular disk with the noop I/O scheduler, which in turn is 15% higher than the regular disk with CFQ as the I/O scheduler.

4.2 Micro Benchmarks

Figure 3 depicts the comparison of throughput of random read workloads for the SMR disk with the noop I/O scheduler, regular disk with the noop I/O scheduler and the regular disk with the CFQ I/O scheduler. The operations per second for the same experiment are shown in Figure 4. The Random read workloads from Figure 3 show that for both with and without direct-I/O, the result was almost the same throughput for all three cases. This indicates that the overheads for reads due to the firmware modification are negligible.

Figure 5 depicts the comparison of throughput of random-write workloads for the SMR disk with the noop I/O scheduler, the regular disk with the noop I/O scheduler, and the regular disk with the CFQ I/O scheduler. The operations per second for the same experiment is shown in Figure 6. From Figure 5, we see that the random-write workload without direct I/O for the SMR disk is 4% lower than the regular disk with the noop I/O scheduler. Furthermore, the regular disk with the

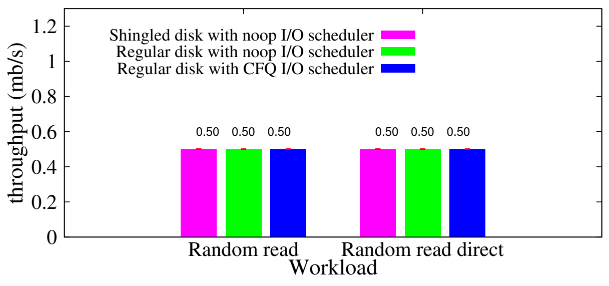

Figure 3: Throughput plot for micro benchmark workloads of random read with and without direct I/O run using Filebench

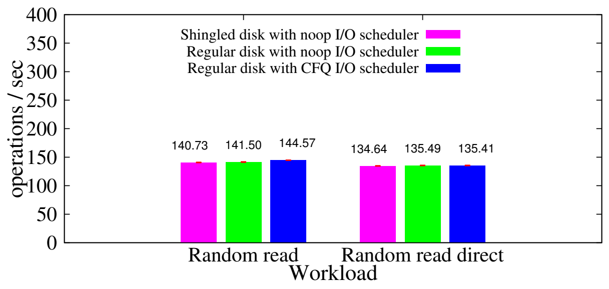

Figure 4: Operations per sec plot for micro benchmark workloads of random read with and without direct I/O run using Filebench

CFQ I/O scheduler has a throughput of 0.5% lower than the noop I/O scheduler. Random writes with the direct I/O workload has its throughput reduced by 92% as compared to random write without direct I/O. Among the random write with direct I/O, a similar pattern for throughput was seen as the random write without direct I/O. The throughput of random write direct workload on the SMR disk is seen to be 15% lower than that of regular disk for the same workload from Figure 5. This might be because of firmware modification which might have disabled disk caching to enforce sequential write.

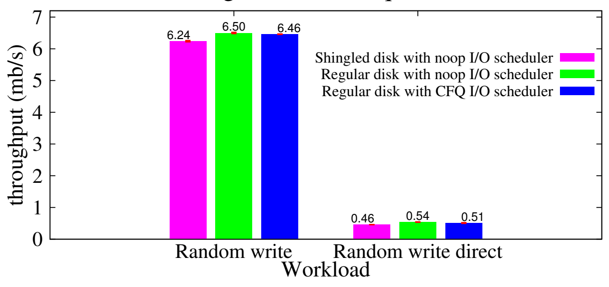

Figure 5: Throughput plot for micro benchmark workloads of random write with and without direct I/O run using Filebench

Figure 7 depicts the comparison of throughput of sequential access workloads for the SMR disk with the noop I/O scheduler, the regular disk with the noop I/O scheduler and the regular disk with the CFQ I/O scheduler. The operations per second for the same experiment is shown in Figure 8.

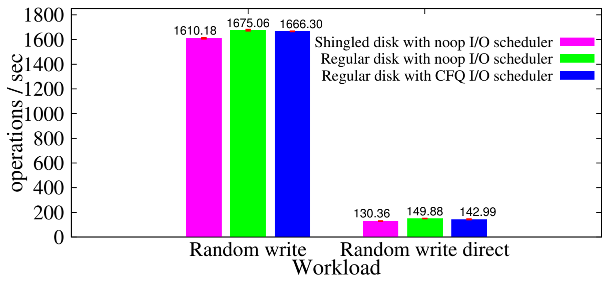

Figure 6: Operations per sec plot for micro benchmark workloads of random write with and without direct I/O run using Filebench

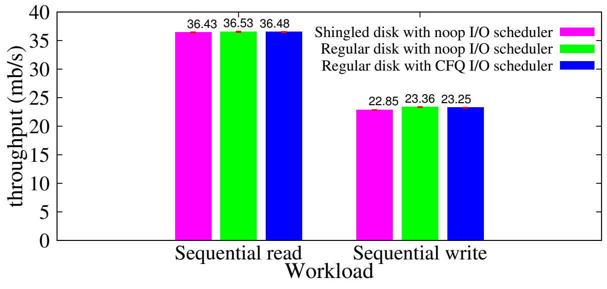

Figure 7: Throughput plot for micro benchmark workloads of sequential read and write run using Filebench

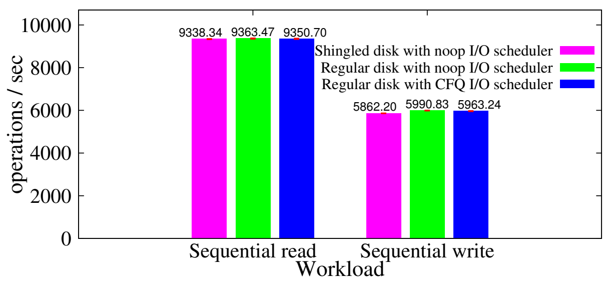

Figure 8: Operations per second plot for micro benchmark workloads of sequential read and write run using Filebench

The sequential access workloads from Figure 7 show a considerable increase in the throughput as compared to the random access workloads in Figure 3 and Figure 5. The pattern within these workloads for the SMR disk, the regular disk with the noop I/O scheduler, and the regular disk with the CFQ as I/O scheduler follow the same pattern as that of random access.

Evaluation summary. In sum, the LTP results were all good. There were no functional discrepancies due to the SMR disk for the LTP tests. The Filebench results showed a maximum drop in throughput of about 15% for random write direct workload. For all other workloads evaluated, the drop in throughput was less than 5%.

5 Conclusions

The LTP tests confirm that Nilfs2 on the SMR disk is stable and can be used for normal workloads. Overall there were no errors observed for normal working conditions.

The Filebench workloads on a whole indicate that read performance remains almost same for both the regular disk and the SMR disk. However, there is a drop in write performance in the SMR disk, believed to be due to firmware modification. The sequential write workload had a drop in throughput of 2%. The random write workload had a drop in throughput of 4%. The random write direct workload had the highest decrease in throughput of about 15%. The decreases in throughput for webserver, mailserver, and fileserver workloads were 1.3%, 6.3%, and 4.9%, respectively.

Future Work. The results from webserver workload showed an improvement in throughput for the CFQ I/O scheduler, compared to the noop I/O scheduler. We believe that the reason is due to the merging of multiple I/O requests by the scheduler. This needs to be verified by comparing the results of a simpler workload like reading from a raw device with I/O read requests within a smaller LBA range.

Garbage collection is one of the main costs in any log-structured file system like Nilfs2. The performance of the file system when garbage collection is running in parallel should be determined to account for performance when the disk is nearly full. When Nilfs2's configuration was modified to run garbage collection continuously on the regular disk, there were kernel errors observed under heavy workload spanning four times the available main memory. These errors need to be investigated in more detail and fixed to make Nilfs2 in Linux more stable while garbage collection is done under heavy load conditions. One possibility is to try and raise the Nilfs2's over-provisioning value of 5% and see if it improves GC stability. Another possibility is to increase the number of Nilfs2's superblock zones from two (first and last zone) to, say, ten—and then distribute them uniformly across the drive. This could reduce latency in accessing a more localized super-block, but increase the cost of synchronization of all super-block replicas.

Another future-work item is to explore the trade-off in zone sizes. The zone sizes in our current prototypes are likely the smallest capacity zone size that are practical with SMR drives. A smaller zone size reduces overall capacity through over-provisioning, but improves performance thanks to shorter garbage collection times. Larger zones will maximize capacity, but take more time to garbage collect. The SMR performance given a specific zone size will therefore depend on the workload in question. Selecting the right zone size native to the SMR is therefore key: this is a function performed by the manufacturer in the factory, and cannot be easily (if at all) changed once the drive is released to the consumer.

Finally, as this is an ongoing project with an evolving technology, we continue to investigate newer generations of SMR drives. We plan to identify specific causes of overheads and propose solutions (e.g., turning on/off check-pointing and other firmware features).

Acknowledgments. Western Digital provided SMR prototypes and support. Vasily Tarasov and Ming Chen helped set up and provided valuable feedback.

References

[1] Y. Cassuto, M. A. A. Sanvido, C. Guyot, D. R. Hall, and Z. Z. B. Indirection systems for shingled-recording disk drives. In *Proceedings of the International IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[2] Filebench. `http://filebench.sf.net`.

[3] Seagate. Breaking capacity barriers with Seagate shingled magnetic recording. `www.seagate.com/tech-insights/breaking-areal-density-barriers-with-seagate-smr-master-ti/`, 2013.

[4] Subrata Modak. Linux Test Project (LTP), 2009. `http://ltp.sourceforge.net/`.

[5] Ikuya Tagawa and Mason Williams. High density data-storage using shingle-write. In *Proceedings of the IEEE International Magnetics Conference*, 2009.

[6] Roger Wood. Shingled magnetic recording and two-dimensional magnetic recording. In *Proceedings of the IEEE IEEE Magnetics Society Santa Clara Valley Chapter*, 2010. `www.ewh.ieee.org/r6/scv/mag/MtgSum/Meeting2010_10_Presentation.pdf`.

[7] C. P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok. Auto-pilot: A Platform for System Software Benchmarking. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, 2005.

[8] A. Yoshiji, R. Konishi, K. Sato, H. Hifumi, Y. Tamura, S. Kihara, and S. Moriai. Nilfs, 2009.