Performance Analysis of File Systems: A Comparison of File Access time and I/O Operations

Moro Bamber, Sam Lilly

***Abstract*— File Systems (FS) are a crucial component in an Operating System (OS). Benchmarking file system performance is as important as ever as we move into the age of big data. Whether it's Microcostf NTFS, or the widely used ext4 system on Linux distributions, understanding their performance is essential in this day and age. Previous studies have shown that benchmarking a file system is more complicated than one may think. We conducted file system benchmarking experiments on both a Oracle Virtual Machine and an NVMe SSD. Part of the experiments were conducted using a micro-workload written for a program called Filebench. We then used a program called Iozone for a more system wide study of read and write operations, simulating a macro wordload. We leveraged a program called Geriatrix that ages file systems to simulate a more real world environment. On the Oracle virtual machine we found that on a micro workload, the fs f2fs was the fastest, but for the macro workload the fs xfs performed the best. Benchmarks run using ext4 on the SSD obtained similar results, exposing slight overheads in virtualization.**

***Index Terms*— File System, OS, Iozone, Filebench, Geriatrix**

INTRODUCTION

In Operating Systems (OSes), the File System plays a pivotal role in read and write access as well as movement and storage of the contents of the machine. In modern computing, speed is of the utmost importance. I/O is a component to an OS that should function as fast as possible to reduce blocking states for processes. File systems play a crucial role in internet infrastructure, and understanding how they perform is important [1]. In this paper we look at different file systems and how they perform in different scenarios. Using tools developed specifically for file system benchmarking we determined key insights into the speed of different file systems as well as examining some effects memory has on throughput speeds. We examined criteria for a good benchmarking system as stated by Kahanwal and Singe in [2]. With this criteria we selected filebench and iozone as benchmarking programs used in this study. We aimed for reproducibility of each benchmark with results that reflected real life workloads. Our goal was to compare file systems and create tests to provide important metrics regarding general throughput. Additionally, we wanted to attempt to recognize any latencies or overheads given by our procedures. As part of our study, we examined workloads on a virtual machine. We saw this as an important aspect to the study as more and more systems are running on virtual machines [3]. The remainder of the study was done using a physical computer. To simulate an accurate environment, Geriatrix was used to age the file system and revert to the same state for each benchmark.

FILEBENCH

We used the program Filebench to test our microworkloads. Filebench was chosen due to its flexibility in creating tests. While it comes with predfiended workloads, we tweaked them to fit parameters specific to the OSes and machines we were using. Defining workloads in Filebench is easy due to its Workload Model Language (WML) which allows the user to easily define specific workloads. For our tests we used a minute as the timespan for the test which is the default for Filebench. As with most experiments, a test should be run several times, this is the same with Filebench. Iozone does not have a set runtime. It does all the tests for different read and write operations and outputs the results once finished.

IOZONE

Iozone was the tool we used for our macro workload. Iozone is a powerful tool that generates bandwidth data among other metrics for the user after running a more comprehensive and larger benchmark test for the file system. Iozone is run from a C file and has options to generate various types of graphs. We used Iozone in the automatic mode which produces output that covers all test file operations (Read, write, re-write, re-read, etc.) but for this study we were only interested in read and write. As with Filebench we used a warm cache and aimed to have the machine in an identical state to get reproducible results. Using the data generated from Iozone we were also able to gain insight into file system performance.

GERIATRIX

Geriatrix is an open source file-system aging application that generates reproducible images. It includes definable parameters that allow for customizable environments to be made. The throughput of SSDs are greatly impacted depending on the aging of the file system [1]. Since an NVMe SSD was used for tests, Geriatrix gave a more real world environment to be simulated. We chose Geriatrix opposed to similar file-system aging applications as it induces fragmentation to both free space and allocated files.

RELATED WORKS

File system benchmarking has always been relevant in the field of computer science. We recognize the difficulties in performing file system benchmarking and overhead caused by these programs highlighted by Tarasov et al. [4]. Overcoming the nuances of benchmarking is difficult, and better tools for the task are yet to be developed. We also recognize the variability in benchmarking described by Cao et al [5]. Work has been done in this field to improve, and understand how to better benchmark, a survey of this landscape was conducted by Traeger and Zadok back in 2007 [6]. Improvements to benchmarking systems since this survey have been made. Zadok and Tarasov - a co-author and [4] and [5] - started contributing to FIlebench after these surveys and improved the program. They wrote the guide [7] to Filebench used to run the tests in this study. In this study we seek to build on prior benchmarking work while also doing tests on hardware within reach to the average computer owning person.

Methods

Using both a macro and micro file system benchmarking tool was critical to our study. Macro-benchmarks run multiple operations to provide the performance of a larger workload. The micro-benchmarks run fewer operations, but allow specific parts of systems to be isolated. These combined results obtain more relevant information for a real workload. For consistency and reproducibility, the file system was aged using Geratrix to produce a realistic file system. Each test was run on the same hardware, with the specifications listed in the Appendix. To further reflect real life systems, we used a warm cache. Systems generally operate after already undergoing activity, therefore a cold cache approach would give undesirable results. Achieving a warm cache was done by running the same test a consistent amount of times and discarding the first result. Benchmarking was run across four different file systems using a virtual machine, then conducted using a single file system on a physical computer.

VIRTUAL MACHINE METHODS

We used the Oracle VM VirtualBox to create four virtual machine's (VM's) with the standard Arch Linux image file[8]. Each VM was configured to use a different file system. The file systems used were btrfs, ext4, xfs, and f2fs. The memory for all VM's was set to 10 GB, and allowed for four processors. Using the archinstall script, we configured the machine to run the GNOME desktop environment for ease of use. Once the VM's were set up, filebench and iozone were installed. Using the command line, we did several runs of the filebench workload to warm up the cache. Once the cache was warm, we ran the test workload five times per file system to achieve results. The workload in question used four threads, with 1000 files. Each file was 1 KB in size and upon start of the test, the memory was preallocated. The workload consisted of three operations or 'flowops'. One to open a file, read the file, then close the file. We let this test run for 60 seconds. To control for processing speed, all tests were done with no other applications open except the terminal. On the host machine, Oracle VM was the only application opened to not use too much CPU power.

PHYSICAL COMPUTER METHODS

Experiments run on the physical computer were done using the 6.8.2 kernel version of Arch Linux. All benchmarks were run on the root partition using the ext4 filesystem. This filesystem was chosen as it remains the most popular and default among Linux distributions. The filesystem was aged using Geriatrix with a 860 GB image, allowing 120 GB remaining space for benchmarking to be conducted. Geriatrix had reached convergence at 50 iterations, executing a 43 TB workload. This was done using the Wang-OS profile, as it had the oldest age distribution among profiles [8]. After any benchmarking had been completed, the system was restored to its previous image to ensure consistent results. Experiments were conducted using a modified workload of the previous section. Filebench authors found a workload should be at least two times the size of system memory to ensure sufficient I/O activity [7]. To better fit system parameters, the size of each file was changed to 120 KB, and the number of threads was increased to 12. With a warm cache, each test was run 5 times with 600 seconds. Each filebench workload was run for 10 minutes to ensure a consistently warm cache and results given from a stable state. Iozone benchmarks were run using a maximum 64 GB file size on write/read tests for an accurate comparison to filebench benchmarks. The cache size was set to 32 MB with a default cache line size of 1 MB. These parameters were changed for a benchmark discussed in a further section.

RESULTS

RESULTS FOR VM'S

The results from the filebench tests for the virtual machine running Arch Linux yielded similar results. Though, f2fs was faster in both operations per second (ops/s), and read/write megabits per second (rd/wr mb/s).

fs	ops/s	rd/wr mb/s
btrfs	1717265	559
ext4	1774885	570.84
xfs	1747567	568.88
f2fs	1822342	593.22

Table 1 - VM Filebench Results

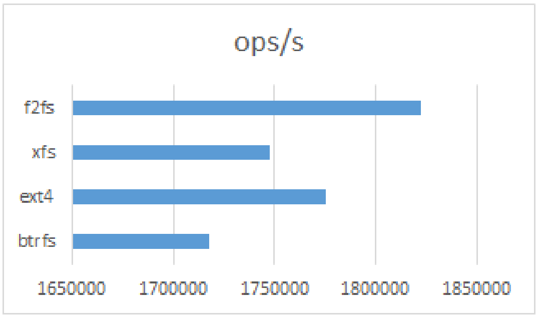

Figure 1 - VM Filebench ops/s

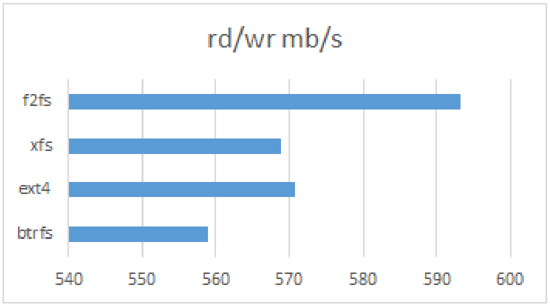

Figure 2 - VM Filebench read/write mb/s

VM IOZONE RESULTS

Iozone generates statistics for the performance of many different file operations. Among which include write, read, re-write, re-read, random-read, random-write, and backward read. For this experiment, we will only be concerned with the performance of reading and writing.

For the Iozone writing test, xfs was the fastest with an average of 3973171 KB/s over all record and file sizes. Second fastest was btrfs at 1541660, third was f2fs at 1494868 and last was ext4 at 1027459. xfs performed extremely well with large record sizes and large file sizes with a maximum occurring at a file size of 524288 KB and a record size of 4096 KB.

For the reading tests xfs once again outclassed the other file sytstems. Its average read in kb/s was 8608748, the next closest was f2fs at 3108705 KB/s. btrfs and ext4 were 2844643 KB/s and 2737791 KB/s respectively. For the fastest - xfs - it performed the best in the low file size and medium size record and performed the lowest in the low record size, low file size category. This trend occurred for all file systems tested.

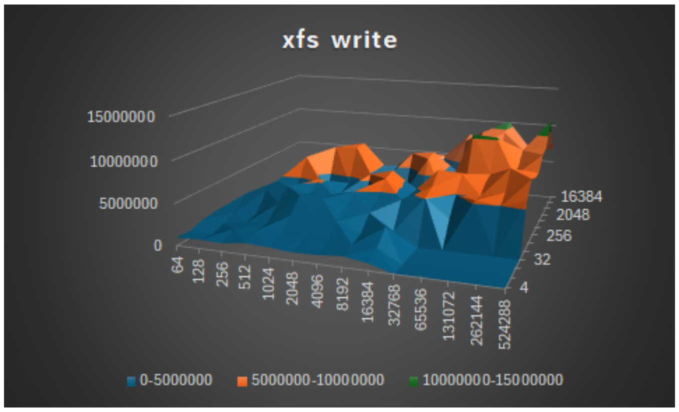

Figure 3 - xfs write VM. Scale 0 - 1.5E07

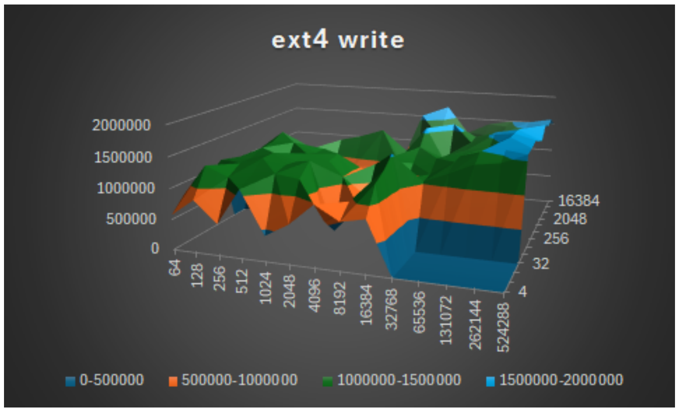

Figure 4 - ext4 write VM. Scale 0 - 2.0E06

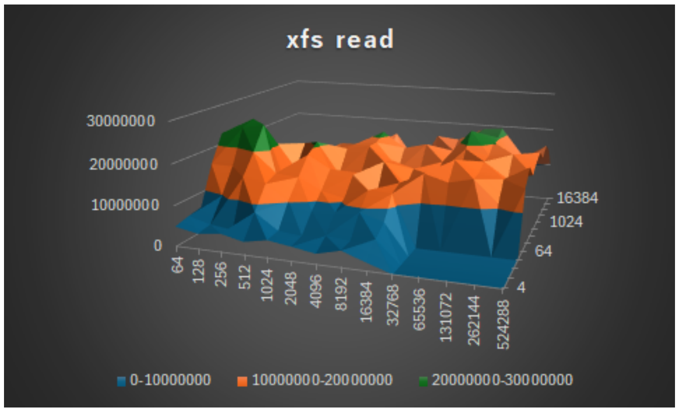

Figure 5 - xfs read VM. Scale 0-3.0E07

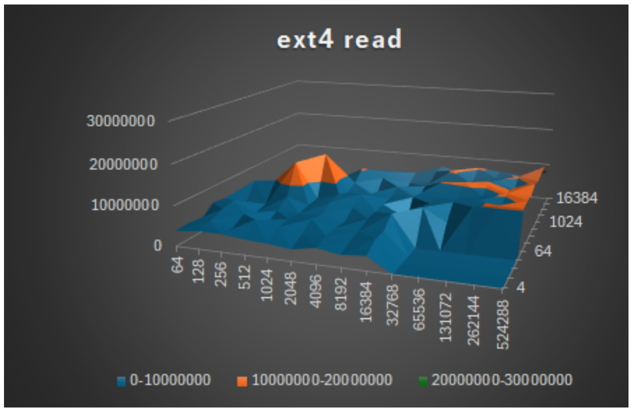

Figure 6 - ext4 read VM. Scale 0-3.0E07

PHYSICAL COMPUTER RESULTS

Filebench gave an average read/write speed of 590.4 MB/s on the ext4 file system installed on the physical computer. These speeds are only 6.4% speed of the iozone benchmark performing the same operation. This discrepancy is expected and will be discussed in the next section.

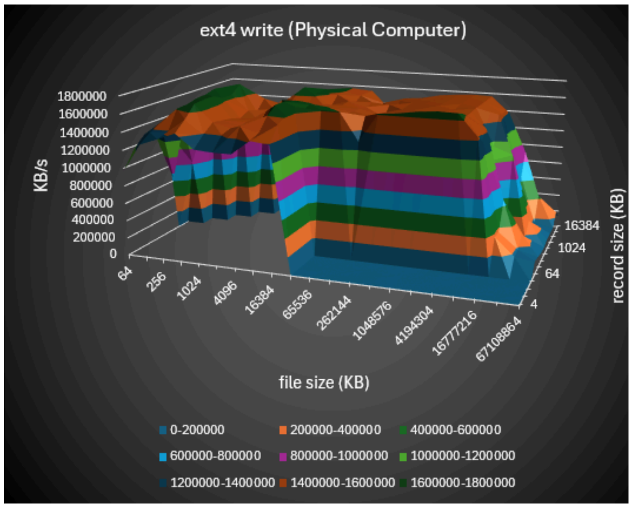

Figure 7 - ext4 write (Physical Computer)

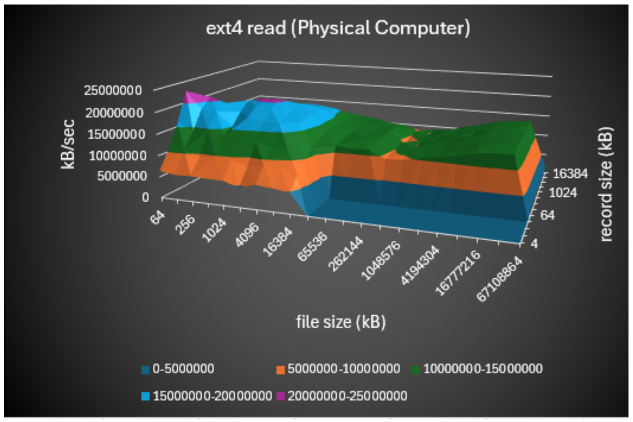

Figure 8 - ext4 read (Physical Computer)

For all iozone graphs, any area indicating 0 are non-measured areas. In order to save excessive time, file sizes greater than 32 MB are not measured when the requested record size is less than 64 KB. On the physical computer running the iozone benchmark, ext4 achieved an average write speed of 1466577 KB/s or 1466 MB/s. Write operations completed with greater than 16 GB file size decline greatly in throughput, with files greater than 64 GB averaging 1384 MB/s. The read operations of ext4 on the physical computer achieved an average speed of 11109571 KB/s or 11109 MB/s. Read operations with a file size exceeding 64 GB had a speed of only 1583 MB/s.

DISCUSSION

For the VM Arch Linux results, f2fs was the fastest in both metrics presented. It could do more ops/s in 60s than the others and it could read and write faster as well. More variance in tests would have been better to do, since we only tested one type of workload, but for the purposes of examining both micro and macro aspects of a file system, we believe it worked well. Using Iozone did not conclude that f2fs was faster on a macro level as well as a micro level. It did well, but not compared to xfs. xfs completely outperformed the other file systems, and by wide margins. We can also see that there is not necessarily a correlation between write speed and read speed, for example, btrfs was slightly faster than f2fs for writing, but f2fs was considerably faster at reading. From the results obtained from the VM's we tested, we can say that f2fs worked better on a micro level, and xfs worked best on a macro level.

In both the VM and physical computer results, filebench operated at significantly slower speeds than iozone. This is expected due to the nature of the benchmarks and the workloads executed. The intent of filebench is to provide an accurate workload simulating different pattern access of files with more threads than iozone, even with the same operations. Additionally, taking the average of read and write speeds from iozone results in a measure of CPU cache speeds rather than I/O performance. For the physical computer, once file size exceeds that of system memory (64 GB) the speeds become more indicative of actual I/O performance. These effects on the iozone benchmarks can be seen looking at the highest throughputs in the graphs. Take for example Figure 8, the highest peak at 64 KB shows the effect of the L1 cache on throughput speeds. Following the next peaks show the L2 and L3 cache effects until operations require the buffer cache. Comparing the graphs of iozone ext4 write benchmarks for the VM and physical computer, we can make some inferences for the difference in performance. The higher throughput with the smallest file and record size of the physical computer could indicate better utilization of the CPU cache than that of the virtual machine. Additionally, the dips throughout Figure 4 could come from latencies intrinsic with the hypervisor layer.

CONCLUSION

An important aspect of file system performance is the environment. Results indicated some overheads resulting from VM benchmarking, however a more accurate system comparison would be required for quantitative conclusions. Based on VM benchmarks xfs has the highest read/write performance specifically for larger file and record sizes. However, under a smaller yet realistic workload f2fs performs faster. Benchmarks on the physical computer and VM show stability of the ext4 file system valid for average users. Benchmarks of xfs indicate a suitable performance for database enterprises or server environments, which include workload sizes magnitudes larger than average. The f2fs file system proved suitable for both applications. Therefore, it is important to correlate the purpose of the machine to the requirements of the file system.

REFERENCES

[1] Chen M. et al. Newer is Sometimes Better: An Evaluation of NFSv4.1. June, 2015. *ACM Sigmetrics Conference*

[2] Kahanwal, B, Singh, T. P. Towards the Framework of the File Systems Performance Evaluation Techniques and the Taxonomy of Replay Traces

[3] Tarasov et al. Virtual Machine Workloads: The Case for New Benchmarks for NAS. February, 2013. 13th USENIX Conference on File and Storage Technologies

[4] Tarasov et al. Benchmarking File System Benchmarking: It *IS* Rocket Science. May, 2011. *13th USENIX Workshop in Hot Topics in Operating Systems*.

[5] Cao, Zhen, and Vasily Tarasov. "On the Performance Variation in Modern Storage Stacks." *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*, 2017, pp. 329-343, https://www.fsl.cs.sunysb.edu/docs/evos/evos-instability-fast17.pdf.

[6] Traeger, A., and E. Zadok. "Notes on a Nine Year Study of File System and Storage Benchmarking." *Byte and Switch*, 2009. Accessed www.byteandswitch.com/storage/storage-management/notes-on-a-nine-year-study-of-file-system-and-storage-benchmarking.php

[7] Tarasov, Vasily, and Erez Zadok. "{Filebench}: A Flexible Framework for File System Benchmarking." *;login: The USENIX Magazine*, vol. 41, no. 1, 2016, pp. 6-12, https://www.usenix.org/publications/login/.

[8] Kadekodi et al. Geriatrix: Aging what you see and what you don't see. A file system aging approach for modern storage systems. July, 2018. In USENIX Annual Technial Conference '18.

[9] Arch Linux Image File. https://archlinux.org/download/

Appendix

filebench Version 1.15.3

Custom Workload Source Code:

```
#Test Workload for OS

set $nthreads=4

define fileset
name="testF",entries=1000,filesize=1k,prealloc,path="/tmp"

 define process name="readerP",instances=1 {
   thread name="readerT",instances=$nthreads {
     flowop openfile name="openOP",filesetname="testF"
     flowop readwholefile name="readOP",filename="testF"
     flowop closefile name="closeOP"
   }
 }

run 60
```

iozone v. 3_414

Relevant physical computer specifications:

- *WD_BLACK SN770 NVMe SSD*
- *AMD Ryzen 5 3600X CPU*
- *ArchLinux Kernel Version 6.8.2*

Member Contributions

Moro Bamber

- Most of Abstract
- Most of the Introduction
- Filebench introduction
- Iozone introduction
- Related Works
- Virtual Machine Methods
- Virtual Machine Results
- Discussion Related to Virtual Machines

Sam Lilly

- Rest of most
- Geriatrix introduction
- Methods
- Physical Machine Methods

- Physical Machine Results
- Second part of discussion
- Conclusion

We feel like we did equal work