Containerized Filesystems: A Performance Investigation of Containers across Filesystems

Jia R. Wu
University of Waterloo

Jichen Zhao
University of Waterloo

Abstract

Container as a Service (CaaS) platforms are becoming increasingly popular with both personal and enterprise applications [1]. In all virtualisation services, virtual machines for example, containers impose some run-time overhead to applications which run inside of them. In this paper, we are looking at the overhead filesystems experience when running in containers and how different filesystems may suffer different level of overhead. We attempt to profile the performance of three separate filesystems, ext4, reiserfs, and xfs running on top of Arch Linux.

1 Introduction

Filesystems are the implementation of specific rules and principles governing how data is stored and retrieved on storage medium. At the time of writing, there are multiple various filesystems supported on the Linux operating system including ext4, reiserfs, xfs and many more. These filesystems all contain central concepts such as blocks and inodes, but differ subtly in their implementations [3].

Containers, also known as standardized units of software, are executables built ontop of a kernel. These containers offer a standardized execution environment agnostic to the environment the container is installed on. In this paper, we chose to examine Docker, a popular and free platform for creating and running containers [4]. Docker has been championed by researchers as a platform which allows for reproducible research [2]. We are interested in researching whether or not container based filesystem operations incur a significant performance overhead relative to native filesystem operations.

We make the following contributions:

- A comparison of ext4, reiserfs and xfs filesystems using Filebench and IOzone.
- An investigation of filesystem performance in containers versus native linux.

2 Methods

In this section, we discuss the setup of our experiments. We do not attempt to re-invent the wheel by writing adhoc benchmarks. Instead, we leverage existing benchmarks such as Filebench and IOzone.

2.1 Environment

For this project, we were using Arch linux distribution, version 4.13.12-1-ARCH, running on a quad-core AMD A8-7600 Radeon R7 with 4 gigabytes of installed memory. The disk we were using was a WD 500GB, 7200rpm hard drive with 32MB cache. We chose Arch over other alternatives mainly because it is lightweight, meaning that there will not be as many daemons competing with our benchmarks for resources.

2.2 Docker

Docker version 17.06.2-ce, API version 1.3, Go version go1.8.3, and git commit cec0b72 were used in this project. The base container was pulled from base/archlinux. All relevant code to regenerate the docker container and environment can be found in Section 7.

2.3 IOzone

IOzone is an open source file system benchmark tool. It generates and measures a variety of file operations in a ”brute force” way. It tests 14 operations including sequential IO and random IO. The command we were using is : ./iozone -a -R -g 3G. The ”-a” option tells IOzone to run all tests. ”-g 3G” tells IOzone try files with size up to 3GB. We are limiting the file size to be no grater than

3GB because anything bigger than that runs extremely slow and even crashes the system. This should not be a thread to validity since files bigger than 3GB are not that common. What IOzone does on this command is that for each size from 64KB to 3GB, that is power of 2 (64KB, 128KB, etc.), it divides the files into chunks and perform the requests one after another. It also uses different chunk sizes from starting from 4KB up to the size of the file.

2.4 Filebench

Filebench, a popular filesystem and storage benchmark for I/O profiling is used to profile our three filesystems, as it permits custom and scalable workloads. Unlike Iozone, Filebench can run workloads that mimic the real world situations. We utilized five workloads which shipped with Filebench: FILESERVER, VARMAIL, VIDEOSERVER, WEBPROXY and WEBSERVER. All scripts were executed with sudo privileges.

Each of these benchmarks were called 5 times after a fresh installation of the filesystem. Workloads generate temporary files prior to execution. Temporary files were discarded with "rm -rf" after each benchmark prior to running the next benchmark to clear the cache. The run-time of Filebench benchmarks were fixed at 300 seconds with the exception of WEBPROXY which was fixed at 60 seconds. This was to mitigate any ramp-up effects that might occur with our HDD. We had to disable address space layout randomization by setting randomize_va_space=0 in /proc/sys/kernel so that Filebench would execute properly and give consistent results. Unless explicitly mentioned, the benchmarks use the predefined settings given by the authors[1].

2.4.1 FILESERVER

This workload creates a directory tree, and calls a sequence of creates, deletes, appends, reads and writes on various files. The authors of Filebench describe this workload as being similar to SPECsfs. We configured the FILESERVER workload to output files of 1.2GB in size, with one flow for each stat, delete, read, close, append, open, close, write, and create operations.

2.4.2 VARMAIL

In this benchmark, multiple operations of create-append-sync, read-append-sync, read and delete are run in a single directory. The purpose of these operations are to simulate I/O operations on a mail server.

2.4.3 VIDEOSERVER

This workload simulates a videoserver by serving video files from one directory and caching videos in a second directory. Videos were configured to be 3.2GB in size.

2.4.4 WEBPROXY

WEBPROXY simulated I/O on a web proxy server. Multiple files were created, written to, closed, and deleted in parallel. This benchmark was configured to run only for 60 seconds as 300 seconds would cause a segfault. The authors of Filebench are aware of this issue on Github.

2.4.5 WEBSERVER

This benchmark is similar to WEBPROXY, however it runs open-read-close operations on files in a directory tree and has an additional step of appending.

3 Results

3.1 IOzone

When running ext4, the file system's performance did not seem to be affected much when running in docker. Let's take the throughput of sequential writes for example. Figure 1 and Figure 2 are surface graphs that shows the throughput of sequential writes in KB/s of different file sizes and chunk sizes, on host directly and in docker respectively. The higher the the altitude of the surface, the higher the average throughput IOzone was able to run at that combination of file size and chunk size. As illustrated by the figures, the tips of both surfaces were at about the same combination of file size and chunk size. It might not be clear on the graphs but from the data, both peak at approximately 85MB/s.

It should be clear that the overall shape of the two graphs are very similar, except that in Figure 1 there are more peaks (yellow area) than in Figure 2 because of the small but noticeable overhead of running inside docker. That means that the overhead of file accessing operations executed from inside the container is constant regardless of what file size someone is working with and what chunk size a specific application would prefer to use. Similar conclusions are made about other operations including random writes, sequential read, and random read.

For reiserfs and xfs, the results were very similar: running in docker imposes a small but noticeable overhead on the bandwidth for all file operations(random IO and sequential IO). As we compare the results, we found that the bandwidth overhead for all operations

is the same across all three file systems. From these results, we can conclude that no one filesystem has a higher performance overhead than the other two when running inside docker. We also discover that in IOzone benchmarks, the bandwidth of xfs is greater than that of ext4, which is in turn greater than reiserfs.

Docker imposes a small but noticeable overhead for file access operations equally across all three file systems.

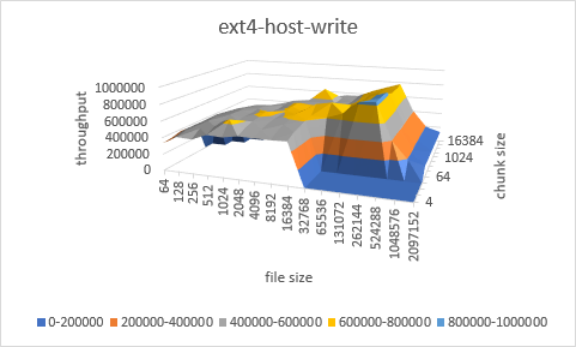

Figure 1: Native Sequential Write(100bytes/s)

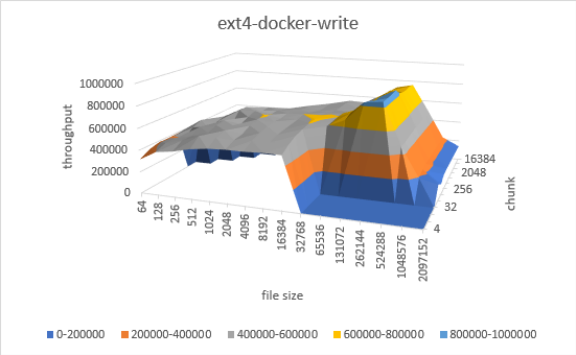

Figure 2: Docker Sequential Write(100bytes/s)

3.2 Filebench

Figures 3 & 4 both depict the results Filebench workloasd on each filesystem. Figures 5 & 6 are relative performance graphs of ext4, reiserfs and xfs on Arch Linux and Docker respectively. As illustrated by these figures, ext4 outperforms reiserfs across the board, and outperforms xfs with the exception of the WEBSERVER benchmark. Since ext4 is the highest performing filesystem, results are scaled to it in the relative performance graphs. Values less than 1 mean that the throughput on those filesystems performed worse than on ext4.

Filebench results suggest that in general, ext4 has a higher I/O throughput relative to reiserfs and xfs.

When examining the performance of Filebench benchmarks in the Docker container, the conclusions are identical for ext4. When comparing reiserfs and xfs however, there are some interesting differences. The VARMAIL and VIDEOSERVER benchmarks are much more similar in terms of performance. In our experiments, we observed that some benchmarks actually ran better in the Docker environment versus native Arch Linux as highlighted by Table 1. Section **??** will discuss this phenomenon in detail.

Filebench benchmarks perform more consistently within Docker than they do on native Arch Linux.

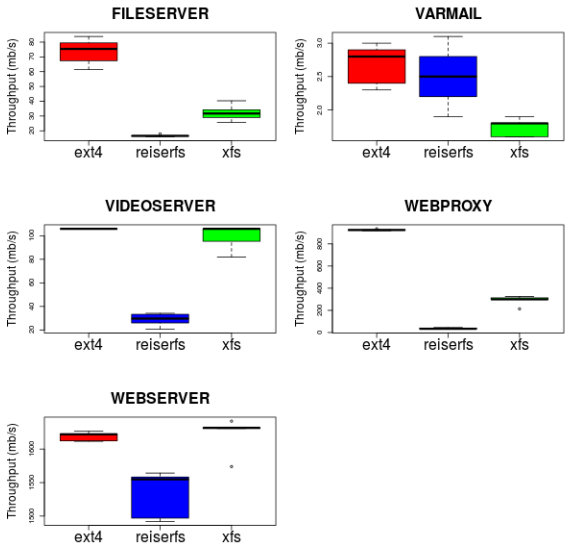

Figure 3: *Native Filebench performance per workload. Higher is better.*

	ext4	reiserfs	xfs
FILESERVER	1.01	0.95	1.29
VARMAIL	0.86	0.69	0.97
VIDEOSERVER	1.00	3.64	1.07
WEBPROXY	0.98	0.66	1.01
WEBSERVER	0.96	0.95	0.99

Table 1: *Benchmark results on Docker scaled to native Arch Linux. Lower values mean benchmarks performed worse on Docker. Higher values mean benchmarks performed better on Docker.*

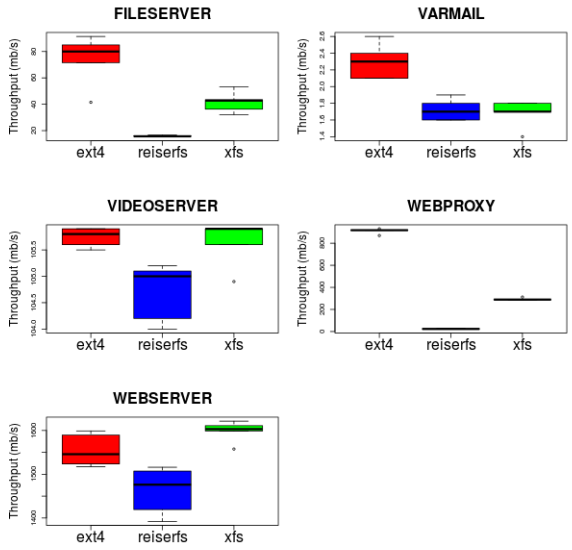

Figure 4: *Docker Filebench performance per workload. Higher is better.*

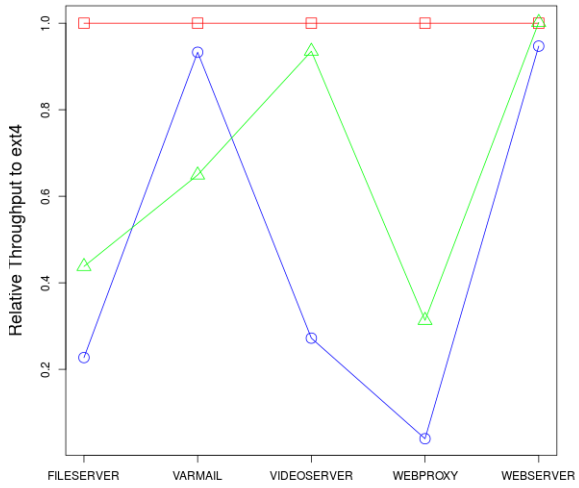

Figure 5: *Relative workload performance on native Arch Linux. Red squares are ext4, blue circles are reiserfs and green triangles are xfs. Throughput is scaled to ext4. Higher numbers mean higher throughput relative to ext4.*

4 Threats To Validity

Filebench is not fully representative of a filesystem's performance. Filebench primarily evaluates the I/O performance of a disk, and is not tuned to evaluate the caching and metadata dimensions of a disk [5]. Workloads specified in the benchmarks used the default values

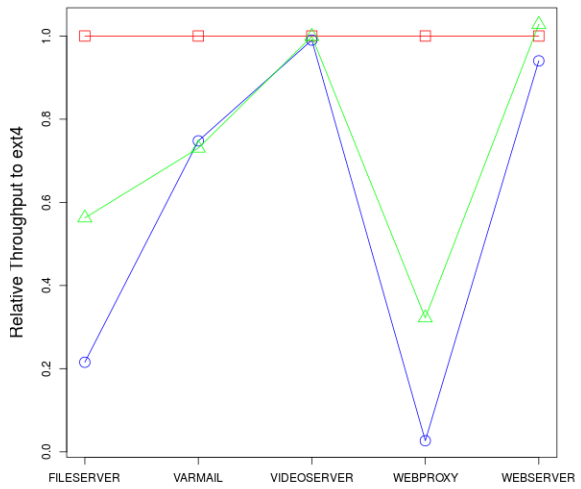

Figure 6: *Relative workload performance within Docker. Red squares are ext4, blue circles are reiserfs and green triangles are xfs. Throughput is scaled to ext4. Higher numbers mean higher throughput relative to ext4.*

that shipped with Filebench, and were not tweaked. These could have been adjusted to better reflect the capability of our server. We observed a peculiarity in our results, such as how VARMAIL and VIDEOSERVER results differ greatly in our Arch Linux environment, but are very similar in the Docker environment. Table 1 depicts the benchmark results of Docker scaled to native Arch Linux. Surprisingly, there are five benchmarks that run better in the Docker container than in native Arch Linux. These benchmarks are FILESERVER-ext4, FILESERVER-xfs, VIDEOSERVER-reiserfs, VIDEOSERVER-xfs, and WEBPROXY-xfs. These results are surprising as execution in a container should not outperform execution on the host machine. One reason why Docker results may be better than native Arch Linux is because of the isolation of the execution environment within Docker. Both scripts were run in privileged mode, however there could be some interference from other system processes during the native execution. This suggests that there is some unknown factor affecting our results. Thus, we refrain from making absolute claims that one filesystem is better than another, and only provide suggestions. Additionally, due to this discrepancy in execution between native Arch Linux and Docker, we do not perform any statistical tests comparing the performance difference of host execution vs container execution.

In IOzone results, we concluded that the overhead

of running in docker was approximately equal for all three file systems. This was drawn without solid statistical support as we had few samples. We would need more samples in order to have enough statistical power to determine whether or not the difference is significant. However, we believe that this conclusion is valid since the overhead was so small: less than 5MB/s of bandwidth reduction. This suggests that the overhead between docker and native linux is identical.

5 Future Work

Future work includes tuning the default Filebench workloads in order to stress our hardware with a representative workload. We only examined one dimension of filesytems, I/O performance, yet there are multiple others such as the on-disk performance, caching performance and even metadata performance. Inclusion of other benchmarking software could allow us to make more concrete claims about filesystems in general and their differences.

IOzone benchmarks could be run multiple times in Docker and native linux in order to obtain enough samples for a statistical test of difference between the two. This study could be repeated with the inclusion of other filsystems.

6 Acknowledgments

We thank Dr. Tim Brecht for providing guidance about the project such as recommending papers relevant to benchmarking filesystems.

7 Availability

All relevant scripts and data can be retireved from the following repository:

`https://github.com/JRWu/fall2017_cs854/`

Notes

[1]The predefined settings can be located here: https://github.com/filebench/filebench/wiki/Predefined-personalities

References

[1] ANDERSON, C. Docker [software engineering]. *IEEE Software 32*, 3 (2015), 102–c3.

[2] BOETTIGER, C. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review 49*, 1 (2015), 71–79.

[3] PROJECT, T. L. D. The linux system administrators guide 5.10.filesystems.

[4] RAD, B. B., BHATTI, H. J., AND AHMADI, M. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS) 17*, 3 (2017), 228.

[5] TARASOV, V., BHANAGE, S., ZADOK, E., AND SELTZER, M. Benchmarking file system benchmarking: It* is* rocket science. In *HotOS* (2011), vol. 13, pp. 1–5.